This is self-archiving version of an accepted article. DOI: 10.1016/j.swevo.2019.100627

Handling bound constraints in CMA-ES: an experimental study[☆]

Rafał Biedrzycki[*]

Institute of Computer Science, Warsaw University of Technology

Abstract

Bound constraints are the lower and upper limits defined for each coordinate of the solution. There are many methods to deal with them, but there is no clear guideline for which of them should be preferred. This paper is devoted to handling bound constraints in the Covariance Matrix Adaptation Evolution Strategy (CMA-ES) algorithm. It surveys 22 Bound Constraint Handling Methods (BCHMs). The experiments cover both unimodal and multimodal functions taken from the CEC 2017 and the BBOB benchmarks. The performance of CMA-ES was found to change when different BCHMs were used. The worst and the best BCHMs were identified. The results of CMA-ES with the best BCHM and restarts were compared on CEC 2017 with the results of recently published derivatives of Differential Evolution (DE).

Keywords: bound constraints, CMA-ES

1. Introduction

In the real vector space $\mathbb{R}^n$, it often happens that the search space is constrained to a certain subset $F \subset \mathbb{R}^n$. Each point from F will be called the *feasible point*, and the set F — the feasible set. A convenient standardized definition of set F is based on the set of constraint functions whose values are expected either to be lower than zero or to be equal to zero for every feasible point. Thus the constrained optimization problem is defined as

$$\mathbf{x} = \arg\min_{\mathbf{x} \in F} q(\mathbf{x}) \tag{1}$$

$$g_i(\mathbf{x}) \leq 0 \quad \text{for all } i = 1, \dots m \tag{2}$$

$$h_j(\mathbf{x}) = 0 \quad \text{for all } j = 1, \dots k \tag{3}$$

[*]Corresponding author

Email address: rbiedrzy@elka.pw.edu.pl (Rafał Biedrzycki)

where $q : \mathbb{R}^n \to R$ is the objective function to be minimized, $g_i : \mathbb{R}^n \to R$ are the inequality constraints and $h_j : \mathbb{R}^n \to R$ are the equality constraints.

Constraints are a mixed blessing for optimization methods. On the one hand, constraints often limit the hypervolume of the feasible set so that optimization methods can better approximate the solution. On the other hand, the introduction of constraints breaks the isotropy of the search space and thus it may disturb the action of optimization methods that are based on iterative transformations of solutions. In particular, evolutionary computation and other metaheuristics which generate new solutions in the vicinity of old solutions may suffer from anisotropy of neighborhood structures. Therefore, a number of constraint handling techniques that modify the action of the base algorithms have been introduced by scholars. Comprehensive studies on that issue can be found, e.g., in [1, 2, 3].

A special case of inequality constraints are *bound constraints*, where the feasible set $\mathbb{F} = [\mathbf{l}, \mathbf{u}]$ is a hyperrectangle in $\mathbb{R}^n$, where $\mathbf{l}$ defines the lower and $\mathbf{u}$ defines the upper bounds of the search space. That type of constraint can be found in many practical problems [4, 5, 6, 7], since in the physical world there are usually upper and lower limitations of mass, energy, length, area, etc. Bound constraints are also used in the majority of benchmark sets for global optimization, like BBOB [8] or the CEC family [9].

There are two distinctive features of bound constraints that facilitate their handling: the feasibility of a point can be checked after at most $2n$ comparisons of real numbers, and it is relatively easy to transform an infeasible point into a feasible one. Therefore, the strategies of bound constraint handling (bound constraint handling methods, BCHMs) are computationally less demanding in comparison to strategies handling the general form of constraints and, perhaps for that reason, bound constraints have not gained much attention from researchers. Nevertheless, there is a practical need to consider various BCHMs since coupling a certain optimization with various BCHMs results in different efficiency in performing optimization. This effect was illustrated in an empirical comparative study [10], where 17 types of BCHM were coupled with seven versions of Differential Evolution (DE). According to the results obtained for the CEC'2017 benchmark set [9], the choice of BCHM can influence both the dynamics of the optimization progress and the quality of final solutions obtained by DE. The study also revealed that the sensitivity to the choice of a BCHM depends on the DE version being considered.

For Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [11], a specialized BCHM based on additive quadratic penalty was defined [4]. In the addendum [12] to the aforementioned article, Hansen proposed a modified version of the original BCHM. Both methods were compared on an ellipsoid function. The updated method was modified further by Sakamoto and Akimoto [13], and examined on four objective functions.

Even though CMA-ES is an important method with hundreds of successful applications [14] and is still being developed, used or extended (e.g. [15, 16, 17]), there is no exhaustive comparative study of BCHMs coupled with it. This article attempts to fill this gap. It surveys 22 BCHMs on two types of tests. The first

Initialize population mean $\mathbf{o}$ using an initial solution
while stop condition not met **do**
  for all $i \in \{1, 2, ..., \lambda\}$ **do**
    $\mathbf{m}_i = \mathrm{N}(\mathbf{o}, \sigma^2 \mathrm{C})$
    if apply penalty **then**
      $\mathbf{x}_i \leftarrow \mathbf{m}_i$
      $q_i \leftarrow$ fitness($\mathbf{m}_i$)
    else if Lamarckian repair **then**
      $\mathbf{x}_i \leftarrow$ repair($\mathbf{m}_i$)
      $q_i \leftarrow$ objective($\mathbf{x}_i$)
    else if Darwinian repair **then**
      $\mathbf{x}_i \leftarrow \mathbf{m}_i$
      $q_i \leftarrow$ objective(repair($\mathbf{m}_i$))
    else if state aware repair **then**
      $\mathbf{x}_i \leftarrow$ repair($\mathbf{o}, \mathbf{m}_i$)
      $q_i \leftarrow$ objective($\mathbf{x}_i$)
    end if
  end for
  $\mathrm{P}_\mu \leftarrow$ the best μ individuals from the set of $\mathbf{x}_i$
  $\mathrm{C} \leftarrow$ update_C($\mathrm{C}, \mathrm{P}_\mu$); $\sigma \leftarrow$ update_σ($\sigma, \mathrm{P}_\mu, \mathbf{o}$); $\mathbf{o} \leftarrow$ mean(P_μ)
end while

Figure 1: Coupling considered BCHMs with CMA-ES algorithm

one is devoted to examining the efficiency of local optimization. The second one is aimed at characterizing global optimization efficiency and is based on the CEC 2017 benchmark set [9].

The article is composed in the following way. Section 2 briefly describes the CMA-ES algorithm with special emphasis on ways of coupling it with different types of BCHM. In Section 3, a set of 22 BCHMs is defined. Section 4 examines the influence of BCHMs on the efficiency of CMA-ES. Section 5 summarizes the observations and concludes the paper.

2. Optimization with bound constraints in CMA-ES

The CMA-ES algorithm [11] is a variant of an evolution strategy that is designed for solving difficult non-linear non-convex optimization problems in continuous domains [18]. It performs optimization in an n-dimensional space by iteratively sampling the search space. In CMA-ES, each i-th individual $\mathbf{m}_i$, called a mutant, is sampled from a multivariate normal distribution, with the shape defined by a covariance matrix C. The step size σ regulates mutation strength. At the beginning of the operation, the mean vector $\mathbf{o}$ is set to an initial solution, and this is updated at the end of each iteration. Each mutant can be infeasible. If this happens, several scenarios are possible, which are illustrated in Fig. 1.

Usually, in CMA-ES variants, several types of penalty methods are utilized, whereas in classical evolutionary algorithms and in the DE family, repair methods are more commonly used. Repair methods can be divided into two groups. In the Lamarckian approach, the repaired individual replaces the infeasible mutant, but in the Darwinian approach, the repaired individual is discarded after objective calculation. The last considered group of methods, which is called here *state aware methods*, also utilize information about the mean vector $\mathbf{o}$. After dealing with out-of-bounds mutants the objective function is calculated. Then, in the selection process, the best μ individuals survive. The survivors are used to update the covariance matrix C, step size σ and the mean vector $\mathbf{o}$.

Within this paper, a widely approved naming convention is used. The function to optimize is called the objective function, but CMA-ES optimizes the fitness function. In most cases, the fitness function is the same as the objective function, but when penalties are applied, the fitness function uses the objective function and other functions responsible for penalizing infeasible solutions.

3. Overview of bound constraint handling methods

All considered BCHMs can be assigned to four groups. *Repair methods* create a feasible solution using only the infeasible one and constraints. *Feasibility preserving* methods utilize knowledge of how mutation is performed, e.g., they can repeat mutation or place corrected individuals somewhere between the parent individual and the mutant. *Specialized methods* are applicable only for the CMA-ES algorithm. They use CMA-ES internal parameters. Finally, *penalty functions* do not replace infeasible individuals but add a penalty to their fitness values. Both *feasibility preserving* and *specialized* methods fall into the *state aware* repair group in Alg. 1.

3.1. Repair methods

A repair method is a mapping $r : \mathbb{X} \to \mathbb{F}$ which assigns a feasible individual to an infeasible one. The majority of repair methods work in a coordinate-wise fashion. The symbol $j = 1, \ldots, n$ will be used in formulas for the coordinate index.

The following list briefly summarizes definitions of repair methods reported in the literature. The naming convention used in this paper was adopted from [19] and expanded.

Reinitialization: the infeasible coordinate value is replaced by a random value ξ:

$$r(m_j) = \begin{cases} m_j & l_j \leq m_j \leq u_j \\ \xi & m_j < l_j \text{ or } m_j > u_j \end{cases} \tag{4}$$

Usually a uniform distribution in the interval $[l_j, u_j]$ is used for ξ. In the case of Differential Evolution (DE), Price observed [20] that this method is the most unbiased one, yet it can disrupt progress towards solutions that lie near the bounds.

Projection: the infeasible coordinate value of the solution is projected onto the violated bound.

$$r(m_j) = \begin{cases} m_j & l_j \leq m_j \leq u_j \\ l_j & m_j < l_j \\ u_j & m_j > u_j \end{cases} \tag{5}$$

This method is widely used in evolutionary algorithms, e.g. [21], because it is fast and easy to implement.

Reflection: the infeasible coordinate value of the solution is reflected back from the boundary by the amount of constraint violation. From the point of view of optimization algorithms, a fitness landscape outside the bound is a reflection of the original landscape.

$$r(m_j) = \begin{cases} m_j & l_j \leq m_j \leq u_j \\ 2l_j - m_j & m_j < l_j \\ 2u_j - m_j & m_j > u_j \end{cases} \tag{6}$$

This method was defined and used in DE, e.g. [22].

Wrapping: the infeasible coordinate value is shifted by the feasible interval $(u_j - l_j)$:

$$r(m_j) = \begin{cases} m_j & l_j \leq m_j \leq u_j \\ u_j + m_j - l_j & m_j < l_j \\ l_j + m_j - u_j & m_j > u_j \end{cases} \tag{7}$$

This approach may prove efficient when the optimization task is periodic in nature, e.g. in the design of digital filters [23].

Transformation: the transformation of the coordinate value starts before hitting the bound:

$$r(m_j) = \begin{cases} m_j & l_j + a_j^l \leq m_j \leq u_j - a_j^u \\ l_j + (m_j - (l_j - a_j^l))^2/4a_j^l & l_j - a_j^l \leq m_j < l_j + a_j^l \\ u_j - (m_j - (u_j + a_j^u))^2/4a_j^u & u_j - a_j^u < m_j \leq u_j + a_j^u \end{cases} \tag{8}$$

where $a_j^l = \min((u_j - l_j)/2, (1 + |l_j|)/20)$, $a_j^u = \min((u_j - l_j)/2, (1 + |u_j|)/20)$. The transformation is identity transform in about 90% of the feasible region. From $u_j - a_j^u$ to $u_j + a_j^u$ it is quadratic. When $m_j > u_j + a_j^u$, m_j is reflected (6) using $u_j + a_j^u$ as a bound. The reflected value is transformed according to (8), which results in periodic output of the transformation, with a period of $2(u_j - l_j + a_j^l + a_j^u)$. Analogical computations are performed when considering the lower bound. This is a default method used by the most sophisticated CMA-ES implementation in Python [24], and in the CMA-ES implementation in C [25].

Projection to midpoint: in contrast to a coordinate-wise repair, here the procedure consists of projecting the infeasible individual onto the boundary along the direction going through the midpoint of the feasible area.

$$r(\mathbf{m}) = (1 - \alpha) \cdot (\mathbf{l} + \mathbf{u})/2 + \alpha \cdot \mathbf{m} \tag{9}$$

where α is the largest value for which $l_j \leq r(m_j) \leq u_j$ for all $j = 1, \ldots, n$. This method was used in DE under the name "scaled mutant" [26].

3.2. Penalty functions

When using penalty functions, the infeasible individual is not changed, but its objective function value is modified. Traditionally, a penalty is added to the objective function value of the infeasible solution [27]. In practice, it may be impossible to compute the objective function value of the infeasible individual. Therefore, in engineering applications, e.g. [4], and some benchmarks, e.g. the Black Box Optimization Competition (BBComp) [28], an objective function is evaluated at the point that is repaired by the projection.

For the brevity of notation, let ψ denote the infeasible mutant.

Death penalty: the fitness of the infeasible individual is set to a large value Q that exceeds the fitness of any feasible individual

$$q_e(\psi) = Q \text{ s.t. } \forall \mathbf{y} \in \mathbb{F} \quad q(\mathbf{y}) < Q \tag{10}$$

This method was used, e.g., in Evolution Strategies [29]. According to [27], this simple method provides good quality results for some problems, but for other problems, it performs quite poorly. It was also observed that the standard deviation of results achieved with the use of this method was larger than that achieved for other methods. This method can be found in the literature [20] under the name "Brick Wall Penalty".

Additive quadratic penalty: the fitness of the infeasible individual is the sum of the objective function value of a repaired solution (usually by *projection*) and the squared values of constraint violations.

$$q_e(\psi) = q(r(\psi)) + \sum_{j:\psi_j<l_j} (l_j - \psi_j)^2 + \sum_{j:\psi_j>u_j} (\psi_j - u_j)^2 \tag{11}$$

The idea of a quadratic penalty is very old [27], but it is still used in practice.

Substitution quadratic penalty: the combination of the two aforementioned functions. The fitness function of the infeasible individual is not computed, which spares budget, but it is set to the sum of a large value Q that exceeds the objective function value of any feasible individual and the squared values of constraint violations.

$$q_e(\psi) = Q + \sum_{j:\psi_j<l_j} (l_j - \psi_j)^2 + \sum_{j:\psi_j>u_j} (\psi_j - u_j)^2 \tag{12}$$

where $\forall \mathbf{y} \in \mathbb{F} \quad q(\mathbf{y}) < Q$. This function realizes the idea of Deb [30] and Mezura-Montes [31].

Multiplicative quadratic penalty: the penalty value is the product of the objective function value of a repaired solution (usually by *projection*) and the sum of squared values of constraint violations.

$$q_e(\psi) = q(r(\psi)) \cdot \left(1 + \sum_{j:\psi_j<l_j} (l_j - \psi_j)^2 + \sum_{j:\psi_j>u_j} (\psi_j - u_j)^2\right) \tag{13}$$

This penalty method is used in the implementation of CMA-ES in the R language [32]. Note that the aforementioned penalty works reasonably well only when $q(\mathbf{x}) > 0$ for all $\mathbf{x}$.

3.3. Feasibility preserving

Rand base: the infeasible coordinate is set to a random location between the distribution mean and the active boundary.

$$r(m_j) = \begin{cases} m_j & l_j \leq m_j \leq u_j \\ U(l_j, o_j) & m_j < l_j \\ U(o_j, u_j) & m_j > u_j \end{cases} \tag{14}$$

where $U(\alpha, \beta)$ denotes the uniform random variate from the range $[\alpha, \beta]$. An analogical approach was applied with DE, but there it was named "Bounce-Back" [20].

Midpoint base: the average of the violated constraint and the distribution mean replaces the infeasible coordinate.

$$r(m_j) = \begin{cases} m_j & l_j \leq m_j \leq u_j \\ (l_j + o_j)/2 & m_j < l_j \\ (o_j + u_j)/2 & m_j > u_j \end{cases} \tag{15}$$

An analogical approach in DE was mentioned in the book [20] but it was not given a name.

Resampling: the mutation operator is repeated until a feasible individual is found. In some cases, this action can slow down the algorithm too much [19]. Therefore, after a predefined number of unsuccessful trials, a repair method is used (usually *projection*).

Conservative: the infeasible solution is replaced by the distribution mean $\mathbf{o}$ taken from CMA-ES:

$$r(\mathbf{m}) = \begin{cases} \mathbf{m} & \text{when } \mathbf{m} \in \mathbb{F} \\ \mathbf{o} & \text{when } \mathbf{m} \notin \mathbb{F} \end{cases} \tag{16}$$

This function was originally introduced for DE and was adopted for CMA-ES. In the case of DE, it was reported [19] that this approach is good for the Weierstrass function.

Projection to base: the infeasible individual is projected onto the boundary along the direction towards the distribution mean.

$$r(\mathbf{m}) = (1 - \alpha) \cdot \mathbf{o} + \alpha \cdot \mathbf{m} \tag{17}$$

where α is the largest value for which it holds $l_j \leq r(m_j) \leq u_j$ for all $j = 1, \ldots, n$. This method is similar to that used in GENOCOP III [33].

3.4. Specialized methods

The group of specialized methods contains BCHMs that are applicable only for the CMA-ES algorithm. These methods use the information extracted from C and σ. For brevity, only their most distinguishing features are discussed here. All the details can be found in the referenced papers.

The considered methods are as follows.

Weighted penalty – the idea was described in [4]. The fitness of the infeasible individual is the sum of the objective function value of a repaired solution (usually by *projection*) and the weighted squared values of constraint violations:

$$q_e(\psi) = q(r(\psi)) + 1/n \sum_{j=1}^{n} \left(\gamma_j \cdot (r(\psi_j) - \psi_j)^2 / \xi_j \right) \tag{18}$$

where $\xi_j = \exp\left(0.9 \left(\log\left(\mathrm{C}_{jj}\right) - (1/n) \sum_{k=1}^{n} \log\left(\mathrm{C}_{kk}\right)\right)\right)$ scales the coordinate-wise distance with respect to the covariance matrix. The weights are initialized by

$$\gamma_j = \frac{2\delta_f}{\sigma^2/n \sum_{k=1}^{n} \mathrm{C}_{kk}} \tag{19}$$

when distribution mean $\mathbf{o}$ is out-of-bounds. The δ_f is the median from the last $20+3n/\lambda$ generations of the interquartile range of unpenalized objective function values. The weights after initialization are increased when the distribution mean o_j is out-of-bounds according to

$$\gamma_j = \gamma_j 1.1^{\min(1, \mu_{eff}/(10n))} \tag{20}$$

where μ_{eff} is a CMA-ES parameter which is calculated during CMA-ES initialization.

Weighted penalty Matlab implementation – the official CMA-ES Matlab implementation [25] differs from the method described in [4]

$$q_e(\psi) = q(r(\psi)) + 1/n \sum_{j=1}^{n} \left(\gamma_j \cdot (r(\psi_j) - \psi_j)^2 \right) \tag{21}$$

There is no scaling by ξ and the weights are initialized by

$$\gamma_j = \frac{2\delta_f}{\sigma^2 n \mathrm{C}_{jj}} \tag{22}$$

when the distribution mean $\mathbf{o}$ is out-of-bounds.

The weights after initialization are increased according to:

$$\gamma_j = \gamma_j 1.2^{\min(1, \mu_{eff}/(10n))} \tag{23}$$

when $|o_j - r(o_j)| > 3 \max(1, \sqrt{n/\mu_{eff}}) \sigma \sqrt{C_{jj}}$, provided that the current mean is shifting deeper into infeasible space than the last mean.

Weighted penalty with decreasing weights – this method was defined in [12], which is an addendum to [4]. In comparison to *weighted penalty*, this method introduces the possibility of decreasing weights according to:

$$\gamma_j = \gamma_j \exp\left(-2/3\right)^{d_\gamma/2} \tag{24}$$

where $d_\gamma = \min\left(1, \mu_{eff}/\left(10n\right)\right)$. The γ_j value is decreased if $\gamma_j > 5\,\mathrm{median}(\delta L_\sigma)$, where δL_σ is δ_f divided by the denominator of (19). Additionally, (19) is replaced by $\gamma_j = 2\delta L_\sigma$, ξ_j are set to 1, and the method of increasing γ_j is also changed to:

$$\gamma_j = \gamma_j \exp\left(\tanh\left(\max\left(0, \delta o_j^\sigma - \delta_{th}\right)/3\right)\right)^{d_\gamma/2} \tag{25}$$

where δo_j^σ is the distance of the o_j from the nearest boundary divided by σC_{jj} and $\delta_{th} = 3\max(1, \sqrt{n}/\mu_{eff})$.
Weighted penalty with trimmed median – this method was defined in [13]. It changes *weighted penalty with decreasing weights* by increasing the speed of the reduction of the weights. In this method, (24) is replaced by:

$$\gamma_j = \gamma_j \min(1, 3 \cdot t/(1/n \sum_{k=1}^{n} \gamma_k)) \tag{26}$$

where t is a trimmed median which is calculated from a short history of δL_σ.

4. Experimental study

BCHMs may change positions of points after they have been generated by CMA-ES or they may change the fitness function values of infeasible points. Any change that has been introduced by a BCHM may, in turn, affect the positions or weights of points which are used in the update formulas for the covariance matrix and the step size multiplier. In this section, combinations of CMA-ES with all BCHMs listed in Section 3 are characterized by analyzing the results yielded by such combinations for various optimization tasks.

There are three series of analyses. The first series is aimed at characterizing the influence of BCHMs on the exploitation efficiency by comparing the results yielded by CMA-ES with and without constraints. The analysis was performed for unimodal objective functions which have a unique optimum inside the feasible area. Therefore, the bound constraints do not hide any other optimum but they only influence the convergence rate. For a perfect BCHM, the convergence rate should not be affected by a distance between the optimum and the boundary of the admissible area. For a realistic BCHM, the percentage of infeasible points would typically increase when the optimum gets closer to the boundary. This, in turn, would slow down the convergence rate. Therefore, in this analysis, each BCHM is characterized by the proportion of the number of fitness evaluations which are needed to reach a certain fitness value, with and without that BCHM. Such proportion is measured for different positions of

the optimum, starting from the middle point of the admissible area up to the boundary.

The second type of analysis deals with a more complicated case when the fitness function may be multimodal. There may be many local optima, inside and outside of the admissible area. The global optimum is located in the corner of the admissible hyperrectangle. This is the most difficult scenario since, in the vicinity of the global optimum, a vast majority of points generated by CMA-ES would be infeasible and BCHMs would be most frequently used. For that scenario, Empirical Cumulative Distribution Function (ECDF) curves were used to characterize the convergence rate of CMA-ES coupled with various BCHMs. The assumed visualization method was inspired by the methodology [34] used in the GECCO Workshop on Real-Parameter Black-Box Optimization Benchmarking (BBOB). In the presented approach, for each objective function and each dimension, a set of 51 fitness levels is defined. These levels are evenly distributed in the logarithmic scale and span the range from the median of the fitness values achieved after the first generation up to the best solution that was found at the end of the evolution. Each ECDF curve shows the portion of fitness levels achieved by the best-so-far solution against the number of fitness function evaluations normalized by the problem dimensionality. Each ECDF curve allows one to aggregate performance over different problems, provided that objective function values of the optimums are identical for those problems.

The third type of analysis illustrates the importance of BCHMs in assessing the efficiency of CMA-ES in performing global optimization with bound constraints. In the analysis, the ECDF curves obtained by running CMA-ES, coupled with various BCHMs, are compared using functions from the CEC'2017 and the BBOB benchmarks. For all CEC'2017 functions and most BBOB functions, it is known that the global optimum position is relatively distant from the bounds of the admissible area. Despite that, there are differences in the convergence speed between various BCHMs.

The results of the aforementioned analyses divide the BCHMs into 3 groups: good, weak and bad. On that base, the most promising methods are selected and compared for each CEC 2017 function using a statistical test.

In the experiments, the CMA-ES implementation from [32] was used, which refers to [35] in its documentation. The implementation utilized *multiplicative penalty* as the only BCHM. Other BCHMs were implemented by the author of this study. All CMA-ES parameters were set typically by the implementation. The budget was set following the CEC 2017 rules. For all functions, their results were shifted to produce 0 in the global optimum. The *reflection*, *projection* and *wrapping* methods were combined with CMA-ES according to the Darwinian and Lamarckian approach. In the *resampling* method, after 100 consecutive unsuccessful repair trials, the Lamarckian *projection* is used. The *projection* method was used to calculate quadratic penalties. The optimization was stopped when the fitness error level dropped below 10^{-8}.

4.1. Influence of bound constraint handling on convergence speed

This section presents the results of experiments which were performed on three types of unimodal, 10-dimensional functions: Sphere (27), Ellipsoid (28) and TwoAxes (29).

$$q_{sph}(\mathbf{x}) = \sum_{i=1}^{n} (x_i - b)^2 \tag{27}$$

$$q_{ell}(\mathbf{x}) = \sum_{i=1}^{n} 10^{6\frac{i-1}{n-i}} (x_i - b)^2 \tag{28}$$

$$q_{twoax}(\mathbf{x}) = \sum_{i=1}^{n} \begin{cases} 10^6 (x_i - b)^2 \text{ when } i \text{ is even} \\ (x_i - b)^2 \text{ otherwise} \end{cases} \tag{29}$$

These functions were previously used in [13] to examine *Weighted penalty with trimmed median*.

The feasible area was defined as a box $[-1, 1]^{10}$, and b was changed in the range $[0.2, 1]$. For each b and each BCHM, 51 independent runs were performed. The expected runtime (ERT) [36] was used as a performance measure. This is a well-established measure as it is used in the COCO framework [37], which in turn is used by the GECCO Workshop on Real-Parameter (BBOB). The ERT is calculated as the sum of the used evaluations divided by the number of runs in which the optimum was found with the aforementioned error level. ERTs were also calculated for the unconstrained case and the resulting values are used as a reference. For each BCHM, a plot of its ERT divided by the reference ERT vs. the value of b is depicted in Fig. 2. The Y-axis was limited to 3 for better readability of the interesting area. As a result, some weak methods are not visible in the plot.

The results of the experiments from Fig. 2 indicate that most BCHMs fail when the optimum approaches the boundary, i.e. their results quickly deteriorate with the increase of b. From the good methods, *transformation* achieved the best speedup. It appears that the quadratic transformation introduced around the boundary shortens steps made by the search-facilitating location of the boundary. Second place goes to the group of methods based on penalties, *resampling* and Darwinian *reflection*. Among them, the simple quadratic penalty performs best for the quadratic function but it is about two times worse than *transformation* for *Ellipsoid* and *TwoAxes*. The *weighted penalty with trimmed median* is the best for *TwoAxes*. For other functions, this method performs similarly to *weighted penalty with decreasing weights* and *weighted penalty*. For the *TwoAxes* function, the *Weigh. pen. Matlab impl.* method which is used in the Matlab implementation of CMA-ES is worse than *weighted penalty*. What is surprising here is that one of the simplest approaches, i.e., Darwinian *reflection*, is as good as more sophisticated methods that utilize knowledge gathered by CMA-ES, i.e., C and σ. The group of good methods also includes *substitution penalty*, *death penalty* and *Darwinian wrapping*, but their performance deteriorates for b very close to 1, i.e., when the optimum is located on the bounds.

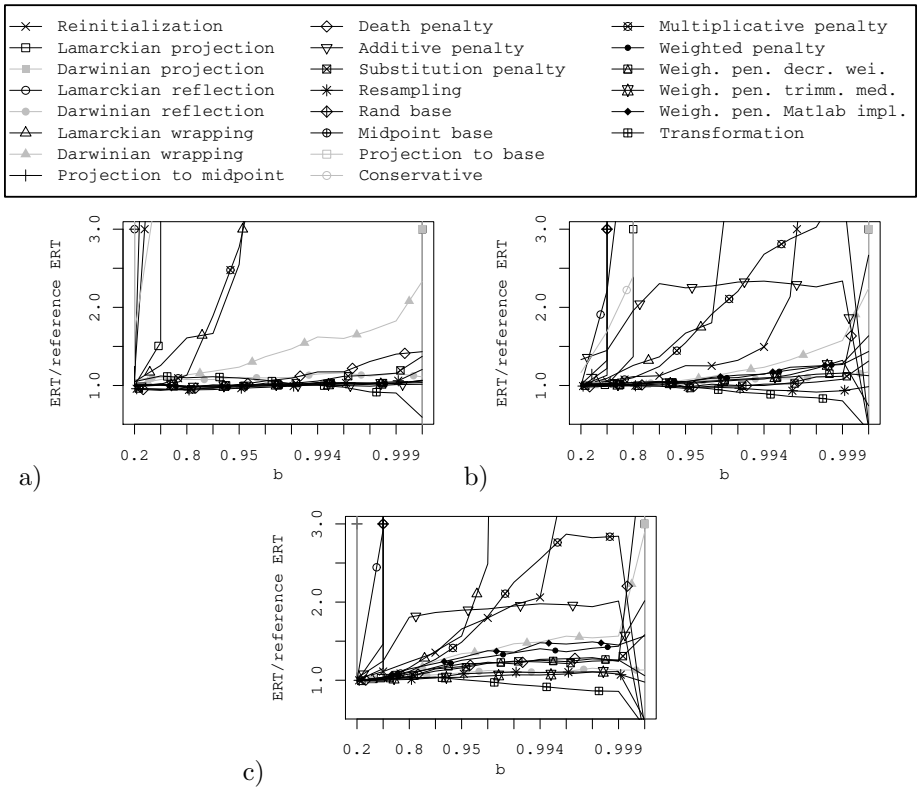

Figure 2: Ratio of ERT values vs. value of b for different BCHMs for quadratic (a), ellipsoid (b) and TwoAxes function (c)

The *transformation* method is the best in terms of speedup, which does not necessarily means that it is the best BCHM. When the method is faster than the unbounded search that means that it introduces a bias to the algorithm. The bias is good for tasks considered in this section but it may be bad for other tasks (cf. Tab. 4). Assuming that the convergence rate should not be affected by BCHM, the best BCHMs are Darwinian *reflection* and *resampling*, as their results were very close to unbounded search results for all examined functions.

It is worth noting that all Darwinian approaches beat their Lamarckian counterparts. Also, both variants of the *projection* methods give good results only in two cases: 1) when the optimum is exactly on the bound; 2) when the optimum is near to the center of the feasible area.

4.2. Influence of bound constraint handling on global optimization performance

This part of the experiments was performed using the CEC 2017 benchmark suite [9], which defines 30 functions for 10, 30, 50, and 100 dimensions. Selected experiments were also repeated using real-parameter noise-less BBOB functions to confirm that the results are not benchmark specific. To visualize the impact of BCHMs on the performance of CMA-ES, Empirical Cumulative Distribution Function (ECDF) curves are used.

4.2.1. Optimum on bounds

To force CMA-ES to extensively use BCHMs, the upper bounds were set to the coordinates of the optimum. The lower bounds were set to -100, as was originally specified by CEC 2017 rules [9]. The analyzed set of functions included the first 20 CEC 2017 problems. From that set, only functions 1-4, 13, and 19 were solved with CMA-ES. The problem is considered to be solved here when the median of results is in the global optimum (with error margin 10^{-8}) for at least one BCHM.

The results of the experiments for solved functions are depicted in Fig. 3 in the form of ECDF curves. To facilitate numerical comparison, the area under ECDF curves (AUC) was calculated and reported in Table 1. The rows of the table were sorted according to mean AUC values, so the best performing method is at the top of the table. The results of the unsolved functions are not presented because curves were near the x-axis and were nearly indistinguishable.

According to Fig. 3 and Tab. 1, BCHMs can be divided into 3 groups: the leaders, good methods and weak methods. The best of the leaders is *resampling*. Its superiority increases with the increase of the problem dimensionality. The second is Darwinian *reflection*, whose performance seems to be dimension-invariant. Then follows a group of methods based on the weighted penalty. The Darwinian *wrapping* closes the group of leaders. The group of good methods consists of *substitution penalty*, *multiplicative penalty* and Darwinian *projection*. The *weighted penalty with trimmed median* is somewhere between good and weak methods. Its performance strongly deteriorated with dimensionality. The group of weak methods includes Lamarckian *projection*, which is frequently perceived as a good BCHM for cases when the optimum is on the bounds. *Transformation*, which is the default method in some advanced CMA-ES implementations,

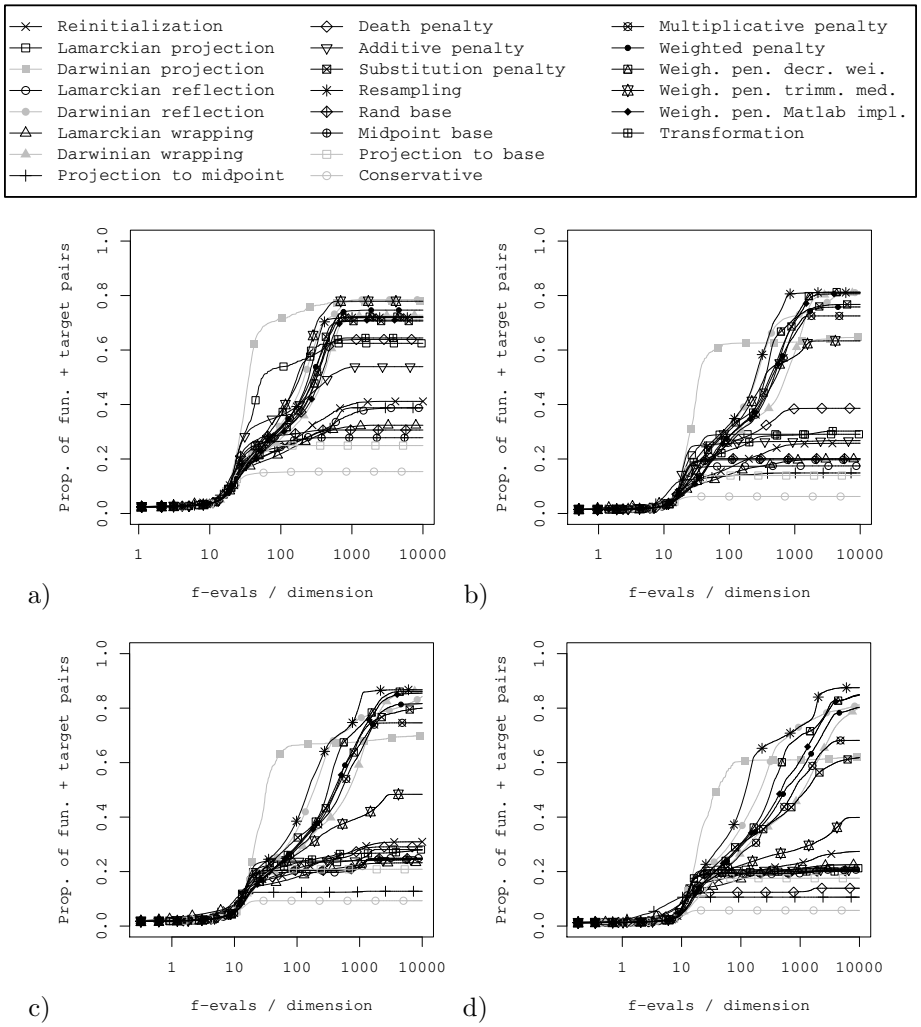

Figure 3: ECDF curves for functions 1-4, 13, 19 from CEC 2017 when the optimum is on bounds for 10 (a), 30 (b), 50 (c) and 100 (d) dimensions for different BCHMs

Table 1: Area under ECDF curves for functions 1-4, 13, 19 from CEC 2017 when the optimum is on bounds

method	10D	30D	50D	100D
Resampling	0.71	**0.79**	**0.84**	**0.84**
Darwinian reflection	0.77	0.77	0.79	0.77
Weighted penalty with decreasing weights	0.71	0.78	0.82	0.79
Weigh. pen. Matlab impl.	0.69	0.77	0.81	0.77
Weighted penalty	0.73	0.73	0.77	0.73
Darwinian wrapping	0.71	0.76	0.80	0.70
Substitution penalty	0.69	0.74	0.76	0.58
Multiplicative penalty	0.71	0.70	0.71	0.64
Darwinian projection	**0.78**	0.64	0.69	0.62
Weighted penalty with trimmed median	0.77	0.62	0.46	0.36
Death penalty	0.63	0.38	0.29	0.14
Transformation	0.64	0.30	0.27	0.20
Lamarckian projection	0.62	0.29	0.27	0.21
Additive penalty	0.53	0.27	0.24	0.21
Reinitialization	0.40	0.25	0.30	0.26
Lamarckian reflection	0.38	0.17	0.24	0.20
Lamarckian wrapping	0.32	0.20	0.23	0.22
Rand base	0.31	0.20	0.24	0.20
Midpoint base	0.28	0.20	0.24	0.20
Projection to base	0.25	0.14	0.21	0.18
Projection to midpoint	0.38	0.15	0.13	0.11
Conservative	0.15	0.06	0.09	0.06

and *additive penalty*, which is used in some other CMA-ES implementations, are also grouped as weak methods. This group also includes *reinitialization*, which sometimes is considered as a good method because it does not bias the search. Somewhat surprising here are the results of *projection*. It is easy to assume that projection is the best when the optimum is on the bounds. In reality, it is highly unlikely that all coordinates of the mutant will be out of bounds, so the mutant will be shifted to the optimum. When only a few coordinates are out of bounds, some of the others may still stick in a local optimum. Besides that, for Lamarckian *projection*, setting the mutant's coordinates to the bounds disrupts the CMA-ES auto-adaptation mechanism, which is detected by the CMA-ES internal stopping criterion. As a result, the search is interrupted much too early, e.g. in the case of function 4 in 30D, the local optimum was found at about 160 iterations, and at about 280 iterations the search was interrupted.

When analyzing results from this and the previous section together (Fig. 2, Fig. 3 and Tab. 1), it can be observed that when the optimum is on the bounds, the best BCHM is *resampling*, closely followed by Darwinian *reflection* and *weighted penalty with decreasing weights*.

4.2.2. Optimum far away from bounds

In many practical applications and in the CEC 2017 benchmark, the optimum does not lie on the bounds. What is more, in the case of CEC 2017, it is guaranteed that the optimum is inside the box $[-80, 80]^n$, which is quite far from the CEC 2017 bounds of $[-100, 100]^n$. To investigate the influence of BCHMs on the results of CMA-ES when the optimum is far from the bounds, all functions from the CEC 2017 benchmark were examined. Even though experiments in this section use the same CEC functions as in Section 4.2.1, resulting optimization problems are different because of different bounds. Therefore, the set of solved functions identified in this section differs from the set identified in Section 4.2.1. Additionally, to check if the results are not specific for CEC 2017 benchmark, the experiments were also performed on 24 real-parameter noise-less 10-dimensional functions from BBOB (Black-Box Optimization Benchmarking) [8] with bounds set to $[-5, 5]^n$. The implementations of these functions were taken from COCO 2.3.1 [37]. The experiments on BBOB functions were performed in the same setup as experiments on CEC functions, i.e. the experiments were performed according to CEC 2017 rules, they were not performed by COCO platform. COCO was only used to get the first instance of each function.

Aggregated results. The results are presented using ECDFs as in section 4.2.1. Fig. 4 shows ECDF curves aggregated for solved functions (CEC functions 1-4, BBOB functions 1, 2, 5, 6, 8-14) and Fig. 5 shows ECDFs aggregated for unsolved functions (CEC functions 5-30, BBOB functions 3, 4, 7, 15-24). To facilitate numerical comparison, the AUC was computed and reported in Tab. 2 and Tab. 3, respectively. The rows in each table were sorted according to mean AUC values, so the best method is at the top of the table.

Figure 4 together with Tab. 2 reveal that with the increase of the problem dimensionality, the differences between the performance of BCHMs also increase.

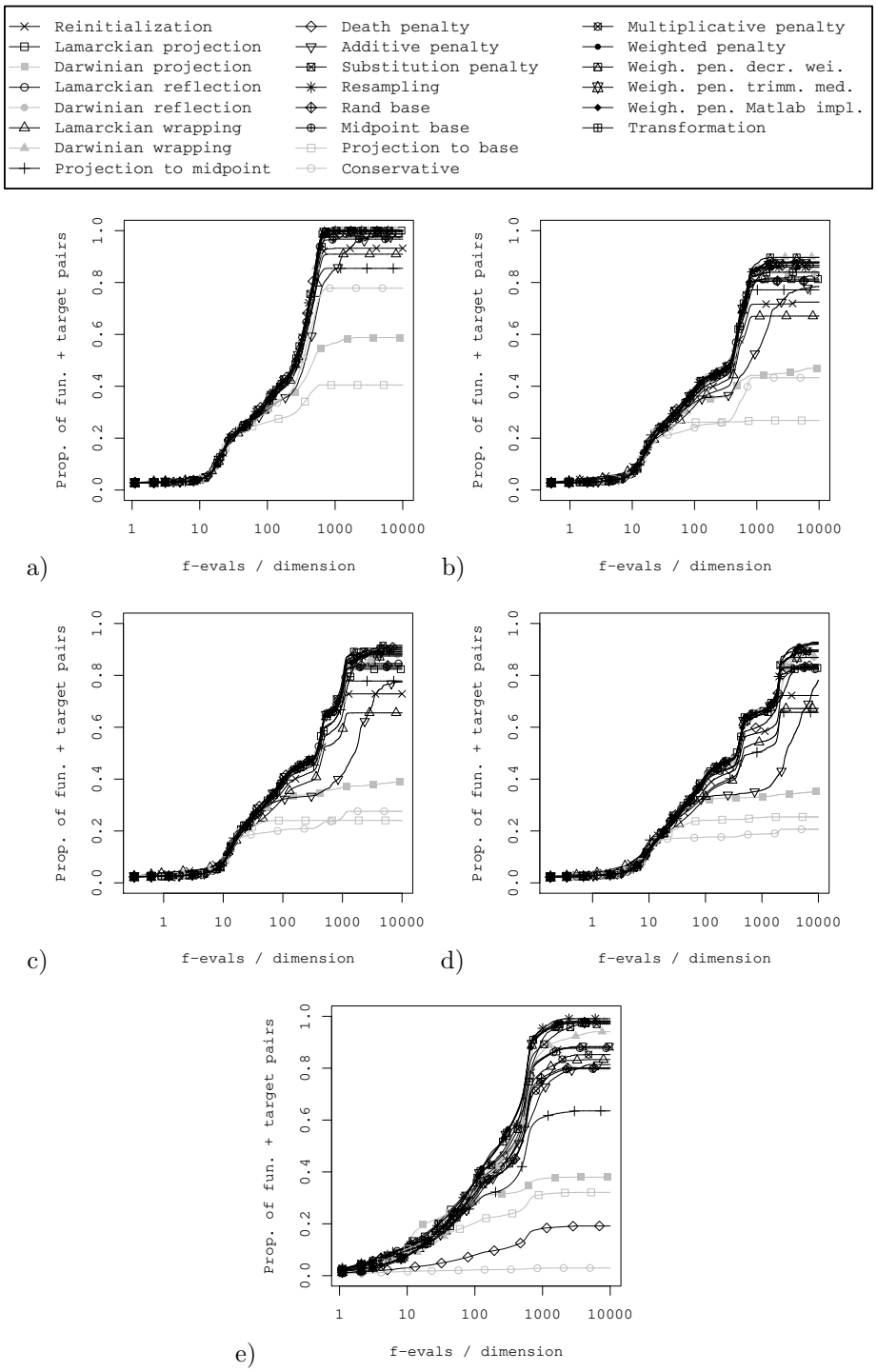

Figure 4: ECDF curves for functions 1-4 from CEC 2017 for 10 (a), 30 (b), 50 (c), 100 (d) dimensions and ECDF curves for functions 1, 2, 5, 6, 8-14 from BBOB for 10 dimensions (e) for different BCHMs

Table 2: Area under ECDF curves for functions 1-4 from CEC 2017 and for functions 1, 2, 5, 6, 8-14 from BBOB

	CEC 2017				BBOB
method	10D	30D	50D	100D	10D
Weigh. pen. with decreasing weights	**0.97**	0.84	0.86	0.84	0.94
Weigh. pen. Matlab impl.	**0.97**	0.85	0.85	0.85	0.95
Weighted penalty	**0.97**	0.84	0.85	**0.86**	0.95
Resampling	**0.97**	0.85	0.86	0.83	**0.96**
Substitution penalty	**0.97**	0.85	0.84	0.84	0.93
Darwinian wrapping	**0.97**	0.85	0.83	0.83	0.90
Weigh. pen. with trimmed median	**0.97**	0.83	0.83	0.82	0.94
Darwinian reflection	**0.97**	0.81	0.82	0.81	0.95
Multiplicative penalty	**0.97**	0.84	**0.87**	0.84	0.81
Transformation	**0.97**	0.81	0.84	0.78	0.94
Lamarckian reflection	0.96	0.80	0.82	0.78	0.85
Lamarckian projection	**0.97**	0.79	0.80	0.79	0.85
Rand base	0.95	0.78	0.80	0.79	0.77
Midpoint base	0.94	0.79	0.81	0.79	0.77
Reinitialization	0.91	0.70	0.71	0.69	0.85
Additive penalty	0.93	0.70	0.67	0.60	0.77
Death penalty	0.96	**0.86**	0.83	0.78	0.18
Lamarckian wrapping	0.89	0.67	0.65	0.64	0.80
Projection to midpoint	0.83	0.74	0.75	0.62	0.61
Darwinian projection	0.57	0.44	0.37	0.34	0.37
Conservative	0.76	0.40	0.25	0.20	0.03
Projection to base	0.40	0.27	0.24	0.25	0.31

It can be also observed that most BCHMs perform equally well, but there are some that perform poorly. Like in Section 4.2.1, *conservative* is in the group of weak methods. Its application results in a population that contains many copies of an individual. That situation is perceived by CMA-ES as a flat land in the fitness landscape. The algorithm tries to escape from the flat land by increasing σ, which raises the number of infeasible solutions. The rise of σ is also a cause for the failure of *projection to base*. Other methods that should also be avoided include Darwinian *projection* and Lamarckian *wrapping*.

The *projection to midpoint* and the *additive penalty* are also below the average. The simple *additive penalty* is worse than weighted penalties, and is even much worse than *multiplicative penalty*.

The results of the experiments on 10-dimensional BBOB functions generally confirm the results of the experiments on CEC 2017 problems. The same functions belong to the group of good methods. For BBOB the best is *resampling* followed by Darwinian *reflection*, weighted penalties and *transformation*. Some differences in the performance of BCHMs for CEC and BBOB can be observed in the set of medium and weak BCHMs, i.e. *death penalty* and *conservative* gave worse results for BBOB than for CEC. This deterioration stems from the fact that each of these BCHMs gives individuals of equal fitness. It is not the problem only when BCHMs are called rarely, but in the case of BBOB BCHMs are called more frequently than in the case of CEC.

In Fig. 5, the aggregated performance of BCHMs for unsolved problems is depicted and the AUC is reported in Tab. 3. The set of weak BCHMs are the same as for solved functions, but in this scenario, it is easier to notice that Lamarckian *wrapping* and *reinitialization* improved performance on CEC 2017 problems at the beginning of the search. Presumably these BCHMs increase diversity in the population. As it was for the solved problems, the results on BBOB problems are in good agreement with the results on CEC 2017. Once again the best is *resampling*.

It is worth noting that the *conservative* method, which is the worst in this study, was better than *reinitialization* in the case of DE[19]. Here, Darwinian *reflection* is one of the best, in DE it was one of the worst. Therefore, conclusions drawn for one optimization method cannot be easily transferred to another. One exception from this rule is *resampling*, which is good for both algorithms and should be good for every reasonable method.

Selected detailed results. In the aggregated results, some BCHMs that are important for CMA-ES are hardly distinguishable. Therefore, this section examines methods that are used in the official, off-the-shelf CMA-ES implementations as well as methods that achieve good results in experiments reported in tables 1-3. The results of these methods were compared using the paired Wilcoxon test at a confidence level of 0.95 with Benjamini-Hochberg correction with expected proportion of false discoveries set to 5%. To save space, the results for all functions from CEC 2017 for $n = 50$ are provided. The results for $n = 30$ and $n = 100$ look similar, for $n = 10$ the differences between BCHMs are smaller. The results are presented in Tab. 4. Since *weighted penalty* is an ancestor of

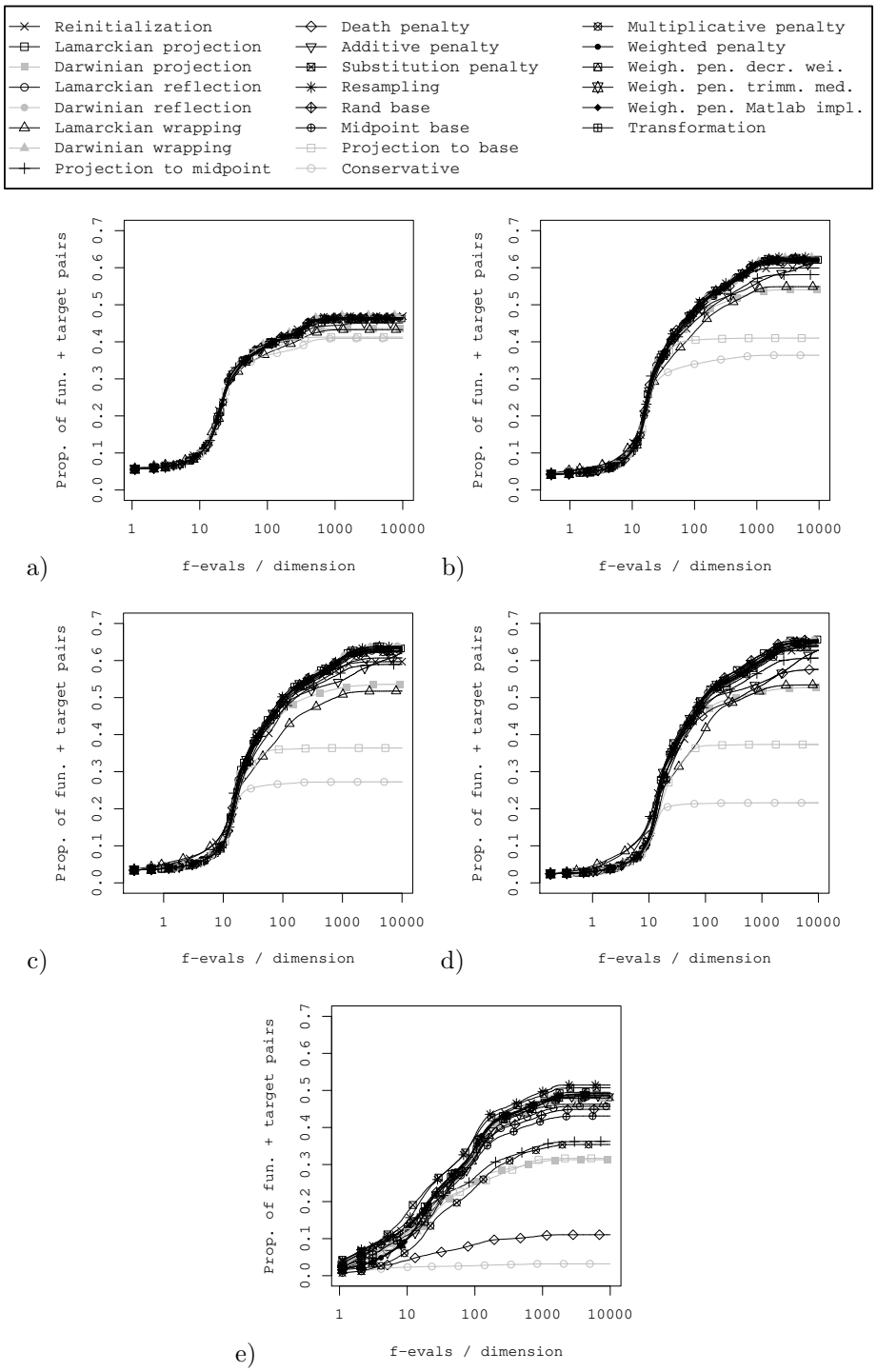

Figure 5: ECDF curves for functions 5-30 from CEC 2017 for 10 (a), 30 (b), 50 (c), 100 (d) dimensions and ECDF curves for functions 3, 4, 7, 15-24 from BBOB for 10 dimensions (e) for different BCHMs

Table 3: Area under ECDF curves for functions 5-30 from CEC 2017 and for functions 3, 4, 7, 15-24 from BBOB

	CEC 2017				BBOB
method	10D	30D	50D	100D	10D
Resampling	**0.46**	**0.61**	0.61	0.63	**0.51**
Darwinian reflection	**0.46**	**0.61**	**0.62**	**0.64**	0.48
Lamarckian projection	0.45	**0.61**	0.61	**0.64**	0.47
Weigh. pen. with trimmed median	0.45	0.60	0.61	0.63	0.48
Darwinian wrapping	**0.46**	**0.61**	0.61	0.63	0.47
Transformation	0.45	**0.61**	0.61	**0.64**	0.47
Weigh. pen. with decreasing weights	0.45	0.60	0.61	0.63	0.49
Substitution penalty	0.45	0.60	0.60	0.62	0.50
Weigh. pen. Matlab impl.	0.45	0.60	0.61	0.63	0.48
Weighted penalty	0.45	0.60	0.61	0.63	0.48
Lamarckian reflection	**0.46**	0.60	0.61	**0.64**	0.45
Rand base	**0.46**	0.60	0.61	**0.64**	0.44
Midpoint base	0.45	0.60	0.61	0.63	0.43
Reinitialization	**0.46**	0.59	0.58	0.62	0.48
Additive penalty	0.44	0.58	0.58	0.59	0.48
Multiplicative penalty	0.45	0.60	0.60	0.62	0.35
Projection to midpoint	**0.46**	0.56	0.57	0.59	0.36
Lamarckian wrapping	0.42	0.53	0.51	0.53	0.46
Death penalty	0.45	0.59	0.59	0.56	0.11
Darwinian projection	0.43	0.53	0.52	0.52	0.31
Projection to base	0.41	0.40	0.36	0.37	0.31
Conservative	0.40	0.35	0.27	0.22	0.03

the group of methods, it was used as a reference method. In Tab. 4, a dot (.) means that there was no significant difference, a plus (+) means that the listed approach is better than the reference method, and a minus (−) means that it is worse.

It can be observed that from the group of descendants of weighted penalty, the best is *weighted penalty with trimmed median*. The second is *weighted penalty with decreasing weights*. The last from the group, *weighted penalty Matlab impl.*, is neither better nor worse than the parent. The most interesting observation is that Darwinian *reflection*, which is never used in practical implementations of CMA-ES, is the best method in the table. The second best is *resampling*, and *transformation* is the third. The worst is *additive penalty*, which is a traditional method and is still used in some implementations. Other simple methods based on penalties, i.e. *substitution penalty* and *multiplicative penalty*, also cannot be recommended as a replacement for the standard BCHM. For *reinitialization*, there are 20 statistically important differences. In most cases, this BCHM worsens the performance of CMA-ES, but for some functions, it helped the algorithm to escape from local optima.

For some BCHMs, a more clear picture emerges when CEC 2017 functions are analyzed in groups. For the simple multimodal group (functions 4-10), the best is *reinitialization*, but it is the worst for unimodal (1-3) and hybrid functions (11-20). Most BCHMs beat the reference method on composition functions (21-30). For that group, the best is *transformation*, followed by Darwinian *reflection* and *resampling*.

Dependence of the number of infeasible individuals on BCHMs. In the case of CEC 2017, bounds are quite far from the optimum. Therefore, it is interesting how frequently an infeasible individual is generated. The BCHMs that generate fewer infeasible samples should be preferred, because every BCHM call takes some time and it can change properties of the optimization algorithm, e.g. the bias. The results of the experiments for $n = 50$ are depicted in Fig. 6. Each infeasibility is counted only once, i.e., if a BCHM produces an infeasible solution it is not counted.

It can be observed that even though bounds in the CEC 2017 benchmark are quite far from the global optimum, the BCHMs are used frequently. For the solved functions, the *multiplicative penalty* generated the fewest infeasible individuals among successful BCHMs. About 10% more infeasible individuals were generated with *weighed penalty* and another 6% more were generated by *resampling*, *weighted penalty with decreasing weights* and *weighted penalty Matlab implementation*.

For unsolved functions, *rand base*, *midpoint base* and Lamarckian *projection* achieved large AUCs and at the same time, they are the best in terms of generating the fewest infeasible individuals. Among weighted penalties, the best is the version with decreasing weights and the worst is the original proposition. For unsolved problems, the *multiplicative penalty* generates even more infeasible individuals than *resampling*.

Weak BCHMs, such as *conservative* and *projection to base*, seldom produce

Table 4: Results of paired Wilcoxon test with a confidence level of 0.95 with Benjamini-Hochberg correction, taking *weighted penalty* as a reference method, measured on 50-dimensional functions from CEC 2017. A dot (.) means that there was no significant difference, a plus (+) means that listed approach is better than the reference method, and a minus (−) means that it is worse

fun. #	Weigh. pen. decr. wei.	Weigh. pen. trimm. med.	Weigh. pen. Matl. impl.	Darw. refl.	Resampling	Subst. pen.	Multip. pen.	Addit. pen.	Reinit.	Transf.
1	.	.	.	.	.	.	.	.	−	.
2	.	.	.	.	.	.	.	−	−	.
3	.	.	.	.	.	.	.	.	−	.
4	.	.	.	.	.	.	.	.	−	.
5	.	.	.	.	+	.	.	.	+	.
6	.	.	.	.	.	.	.	.	+	.
7	.	.	.	.	.	.	.	.	+	.
8	.	.	.	.	.	.	.	−	+	.
9	.	.	.	.	.	.	.	.	+	.
10	.	.	.	.	.	.	.	.	.	.
11	.	.	.	.	.	.	.	.	.	−
12	.	.	.	.	.	.	.	−	−	.
13	.	.	.	.	.	.	.	.	−	.
14	.	.	.	.	.	.	.	.	−	.
15	.	.	.	.	.	.	.	−	−	.
16	.	.	.	+	.	.	.	.	.	.
17	.	.	.	.	.	.	.	.	.	.
18	.	.	.	.	.	.	.	−	−	.
19	.	.	.	+	.	.	.	.	−	.
20	.	.	.	.	.	.	.	.	+	.
21	.	.	.	.	.	.	.	.	.	.
22	.	.	.	.	.	.	.	.	.	.
23	.	.	.	.	.	.	.	.	+	.
24	.	+	.	+	+	−	−	.	.	+
25	.	.	.	.	.	.	.	.	.	+
26	.	.	.	+	+	.	−	.	.	+
27	.	+	.	+	+	−	−	+	.	+
28	.	.	.	.	.	.	.	.	−	.
29	.	.	.	+	+	.	.	.	+	+
30	+	+	.	+	+	.	+	.	−	+
+/−	1/0	3/0	0/0	7/0	6/0	0/2	1/3	1/5	8/12	6/1

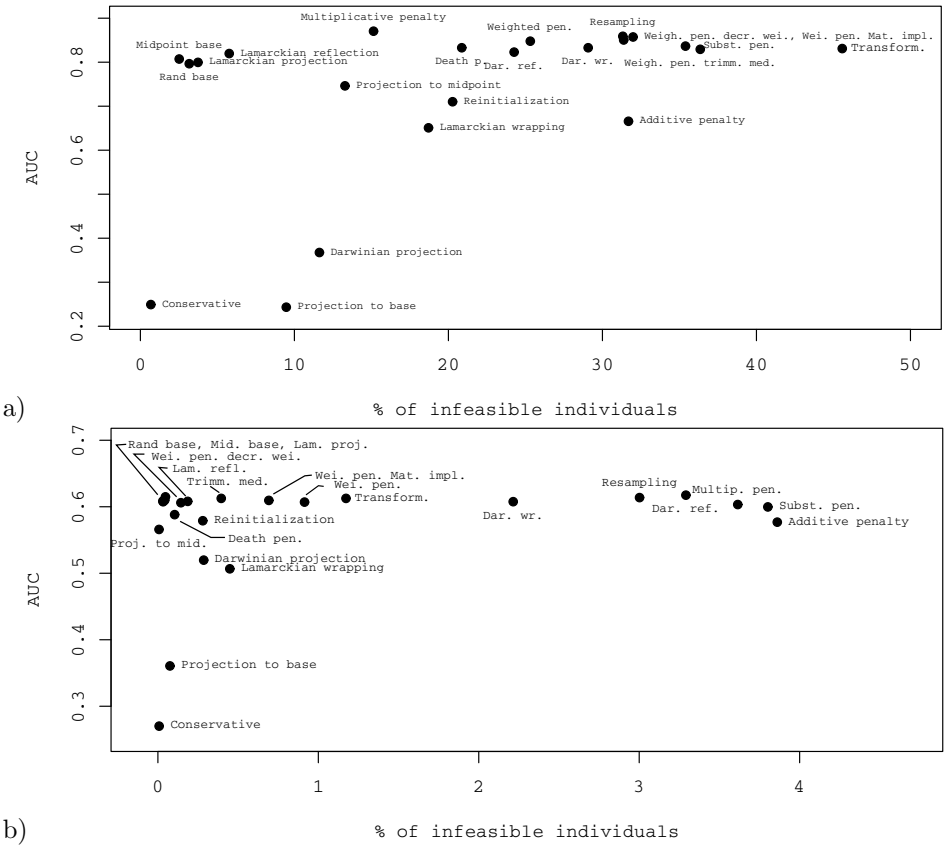

Figure 6: The average percentage of generated infeasible individuals for CEC 2017 problems in 50D for functions 1-4 (a) and for functions 5-30 (b)

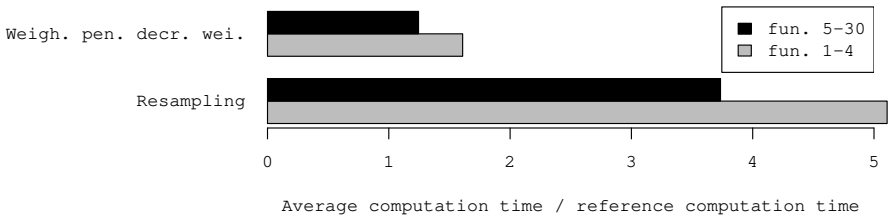

Figure 7: Relative time of computation for selected BCHMs for functions from CEC 2017 in 30D.

infeasible solutions. The reason for that is a relatively quick interruption of the search when CMA-ES approaches the boundary, which is caused by the stop criterion that detects unreasonable values of CMA-ES parameters, e.g. σ.

Influence of BCHMs on computation time. Most BCHMs are similar in terms of computational complexity and time of computation, but two interesting methods: *resampling* and *weighted penalty with decreasing weights* seem to be complicated and time-consuming. To estimate the computational overhead of the BCHM, the sum of times of all independent runs of the search with the examined BCHM was divided by the number of utilized evaluations of the objective function. The normalized values were referred to the normalized result of the good and fast method, i.e., Darwinian *reflection*. That proportion, aggregated separately for solved and unsolved 30-dimensional functions, is depicted in Fig. 7.

It can be observed that using *resampling* for functions 1-4 makes optimization about 5 times slower than with the use of the reference method. The *weighted penalty with decreasing weights* makes optimization only 1.6 times slower. Even though BCHMs are less used for functions 5-30, *resampling* makes optimization about 3.7 times slower.

4.3. Relation to recently published results

From the practitioner's point of view, it is interesting to check how results of CMA-ES with the best BCHM relate to the recently published results on CEC 2017 benchmark for single objective real-parameter numerical optimization. To investigate that, the newest version (2.7.1) of Hansen's *cma* package was used [24]. The package was configured to use the IPOP-CMA-ES [38] variant with up to 9 restarts and $\sigma = 0.3(u - l)$. It also allows one to use BIPOP-CMA-ES [39] but the results of BIPOP were generally worse than those of IPOP on the considered functions. The experiments were performed in 10, 30 and 50 dimensions, but only the results in 30 dimensions are shown in this section. The complete results of IPOP-CMA-ES are available at [40] in raw tables. From the set of the best BCHMs, Darwinian *reflection* was used because it is the fastest method. The results of using it in IPOP-CMA-ES were compared to the results

Table 5: The mean and standard deviation of IPOP-CMA-ES, IPOP-CMA-ES with Darwinian *reflection* and recently published variants of L-SHADE on 30-dimensional functions from CEC 2017. The best values in each row are typeset in bold. Additionally, the influence of changing default IPOP's BCHM into Darwinian *reflection* was verified using the paired Wilcoxon test with a confidence level of 0.95. A tilde (∼) means that there was no significant difference, a plus (+) means that Darwinian *reflection* is better than default method, and a minus (−) means that it is worse

F.	IPOP-CMA-ES	IPOP-CMA-ES D. refl.	SALSHADE-cnEPSin	SALSHADE
1	0.00e+0 ± 0e+0	0.00e+0 ± 0e+0 (∼)	0.00e+0 ± 0e+0	0.00e+0 ± 0e+0
3	0.00e+0 ± 0e+0	0.00e+0 ± 0e+0 (∼)	0.00e+0 ± 0e+0	0.00e+0 ± 0e+0
4	5.63e+1 ± 1e+1	5.29e+1 ± 1e+1 (∼)	**4.90e+1** ± 3e+0	5.57e+1 ± 5e+0
5	**4.92e+0** ± 2e+0	5.56e+0 ± 2e+0 (∼)	1.24e+1 ± 2e+0	1.93e+1 ± 4e+0
6	1.40e−7 ± 9e−7	**0.00e+0** ± 0e+0 (∼)	**0.00e+0** ± 0e+0	**0.00e+0** ± 0e+0
7	**5.31e+0** ± 4e+0	7.29e+0 ± 8e+0 (∼)	4.32e+1 ± 2e+0	5.35e+1 ± 7e+0
8	5.07e+0 ± 2e+0	**4.86e+0** ± 1e+0 (∼)	1.36e+1 ± 2e+0	1.74e+1 ± 4e+0
9	0.00e+0 ± 0e+0	0.00e+0 ± 0e+0 (∼)	0.00e+0 ± 0e+0	0.00e+0 ± 0e+0
10	3.76e+2 ± 2e+2	**2.05e+2** ± 2e+2 (+)	1.47e+3 ± 2e+2	3.15e+3 ± 4e+2
11	1.06e+1 ± 1e+1	8.58e+0 ± 1e+1 (∼)	**3.93e+0** ± 2e+1	1.06e+1 ± 1e+1
12	8.60e+2 ± 3e+2	9.16e+2 ± 2e+2 (∼)	3.43e+2 ± 2e+2	**2.17e+2** ± 1e+2
13	1.96e+1 ± 8e+0	1.72e+1 ± 4e+0 (∼)	**1.70e+1** ± 5e+0	2.80e+1 ± 8e+0
14	4.04e+1 ± 3e+1	4.56e+1 ± 4e+1 (∼)	**2.20e+1** ± 4e+0	2.60e+1 ± 1e+0
15	1.20e+2 ± 1e+2	2.66e+2 ± 1e+2 (−)	**3.65e+0** ± 2e+0	8.97e+0 ± 1e+0
16	1.79e+2 ± 1e+2	2.01e+2 ± 1e+2 (∼)	**1.38e+1** ± 4e+1	1.29e+2 ± 7e+1
17	6.07e+1 ± 3e+1	6.31e+1 ± 2e+1 (∼)	**2.83e+1** ± 6e+0	6.22e+1 ± 9e+0
18	9.22e+1 ± 6e+1	1.49e+2 ± 8e+1 (−)	**2.06e+1** ± 9e−1	2.39e+1 ± 1e+0
19	1.61e+1 ± 1e+1	9.17e+1 ± 5e+1 (−)	**5.91e+0** ± 2e+0	1.38e+1 ± 1e+0
20	6.83e+1 ± 4e+1	5.95e+1 ± 4e+1 (+)	**3.08e+1** ± 6e+0	8.28e+1 ± 1e+1
21	2.08e+2 ± 2e+0	**2.07e+2** ± 2e+0 (∼)	2.13e+2 ± 2e+0	2.19e+2 ± 4e+0
22	1.94e+2 ± 2e+2	1.30e+2 ± 9e+1 (+)	**1.00e+2** ± 0e+0	**1.00e+2** ± 0e+0
23	3.62e+2 ± 4e+0	3.60e+2 ± 5e+0 (+)	**3.54e+2** ± 4e+0	3.65e+2 ± 6e+0
24	4.32e+2 ± 3e+0	**4.28e+2** ± 2e+0 (+)	4.29e+2 ± 3e+0	4.32e+2 ± 4e+0
25	3.87e+2 ± 6e−1	3.87e+2 ± 6e−3 (∼)	**3.86e+2** ± 7e−3	**3.86e+2** ± 1e−1
26	1.12e+3 ± 1e+2	**9.35e+2** ± 1e+2 (+)	9.51e+2 ± 5e+1	1.00e+3 ± 7e+1
27	5.11e+2 ± 6e+0	**4.87e+2** ± 1e+1 (+)	5.03e+2 ± 4e+0	5.01e+2 ± 5e+0
28	3.06e+2 ± 2e+1	3.09e+2 ± 2e+1 (∼)	**3.00e+2** ± 4e+1	3.28e+2 ± 4e+1
29	5.81e+2 ± 7e+1	5.12e+2 ± 9e+1 (+)	**4.38e+2** ± 1e+1	5.16e+2 ± 1e+1
30	2.22e+3 ± 1e+2	**9.91e+2** ± 7e+2 (+)	1.97e+3 ± 4e+1	1.97e+3 ± 1e+1

of recently published variants of L-SHADE [41] because its derivatives are among the winners of the CEC 2017 and CEC 2018 competitions on single objective bound-constrained optimization. It seems that step-by-step improvements made in this family of algorithms created a success story and further improvements still constitute a good research direction [42]. The set of methods included in the comparison include SALSHADE [43] and SALSHADE-cnEPSin [44]. The results of these methods were taken from the corresponding referenced articles. The experiments also investigate the influence of Darwinian *reflection* on IPOP-CMA-ES performance. The results are provided in Table 5.

According to the results shown in Table 5, there are 12 statistically significant differences between the results of IPOP with Darwinian *reflection* and IPOP's default configuration, which uses *transformation*. In 9 cases Darwinian *reflection* improved the results. Therefore, BCHMs are also important in IPOP version of CMA-ES and Darwinian *reflection* improves the average performance of IPOP on CEC 2017.

Even though IPOP with Darwinian *reflection* was not tuned to solve CEC 2017 problems, it yielded averages that are better than L-SHADE derivatives for 7 functions. It was the best algorithm on shifted and rotated versions of non-

continuous Rastrigin's and Schwefel's functions (number 8 and 10). It was also good at optimizing composition functions (21-30). When comparing modified IPOP with SALSHADE it seems that IPOP is slightly better (15:8 wins in the comparison of the means), but clearly SALSHADE-cnEPSin is the best algorithm in the table, as it has the best mean in 16 cases and modified IPOP has the best mean in 8 cases. SALSHADE-cnEPSin is especially good at hybrid functions (11-20).

The results can be concluded as follows. The choice of BCHM is important in the modern competition scenario. Having proper BCHM improves the results of IPOP-CMA-ES on the average, which allows it to beat the results of one of recently proposed algorithms, but does not allow it to win the competition with the most recent variant of L-SHADE. It is highly probable that future modifications of IPOP-CMA-ES (with proper BCHM) can make it quite a competitive method.

5. Conclusions

In this paper, the influence of 22 BCHMs on the performance of the CMA-ES algorithm was investigated. The empirical tests included three scenarios: 1) three unimodal objective functions with changing distances from the optimum to the boundary; 2) functions from the CEC 2017 benchmark with upper bounds set to the optimum; 3) functions from the CEC 2017 benchmark with bounds set to $[-100, 100]^n$ and functions from the BBOB benchmark with bounds set to $[-5, 5]^n$.

The results of the experiments make it possible to draw several conclusions. The most important of them is that the choice of BCHM does make a difference to CMA-ES. Even though optima are quite far from the bounds in the regular CEC 2017 benchmark, BCHMs are used frequently. For the benchmark, the best results were achieved by BCHMs that are not used in off-the-shelf CMA-ES implementations, i.e. by Darwinian *reflection* and *resampling*. Using Darwinian *reflection* in IPOP version of CMA-ES gives also better average results than IPOP's default configuration, and makes it more competitive with the most recent variants of L-SHADE.

From the group of BCHMs used in existing implementations, *transformation* is the best. In that group, the idea of decreasing weights improves *weighted penalty*, which improves *additive penalty*. Therefore, experimenters should bear in mind that even official CMA-ES implementations come with different BCHMs, so attention should be paid to the choice of implementation.

Even though repair by projection is fast and simple to implement, all its variants have a negative impact on CMA-ES performance. Supposedly they induce a disadvantageous change of the distribution of points generated by CMA-ES.

It was also observed that the influence of BCHMs on one optimization method (e.g., DE) may not be the same as their influence on another (e.g., CMA-ES). One exception to this rule is *resampling*, which is good in CMA-ES and in many variants of DE.

References

[1] O. Kramer, A review of constraint-handling techniques for evolution strategies, Applied Comp. Int. Soft Computing 2010. doi:10.1155/2010/185063.

[2] E. Mezura-Montes, C. A. Coello Coello, Constraint-handling in nature-inspired numerical optimization: Past, present and future, Swarm and Evolutionary Computation 1 (4) (2011) 173 – 194. doi:10.1016/j.swevo.2011.10.001.

[3] C. A. C. Coello, List of references on constraint-handling techniques used with evolutionary algorithms, http://www.cs.cinvestav.mx/~constraint/, (Accessed: 6 March 2019).

[4] N. Hansen, A. S. P. Niederberger, L. Guzzella, P. Koumoutsakos, A method for handling uncertainty in evolutionary optimization with an application to feedback control of combustion, IEEE Trans. Evol. Comput. 13 (1) (2009) 180–197. doi:10.1109/TEVC.2008.924423.

[5] R. Biedrzycki, J. Arabas, A. Jasik, M. Szymański, P. Wnuk, P. Wasylczyk, A. Wójcik-Jedlińska, Application of evolutionary methods to semiconductor double-chirped mirrors design, in: T. Bartz-Beielstein, J. Branke, B. Filipič, J. Smith (Eds.), Parallel Problem Solving from Nature – PPSN XIII, Vol. 8672 of Lecture Notes in Computer Science, Springer, 2014, pp. 761–770. doi:10.1007/978-3-319-10762-2_75.

[6] R. Biedrzycki, D. Jackiewicz, R. Szewczyk, Reliability and efficiency of differential evolution based method of determination of Jiles-Atherton model parameters for X30Cr13 corrosion resisting martensitic steel, Journal of Automation, Mobile Robotics and Intelligent Systems 8 (4) (2014) 63–68. doi:10.14313/JAMRIS_4-2014/39.

[7] J. Arabas, L. Bartnik, S. Szostak, D. Tomaszewski, Global extraction of MOSFET parameters using the EKV model: Some properties of the underlying optimization task, in: 2009 MIXDES-16th International Conference Mixed Design of Integrated Circuits Systems, 2009, pp. 67–72.

[8] N. Hansen, S. Finck, R. Ros, A. Auger, Real-parameter black-box optimization bench-marking 2009: Noiseless functions definitions., Tech. rep., Inria (RR-6829) (2009).

[9] N. H. Awad, M. Ali, J. Liang, B. Qu, P. N. Suganthan, Problem definitions and evaluation criteria for the CEC 2017 special session and competition on real-parameter optimization, Tech. rep., Nanyang Technol. Univ., Singapore and Jordan Univ. Sci. Technol. and Zhengzhou Univ., China (2016).

[10] R. Biedrzycki, J. Arabas, D. Jagodziński, Bound constraints handling in differential evolution: An experimental study, Swarm and Evolutionary Computation 50 (2019) 100453. doi:10.1016/j.swevo.2018.10.004.

[11] N. Hansen, A. Ostermeier, Completely derandomized self-adaptation in evolution strategies, Evolutionary Computation 9 (2) (2001) 159–195. doi:10.1162/106365601750190398.

[12] N. Hansen, Errata/addenda for "A method for handling uncertainty in evolutionary optimization with an application to feedback control of combustion", http://www.cmap.polytechnique.fr/~nikolaus.hansen/TEC2009online.pdf, (Accessed: 6 March 2019).

[13] N. Sakamoto, Y. Akimoto, Modified box constraint handling for the Covariance Matrix Adaptation Evolution Strategy, in: Proceedings of the Genetic and Evolutionary Computation Conference Companion, ACM, NY, USA, 2017, pp. 183–184. doi:10.1145/3067695.3075986.

[14] N. Hansen, References to cma-es applications, http://www.cmap.polytechnique.fr/~nikolaus.hansen/cmaapplications.pdf, (Accessed: 6 March 2019).

[15] M. S. Nobile, A. Tangherloni, L. Rundo, S. Spolaor, D. Besozzi, G. Mauri, P. Cazzaniga, Computational intelligence for parameter estimation of biochemical systems, in: 2018 IEEE Congress on Evolutionary Computation (CEC), 2018, pp. 1–8. doi:10.1109/CEC.2018.8477873.

[16] G. Zhang, Y. Shi, Hybrid sampling evolution strategy for solving single objective bound constrained problems, in: 2018 IEEE Congress on Evolutionary Computation (CEC), 2018, pp. 1–7. doi:10.1109/CEC.2018.8477908.

[17] D. Liu, S. Huang, J. Zhong, Surrogate-assisted multi-tasking memetic algorithm, in: 2018 IEEE Congress on Evolutionary Computation (CEC), 2018, pp. 1–8. doi:10.1109/CEC.2018.8477830.

[18] N. Hansen, The CMA evolution strategy: A tutorial, arXiv:1604.00772.

[19] J. Arabas, A. Szczepankiewicz, T. Wroniak, Experimental comparison of methods to handle boundary constraints in Differential Evolution, in: R. Schaefer, et al. (Eds.), Parallel Problem Solving from Nature, PPSN XI, Springer Berlin Heidelberg, 2010, pp. 411–420. doi:10.1007/978-3-642-15871-1_42.

[20] K. Price, R. M. Storn, J. A. Lampinen, Differential Evolution: A Practical Approach to Global Optimization (Natural Computing Series), Springer-Verlag, Berlin, Heidelberg, 2005. doi:10.1007/3-540-31306-0.

[21] J. Brest, S. Greiner, B. Boskovic, M. Mernik, V. Zumer, Self-adapting control parameters in differential evolution: A comparative study on numerical benchmark problems, Trans. Evol. Comp 10 (6) (2006) 646–657. doi:10.1109/TEVC.2006.872133.

[22] J. Ronkkonen, S. Kukkonen, K. V. Price, Real-parameter optimization with differential evolution, in: 2005 IEEE Congress on Evolutionary Computation, Vol. 1, 2005, pp. 506–513 Vol.1. doi:10.1109/CEC.2005.1554725.

[23] N. Karaboga, B. Cetinkaya, Design of digital FIR filters using differential evolution algorithm, Circuits, Systems and Signal Processing 25 (5) (2006) 649–660. doi:10.1007/s00034-005-0721-7.

[24] N. Hansen, Sources of cma package, https://github.com/CMA-ES/pycma, (Accessed: 21 October 2019).

[25] N. Hansen, CMA-ES source code, http://cma.gforge.inria.fr/cmaes_sourcecode_page.html, (Accessed: 6 March 2019).

[26] V. Kreischer, T. T. Magalhaes, Barbosa, et al., Evaluation of bound constraints handling methods in Differential Evolution using the CEC2017 benchmark, in: XIII Brazilian Congress on Computational Intelligence, Rio de Janeiro, Brazil, 2017.

[27] Z. Michalewicz, M. Schoenauer, Evolutionary algorithms for constrained parameter optimization problems, Evol. Comput. 4 (1) (1996) 1–32. doi:10.1162/evco.1996.4.1.1.

[28] I. Loshchilov, T. Glasmachers, Black box optimization competition, https://bbcomp.ini.rub.de/, (Accessed: 5 March 2019).

[29] T. Bäck, F. Hoffmeister, H.-P. Schwefel, A survey of evolution strategies, in: Proceedings of the Fourth International Conference on Genetic Algorithms, Morgan Kaufmann, 1991, pp. 2–9.

[30] K. Deb, An efficient constraint handling method for genetic algorithms, Computer Methods in Applied Mechanics and Engineering 186 (2) (2000) 311 – 338. doi:10.1016/S0045-7825(99)00389-8.

[31] E. Mezura-Montes, C. A. Coello Coello, E. I. Tun-Morales, Simple feasibility rules and differential evolution for constrained optimization, in: R. Monroy, G. Arroyo-Figueroa, L. E. Sucar, H. Sossa (Eds.), MICAI 2004: Advances in Artificial Intelligence, Springer Berlin Heidelberg, Berlin, Heidelberg, 2004, pp. 707–716. doi:10.1007/978-3-540-24694-7_73.

[32] H. Trautmann, O. Mersmann, D. Arnu, cmaes: Covariance Matrix Adapting Evolutionary Strategy, R package version 1.0-11 (2011).

[33] Z. Michalewicz, G. Nazhiyath, Genocop III: a co-evolutionary algorithm for numerical optimization problems with nonlinear constraints, in: Proceedings of 1995 IEEE International Conference on Evolutionary Computation, Vol. 2, 1995, pp. 647–651 vol.2. doi:10.1109/ICEC.1995.487460.

[34] N. Hansen, A. Auger, D. Brockhoff, D. Tusar, T. Tusar, COCO: Performance assessment, arXiv:1605.03560.

[35] N. Hansen, The CMA evolution strategy: a comparing review, in: J. A. Lozano (Ed.), Towards a new evolutionary computation: advances on estimation of distribution algorithms, Springer, 2006, pp. 75–102. doi:10.1007/3-540-32494-1_4.

[36] N. Hansen, A. Auger, S. Finck, R. Ros, Real-parameter black-box optimization benchmarking 2009: Experimental setup, Tech. rep., Inria (RR-6828) (2009).

[37] N. Hansen, D. Brockhoff, O. Mersmann, T. Tusar, D. Tusar, O. A. ElHara, P. R. Sampaio, A. Atamna, K. Varelas, U. Batu, D. M. Nguyen, F. Matzner, A. Auger, COmparing Continuous Optimizers: numbbo/COCO on Github (Mar. 2019). doi:10.5281/zenodo.2594848.

[38] A. Auger, N. Hansen, A restart CMA evolution strategy with increasing population size, in: 2005 IEEE Congress on Evolutionary Computation, Vol. 2, 2005, pp. 1769–1776 Vol. 2.

[39] N. Hansen, Benchmarking a BI-population CMA-ES on the BBOB-2009 function testbed, in: Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers, GECCO '09, ACM, New York, NY, USA, 2009, pp. 2389–2396. doi:10.1145/1570256.1570333.

[40] R. Biedrzycki, Results of the IPOP-CMA-ES variants on CEC 2017 benchmark, http://staff.elka.pw.edu.pl/~rbiedrzy/publ/IPOP.html, (Accessed: 31 October 2019).

[41] R. Tanabe, A. S. Fukunaga, Improving the search performance of SHADE using linear population size reduction, in: 2014 IEEE Congress on Evolutionary Computation (CEC), 2014, pp. 1658–1665. doi:10.1109/CEC.2014.6900380.

[42] A. P. Piotrowski, J. J. Napiorkowski, Step-by-step improvement of JADE and SHADE-based algorithms: Success or failure?, Swarm and Evolutionary Computation 43 (2018) 88–108. doi:10.1016/j.swevo.2018.03.007.

[43] R. Salgotra, U. Singh, G. Singh, Improving the adaptive properties of LSHADE algorithm for global optimization, in: 2019 International Conference on Automation, Computational and Technology Management (ICACTM), 2019, pp. 400–407. doi:10.1109/ICACTM.2019.8776747.

[44] R. Salgotra, U. Singh, S. Saha, A. Nagar, New improved SALSHADE-cnEpSin algorithm with adaptive parameters, in: 2019 IEEE Congress on Evolutionary Computation (CEC), 2019, pp. 3150–3156. doi:10.1109/CEC.2019.8789983.