A Simple Approach for Constrained Optimization - An Evolution Strategy that Evolves Rays

Patrick Spettel and Hans-Georg Beyer

Research Center Process and Product Engineering
Vorarlberg University of Applied Sciences
Dornbirn, 6850, Austria
email: {Patrick.Spettel,Hans-Georg.Beyer}@fhv.at

***Abstract*—This paper applies an evolution strategy (ES) that evolves rays to single-objective real-valued constrained optimization problems. The algorithm is called Ray-ES. It was proposed as an ad hoc optimization approach for dealing with the unconstrained real-parameter optimization problem class called HappyCat. To our knowledge, the application of the Ray-ES to constrained problems is new. It serves as a simple alternative to other approaches such as for example differential evolution (DE). This paper describes how the Ray-ES can be applied to a constrained setting. The algorithm is tested on a variety of different test problems. Additionally, it is compared to DE approaches.**

I. Introduction

This paper presents a simple evolution strategy (ES) for constrained optimization. It is based on an idea presented in [1]: the so-called Ray-ES. The Ray-ES evolves ray directions starting from a fixed point in the search domain and evaluates them using (simple) line searches. The line search for a particular ray tries to find the best objective function value on this ray. The idea is to find a ray direction that contains the optimum [1]. The Ray-ES is based on the $(\mu/\mu_I, \lambda)$-σ-Self-Adaptation-ES ($(\mu/\mu_I, \lambda)$-σSA-ES) [2].

In [1], the Ray-ES was proposed to deal with the so-called HappyCat function. The HappyCat function was introduced in [1]. The motivation for the HappyCat function came from the ridge function. For the ridge function, once the ridge direction has been learnt, a straight path has to be followed to reach the optimum. The HappyCat function is constructed with the ridge in mind: the path to follow is not straight but spherical. It has been shown in [1] that the Covariance Matrix Adaptation (CMA) ES [3], [4], differential evolution (DE) algorithms [5], [6], [7] and particle swarm optimization (PSO) algorithms [8], [9] do not perform well on this function. Hence, the Ray-ES was proposed for dealing with this special function class in an ad hoc manner.

To our knowledge, this idea has not yet been considered for constrained optimization. Therefore, it is of interest to understand how the approach performs on optimization problems with constraints. The idea is particularly interesting because it is simple and it is applicable to the given problem constraints without any pre-processing. Moreover, there is no assumption on the constraints. The algorithm supports non-linear constraints natively. We only consider the simplest form of the Ray-ES as introduced in [1] (with a slight variation for the line search). As a consequence we present first results. The approach has potential for extension.

The rest of the paper is organized as follows. Section II describes the optimization problem under consideration. Then, we describe the Ray-ES algorithm in Section III. In Section IV we describe the experimental setup and present the experimentation results. Finally, Section V contains a conclusion and an outlook.

Notations Boldface $\mathbf{x} \in \mathbb{R}^N$ is a column vector with N real-valued components and $\mathbf{x}^T$ is its transpose. x_n and equivalently $(\mathbf{x})_n$ denote the n-th element of a vector $\mathbf{x}$. $x_{(k:N)}$ and equivalently $(\mathbf{x})_{(k:N)}$ are the order statistic notations, i.e., they denote the k-th smallest of the N elements of the vector $\mathbf{x}$. $||\mathbf{x}|| = \sqrt{\sum_{n=1}^{N} {x_n}^2}$ denotes the euclidean norm (ℓ_2 norm). $\mathbf{X}$ is a matrix, $\mathbf{X}^T$ its transpose. $\mathbf{0}$ is the vector or matrix (depending on the context) with all elements equal to zero. $\mathbf{I}$ is the identity matrix. $\mathcal{N}(\boldsymbol{\mu}, \mathbf{C})$ denotes the multivariate normal distribution with mean $\boldsymbol{\mu}$ and covariance $\mathbf{C}$, $\mathcal{N}(\mu, \sigma^2)$ the normal distribution with mean μ and variance σ^2.

II. Optimization Problem

We consider single-objective real-valued constrained optimization problems of the form

$$f(\mathbf{x}) \to \min! \tag{1a}$$

$$\text{s.t. } \forall n = 1, \dots, N : \check{x}_n \leq x_n \leq \hat{x}_n, \tag{1b}$$

$$\forall i = 1, \dots, K : g_i(\mathbf{x}) \leq 0. \tag{1c}$$

Equation (1a) states the optimization goal (here minimization) and the objective function $f : \mathbb{R}^N \to \mathbb{R}$. The set of objective parameters is $\mathbf{x} \in \mathbb{R}^N$ where N is the search space dimensionality. Box constraints (Equation (1b)) provide bounds for the objective parameters where $\check{\mathbf{x}}$ and $\hat{\mathbf{x}}$ contain the lower and upper bounds, respectively. Inequality constraints (Equation (1c)) are functions that depend on the objective parameters and delimit the search space. This formulation can be done without loss of generality because a maximization problem can be turned into an equivalent minimization problem. Non-zero right hand sides for the constraints can be moved to the left hand sides. Opposite inequalities can be dealt with by multiplication with -1. And strict inequality constraints $g_i(\mathbf{x}) < 0$ can be handled as $g_i(\mathbf{x}) \leq 0 - \epsilon \implies g_i(\mathbf{x}) + \epsilon \leq 0$ for a small $\epsilon \in \mathbb{R}$ with $\epsilon > 0$.

III. Algorithm

A. Core Ray-ES

Algorithm 1 shows the algorithm in pseudo-code. An individual $\mathfrak{a}_l$ is represented as a tuple $\mathfrak{a}_l = (f_l, \mathbf{x}_l, \mathbf{r}_l, \sigma_l)$ containing the fitness f_l, the individual's object parameter vector $\mathbf{x}_l$, the ray direction $\mathbf{r}_l$, and the mutation strength σ_l. Parameters are initialized in Line 1. The ray origin $\mathbf{o}$ and the initial ray direction vector $\mathbf{r}$ are initialized in Lines 2 to 4. The individual and the generation for the best-so-far (bsf) tracking are initialized in Line 5 and the generation counter in Line 6. Then, the generation loop is entered in Line 7. In every generation λ offspring are created and the feasible offspring are gathered in a list (Lines 8 to 19). The offspring's mutation strength is generated from the parental mutation strength using a log-normal distribution (Line 10). A new ray is computed using this mutation strength in Line 11. The ray vector is normalized (Line 12) and the line search for the current offspring is called (Line 13). The parameters to the line search are, respectively, the normalized ray $\tilde{\mathbf{r}}_l$, the maximal line search ray length L, the number of line partition segments k, the origin $\mathbf{o}$, the stopping segment size ϵ, the objective function f, the inequality constraint function $\mathbf{g}$, the lower bound vector $\check{\mathbf{x}}$, and the upper bound vector $\hat{\mathbf{x}}$. If at least one feasible point on the line is found[1], the offspring individual is appended to the list of feasible offspring and the bsf is updated (Lines 13 to 18). After the offspring generation loop, the number of feasible offspring is determined (Line 20). If there are feasible offspring, they are ranked according to the order relation

$$\mathfrak{a}_l \succ \mathfrak{a}_m \Leftrightarrow f(\tilde{\mathbf{x}}_l) < f(\tilde{\mathbf{x}}_m) \tag{2}$$

in Line 22. The parental $\mathbf{r}$ and σ for the next generation are updated in Lines 24 to 26, respectively, and finally the generation counter is incremented in Line 28. The centroid computation for the update of $\mathbf{r}$ and σ takes into account the feasibility. That is, the number of feasible offspring can potentially be smaller than the parameter μ. If this is the case, only all the feasible offspring are considered for the centroid generation. Otherwise, the centroid computation is done as usual using the μ best offspring. The generation loop is terminated if the maximum number of generations is reached or the σ value decreases below a threshold. Additionally, the bsf is taken into account. If it has not been updated for the previous G_{lag} generations, the generation loop is terminated as well. (Line 29).

An essential part of the Ray-ES is the line search. The line search evaluates a ray by optimizing the objective function along this particular ray. The fitness for a ray is the best objective function value that is found along the direction of this ray assuming a pre-defined maximal length. Algorithm 2 shows the pseudo-code for the LineSearch algorithm. The line starting at position $\mathbf{o}$ in search space with direction $\mathbf{r}$ and maximal length L is searched for the minimum by first probing it at $2k+1$ positions (k positions in the negative direction, k positions in the positive direction and the origin itself). The Δr is the length of one partition segment. The best of the feasible positions is taken for the next iteration. For the next iteration Δr is divided by k. This loop continues as long as the Δr is larger than some threshold ϵ with the result of getting to the best value on the line with increasing level of detail. The best position that was found is returned in the end.

Algorithm 1 Ray-ES.

```
1:  Initialize parameters $\mu, \lambda, \sigma, \tau, G_{\text{lag}}, gen_{\text{stop}}, \sigma_{\text{stop}}, L, k, \epsilon$
2:  Initialize $\mathbf{o}$
3:  $\mathbf{r} \leftarrow \mathcal{N}(0, \mathbf{I})$          ▷ Initialize random ray direction
4:  $\mathbf{r} \leftarrow \frac{\mathbf{r}}{||\mathbf{r}||}$          ▷ Normalize random ray direction
5:  $(\mathfrak{a}_{\text{bsf}}, gen_{\text{bsf}}) \leftarrow ((\infty, \mathbf{0}, \mathbf{r}, \sigma), 0)$
6:  $gen \leftarrow 0$
7:  repeat
8:      feasibleOffspring $\leftarrow$ empty list
9:      for $l \leftarrow 1$ to $\lambda$ do
10:         $\tilde{\sigma}_l \leftarrow \sigma e^{\tau \mathcal{N}_l(0,1)}$
11:         $\tilde{\mathbf{y}}_l \leftarrow \mathbf{r} + \tilde{\sigma}_l \mathcal{N}_l(\mathbf{0}, \mathbf{I})$
12:         $\tilde{\mathbf{r}}_l \leftarrow \frac{\tilde{\mathbf{y}}_l}{||\tilde{\mathbf{y}}_l||}$
13:         $(\tilde{\mathbf{x}}_l, \tilde{f}_l,$ foundFeasible$) \leftarrow$
                    LineSearch($\tilde{\mathbf{r}}_l$, $L$, $k$, $\mathbf{o}$, $\epsilon$, $f$, $\mathbf{g}$, $\check{\mathbf{x}}$, $\hat{\mathbf{x}}$)
14:         if foundFeasible then
15:             $\tilde{\mathfrak{a}}_l \leftarrow (\tilde{f}_l, \tilde{\mathbf{x}}_l, \tilde{\mathbf{r}}_l, \tilde{\sigma}_l)$
16:             $(\mathfrak{a}_{\text{bsf}}, gen_{\text{bsf}}) \leftarrow \begin{cases} (\tilde{\mathfrak{a}}_l, gen+1) & \text{if } \tilde{\mathfrak{a}}_l \succ \mathfrak{a}_{\text{bsf}} \\ (\mathfrak{a}_{\text{bsf}}, gen_{\text{bsf}}) & \text{otherwise} \end{cases}$
17:             Append $\tilde{\mathfrak{a}}_l$ to feasibleOffspring
18:         end if
19:     end for
20:     $n_{\text{feasible}} \leftarrow$ number of elements in list feasibleOffspring

21:     if $n_{\text{feasible}} > 0$ then
22:         rankOffspringPopulation(feasibleOffspring)
                        according to "$\succ$" (Equation (2))
23:         $\mu_{\text{adapted}} \leftarrow \min(\mu, n_{\text{feasible}})$
24:         $\mathbf{r} \leftarrow \frac{1}{\mu_{\text{adapted}}} \sum_{m=1}^{\mu_{\text{adapted}}} \tilde{\mathbf{r}}_{m;n_{\text{feasible}}}$
25:         $\mathbf{r} \leftarrow \frac{\mathbf{r}}{||\mathbf{r}||}$
26:         $\sigma \leftarrow \frac{1}{\mu_{\text{adapted}}} \sum_{m=1}^{\mu_{\text{adapted}}} \tilde{\sigma}_{m;n_{\text{feasible}}}$
27:     end if
28:     $gen \leftarrow gen + 1$
29: until $gen > gen_{\text{stop}} \vee \sigma < \sigma_{\text{stop}} \vee gen - gen_{\text{bsf}} \geq G_{\text{lag}}$
```

B. Modified LineSearch

The LineSearch algorithm (Algorithm 2) potentially requires unnecessary constraint and objective function evaluation budget by querying infeasible points. To overcome this, we propose a different line search. It is outlined in Algorithm 3.

[1]Note that the algorithm is kept general such that infeasible ray origins are supported. The "foundFeasible" variable would not be necessary if we would always start with a feasible origin.

Algorithm 2 LineSearch.

```
function LineSearch($\mathbf{r}$, L, k, $\mathbf{o}$, $\epsilon$, $f$, $\mathbf{g}$, $\check{\mathbf{x}}$, $\hat{\mathbf{x}}$)
    assert($||\mathbf{r}|| = 1$)
    $\mathbf{x}_{\text{best}} \leftarrow \mathbf{o}$
    $f_{\mathbf{x}_{\text{best}}} \leftarrow f(\mathbf{x}_{\text{best}})$
    foundFeasible $\leftarrow$ false
    $\Delta r \leftarrow \frac{L}{k}$
    while $\Delta r > \epsilon$ do
        $\mathbf{x}_{\text{currbest}} \leftarrow \mathbf{x}_{\text{best}}$
        $f_{\text{currbest}} \leftarrow f_{\mathbf{x}_{\text{best}}}$
        for $p \leftarrow 1$ to $2k+1$ do
            $\mathbf{x}_p \leftarrow \mathbf{x}_{\text{best}} + \Delta r \cdot (p-k-1) \cdot \mathbf{r}$
            if $\mathbf{x}_p$ is feas. acc. to eqs. (1b) and (1c) then
                foundFeasible $\leftarrow$ true
                $f_p \leftarrow f(\mathbf{x}_p)$
                if $f_p < f_{\text{currbest}}$ then
                    $f_{\text{currbest}} \leftarrow f_p$
                    $\mathbf{x}_{\text{currbest}} \leftarrow \mathbf{x}_p$
                end if
            end if
        end for
        $\mathbf{x}_{\text{best}} \leftarrow \mathbf{x}_{\text{currbest}}$
        $f_{\mathbf{x}_{\text{best}}} \leftarrow f_{\text{currbest}}$
        $\Delta r \leftarrow \frac{\Delta r}{k}$
    end while
    return($\mathbf{x}_{\text{best}}$, $f_{\mathbf{x}_{\text{best}}}$, foundFeasible)
end function
```

The parameters (Line 1) include ray information (ray direction $\mathbf{r}$ and the ray origin $\mathbf{o}$), termination criteria (step size threshold ϵ and maxIter), step size information (the initial step size α_{init}, the step size increase factor α_{inc}, and the step size decrease factor α_{dec}), the objective function f, the constraints (the inequality constraint function $\mathbf{g}$, the lower bound vector $\check{\mathbf{x}}$, and the upper bound vector $\hat{\mathbf{x}}$), and the searchDirections. The main idea is to search along the ray $\mathbf{r}$ starting at the origin $\mathbf{o}$. The search directions are specified by the set searchDirections which can contain -1, 1, or both (Lines 9 to 36). The search step size α is initially set to α_{init}. Using the current step size, the search is done along the line as long as the particular point corresponding to the current step is feasible, better as the previous query point, and the difference in the objective function value between the current and the previous query point is greater than 10^{-10}. If one of these conditions is violated, we decrease the step size and repeat the process starting at the previous feasible point. The intuition behind this is to get to the optimum with high precision. If the step size decreases below some threshold ϵ or the maximum number of iterations is reached, we stop. In addition, we increase the step size every time a newly queried point is feasible and better than the previous one. The idea behind this is that multiple successes indicate that larger steps can be beneficial. The best point found during the whole process is tracked and finally returned (Lines 4, 6, 7, 28 to 30, and 37). Note that we indicate the "short-circuit and" by the **and** keyword. By making use of this, we can avoid some objective function evaluations.

Algorithm 3 Modified LineSearch.

```
function ModifiedLineSearch($\mathbf{r}$, $\mathbf{o}$, $\epsilon$, maxIter, $\alpha_{\text{init}}$, $\alpha_{\text{inc}}$, $\alpha_{\text{dec}}$, $f$, $\mathbf{g}$, $\check{\mathbf{x}}$, $\hat{\mathbf{x}}$, searchDirections)
    assert($||\mathbf{r}|| = 1$)
    assert(searchDirections $\subseteq \{-1, 1\}$)
    foundFeasible $\leftarrow$ false
    $f_o \leftarrow f(\mathbf{o})$
    $\mathbf{x}_{\text{best}} \leftarrow \mathbf{o}$
    $f_{\text{best}} \leftarrow f_o$
    isFeasible$_o \leftarrow \mathbf{true} \Leftrightarrow$ eqs. (1b) and (1c) hold for $\mathbf{o}$
    for $d \in$ searchDirections do
        isFeasible$_x \leftarrow$ isFeasible$_o$
        $\mathbf{x}_{\text{prev}} \leftarrow \mathbf{o}$
        $f_{x_{\text{prev}}} \leftarrow f_o$
        $\alpha \leftarrow \alpha_{\text{init}}$
        iter $\leftarrow 0$
        while $\alpha > \epsilon$ and iter $<$ maxIter do
            $\mathbf{x} \leftarrow \mathbf{x}_{\text{prev}}$
            $f_x \leftarrow f_{x_{\text{prev}}}$
            repeat    ▷ Search loop with current step size
                $\mathbf{x}_{\text{prev}} \leftarrow \mathbf{x}$
                $f_{x_{\text{prev}}} \leftarrow f_x$
                $\mathbf{x} \leftarrow \mathbf{x} + d\alpha\mathbf{r}$
                isFeasible$_x \leftarrow$
                    $\mathbf{true} \Leftrightarrow$ eqs. (1b) and (1c) hold for $\mathbf{x}$
                if isFeasible$_x$ then
                    $f_x \leftarrow f(\mathbf{x})$
                end if
                if isFeasible$_x$ and $f_x < f_{\text{best}}$ then
                    $\alpha \leftarrow \alpha\alpha_{\text{inc}}$
                    foundFeasible $\leftarrow$ true
                    $\mathbf{x}_{\text{best}} \leftarrow \mathbf{x}$
                    $f_{\text{best}} \leftarrow f_x$
                end if
            until $\neg$(isFeasible$_x$ and $f_x < f_{x_{\text{prev}}}$
                    and $|f_x - f_{x_{\text{prev}}}| > 10^{-10}$)
            $\alpha \leftarrow \alpha/\alpha_{\text{dec}}$
            iter $\leftarrow$ iter $+ 1$
        end while
    end for
    return($\mathbf{x}_{\text{best}}$, $f_{\text{best}}$, foundFeasible)
end function
```

For the Ray-ES with the different LineSearch we distinguish two cases. The first case is the ES in adapting phase and the second case is a more converged ES. This means that while in the former case the ray directions between two generations can have larger deviations, in the latter case we assume the rays to be approximately parallel (within some threshold). We make use of this by calling the line search in a different way. In the first case we call it with $\alpha_{\text{init}} = 2|(\hat{\mathbf{x}})_{(N:N)} - (\check{\mathbf{x}})_{(1:N)}|$. Intuitively, this means that we make sure to search the whole line across the box constraints. In the second case we make use of the similarity between the rays of two subsequent generations gen (parental generation) and $gen+1$ (offspring generation) and only search in the proximity of the best point of the ray in generation gen. Technically, we measure the similarity, say s, using the dot product $s = \mathbf{r}^{(gen)^T}\mathbf{r}^{(gen+1)}$. If $|s-1| < \epsilon_s$ (directions parallel within some threshold ϵ_s), we say that we are in case two. Then we project the best point on $\mathbf{r}^{(gen)}$ (that we already know from the line search of the previous generation) onto $\mathbf{r}^{(gen+1)}$. Because the rays are almost parallel and have the same origin (by definition of the algorithm) we assume the optimal point for the new ray to be somewhere in the same area. Therefore, we call the Modified LineSearch with the projected best point as the origin and a small initial step size. Algorithm 4 shows the pseudo code of this case distinction.

Algorithm 4 Modified LineSearch case distinction.

```
1:  assert($||\mathbf{r}^{(gen)}|| = 1$)
2:  assert($||\mathbf{r}^{(gen+1)}|| = 1$)
3:  $s \leftarrow \mathbf{r}^{(gen)^T}\mathbf{r}^{(gen+1)}$
4:  if $|s-1| < \epsilon_s$ then
5:      $l \leftarrow (\mathbf{x}_{\text{best}}^{(gen)} - \mathbf{o})^T\mathbf{r}^{(gen+1)}$
6:      $x_{\text{best}_{\text{projected}}} \leftarrow \mathbf{o} + l\mathbf{r}^{(gen+1)}$
7:      Call ModifiedLineSearch with $x_{\text{best}_{\text{projected}}}$
            as the origin and a small step size.
8:  else
9:      Call ModifiedLineSearch with $\mathbf{o}$ as origin and $L$
            as the initial step size.
10: end if
```

A further optimization for the first case is to switch to searching in either -1 or 1 direction, but not both. The key observation for this is that after some generations of the ES, all the offspring of a particular generation gen have the same search direction d. The algorithm can identify such a situation and in generation $gen+1$ only search in direction d.

IV. Experimental Evaluation

The Ray-ES is tested with the BBOB COCO framework [10]. The experiments were run on a cluster with 5 nodes. Every node has an Intel 8-core Xeon E5420 2.50GHz processor with 8GiB of RAM running a GNU/Linux system. The post-processing tool with slight adjustments was used to generate the figures. The algorithm's parameters are set as summarized in Table I [2]. The choice of the learning parameter $\tau = 1/\sqrt{2N}$ is motivated by the τ-scaling rule $\tau \propto 1/\sqrt{N}$ that has been theoretically derived for the σSA-ES applied to the sphere model (see [11, Section 7.4.2.2]). The chosen value $L = 2|(\hat{\mathbf{x}})_{(N:N)} - (\check{\mathbf{x}})_{(1:N)}|$ is motivated by the box constraints of the problem. This value makes sure that a particular line is searched across the whole box. The value for the origin $\mathbf{o}$ has been set to a feasible point provided by the BBOB COCO framework. Because the Modified LineSearch requires

[2]We provide our code that we used for the experiments in the GitHub repository https://github.com/patsp/Ray-ES.

λ	$4N$
μ	$\lfloor \frac{\lambda}{4} \rfloor$
σ (initial value)	$\frac{1}{\sqrt{N}}$
τ	$\frac{1}{\sqrt{2N}}$
G_{lag}	$50N$
gen_{stop}	100000
σ_{stop}	10^{-6}
L	$2\|(\hat{\mathbf{x}})_{(N:N)} - (\check{\mathbf{x}})_{(1:N)}\|$, i.e., twice the absolute value of the difference between the largest upper bound of all the variables and the smallest of the lower bounds of all the variables.
k	2
ϵ	10^{-10}
ϵ_s	10^{-3}
maxIter	100
α_{inc}	1.5
α_{dec}	10
$\mathbf{o}$	For the BBOB COCO problems $\mathbf{o}$ is set to the feasible initial solution that the BBOB COCO framework provides to the optimization algorithm.

TABLE I: Parameter settings for the Ray-ES experiments.

a feasible origin, this has been done in this way such that both, the Standard LineSearch and the Modified LineSearch, can be run with the same parameters. The results are therefore comparable. Another option would have been to perform a preprocessing step of searching for a feasible solution. The values for λ, μ, initial σ, G_{lag}, gen_{stop}, σ_{stop} k, ϵ, ϵ_s, maxIter, α_{inc}, and α_{dec} have been empirically determined in manual experiments. To this end, different values for those parameters have been chosen. The Ray-ES has then been tested on the BBOB COCO problems and the best parameters have been chosen for the final experiments. The results of these final experiments are presented in this section.

As a first step linear constraints are considered. And as a second step non-linear perturbations are considered as provided by the BBOB COCO framework. The adapted framework[3] is based on the code in the branch `development`[4] in [12]. A documentation can be found in [13] under `docs/bbob-constrained/functions/build` after building it according to the instructions.

The BBOB COCO framework provides a test suite for constrained black-box optimization benchmarking. It contains 48 constrained functions with dimensions $N \in \{2, 3, 5, 10, 20, 40\}$. For every problem, random instances can

[3]We provide the adapted code in a GitHub fork of the BBOB COCO framework, https://github.com/patsp/coco. The changes are in the new branch `development-sppa-2`. This branch is based on the `development` branch of https://github.com/numbbo/coco with changes up to and including Dec 10, 2017. A list of the changes is also provided in the supplementary material (Section VI-B). The supplementary material can be found in the GitHub repository https://github.com/patsp/Ray-ES (`rayes_supplementary_material.pdf.xz`).

[4]Because the `bbob-constrained` suite is still under development, we provide a fork. This makes our results reproducible. Even though it is still under development, this suite gives a good indication of the algorithm's performance in comparison to other methods.

be generated. The 48 problems are constructed by combining 8 functions of the standard BBOB COCO suite for single-objective optimization with 6 different numbers of constraints, namely 1, 2, 6, $6 + N/2$, $6 + N$, and $6 + 3N$ constraints. The 8 functions are Sphere, Separable Ellipsoid, Linear Slope, Rotated Ellipsoid, Discus, Bent Cigar, Sum of Different Powers, and the Separable Rastrigin. The constraints are linear with nonlinear perturbations and defined by their gradient. These constraints are generated by sampling their gradient vectors from a normal distribution and ensuring that the feasible region is not empty. The generic algorithm of generating a constrained problem is outlined in [13], `docs/bbob-constrained/functions/build`. The preference of the BBOB COCO framework over, e.g., the CEC 2017 constrained optimization competition benchmark problems [14] is two-fold. One reason has been the aim to compare the Ray-ES on linear constraints and non-linear constraints. Another reason has been that the search for an initial feasible point for the origin can be expensive for some CEC 2017 problems caused by a "small" feasible region.

The optimization problem in the BBOB COCO framework is stated as

$$f(\mathbf{x}) \to \min! \tag{3a}$$

$$\text{s.t. } \mathbf{g}(\mathbf{x}) \leq \mathbf{0} \tag{3b}$$

$$\check{\mathbf{x}} \leq \mathbf{x} \leq \hat{\mathbf{x}} \tag{3c}$$

where $f : \mathbb{R}^N \to \mathbb{R}$ and $\mathbf{g} : \mathbb{R}^N \to \mathbb{R}^K$. Note that this problem formulation is equivalent to Equation (1). The Ray-ES can therefore be applied directly on this problem.

In the following sections we present bootstrapped empirical cumulative distribution functions (ECDFs) of runs of algorithms for all the constrained problems of the BBOB COCO bbob-constrained test suite aggregated[5] for dimensions 2, 3, 5, 10, 20, 40. For these, the performance of the algorithm is evaluated on 15 independent randomly generated instances of each constrained test problem. Based on the observed run lengths, ECDF graphs are generated. These graphs show the percentages of function target values reached for a given budget of function and constraint evaluations[6] per search space dimensionality. The function target values used are the standard BBOB ones: $f_{\text{opt}} + 10^k, k \in \{-8, \ldots, 2\}$.

A. Experimental Results for the Algorithm with the Standard LineSearch

1) Linear Constraints without Non-Linear Perturbations: Figure 1a shows the ECDF plots for the Ray-ES with the Standard LineSearch on the BBOB COCO problems. The results are aggregated over all the functions in the BBOB COCO constrained suite with the non-linear perturbations turned off.

2) Linear Constraints with Non-Linear Perturbations: Figure 1b shows the ECDF plots for the Ray-ES with the Standard LineSearch on the BBOB COCO problems. The results are aggregated over all the functions in the BBOB COCO constrained suite with the non-linear perturbations turned on.

The plots show that the most difficult target is only reached for the case of 2 dimensions. The performance decreases with higher dimensions. We also see that the performance of the Ray-ES for the case with non-linear perturbations is very similar. This indicates that the Ray-ES is able to cope well with the non-linear perturbations.

B. Experimental Results for the Algorithm with the Modified LineSearch

1) Linear Constraints without Non-Linear Perturbations: Figure 1c shows the ECDF plots for the Ray-ES with the Modified LineSearch on the BBOB COCO problems. The results are aggregated over all the functions in the BBOB COCO constrained suite with the non-linear perturbations turned off.

2) Linear Constraints with Non-Linear Perturbations: Figure 1d shows the ECDF plots for the Ray-ES with the Modified LineSearch on the BBOB COCO problems. The results are aggregated over all the functions in the BBOB COCO constrained suite with the non-linear perturbations turned on.

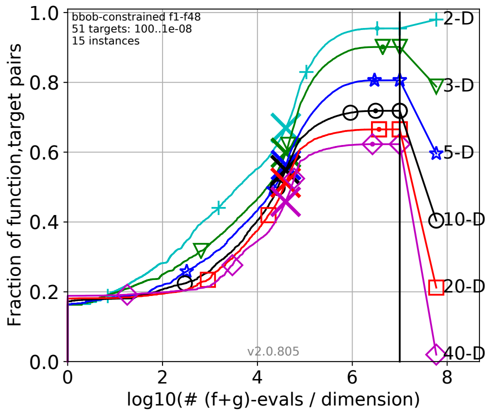
The targets that are reached with the Modified LineSearch are almost the same as for the Standard LineSearch. However, we see that the targets are reached with less evaluations. For example for the case of dimension 40. The proportion of function and target pairs reached with $\leq 10^5$ function and constraint evaluations is about 0.7 for the Modified LineSearch and slightly below 0.6 for the Standard LineSearch. This is what we expect because the idea for the Modified LineSearch is to avoid unnecessary evaluations. But we also see a disadvantage of the Modified LineSearch. The most difficult target is not reached in every run. This can be seen by the crosses that indicate the medians of the sum of objective function and constraint evaluations of instances that did not reach the most difficult target.

C. Comparison with other approaches

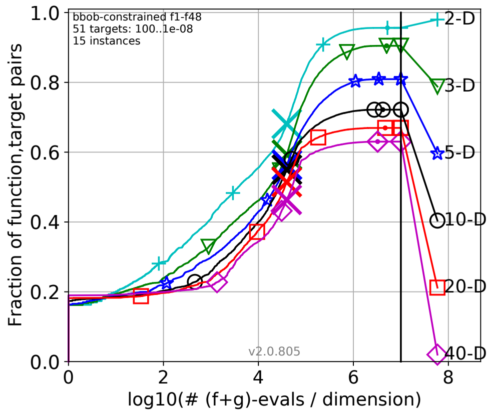
In order to compare the Ray-ES proposed in this work, other algorithms are benchmarked in the same adapted BBOB COCO suite. We have chosen two variants of DE: "Self-adaptive Differential Evolution Algorithm for Constrained Real-Parameter Optimization" (conSaDE) [6] and "Constrained Optimization by the ε Constrained Differential Evolution with an Archive and Gradient-Based Mutation" (εDEag) [7]. They showed promising results in competition benchmarks. The εDEag won the CEC 2010 constrained real-parameter optimization competition [15]. The conSaDE achieved the third place (behind a version of εDE and a PSO variant) in the CEC 2006 constrained real-parameter optimization competition [16]. For both DE variants, the

[5]We provide additional experimental results in the supplementary material (Section VI-A). There, the results are presented in more detail. In particular, the results are shown for every function separately.

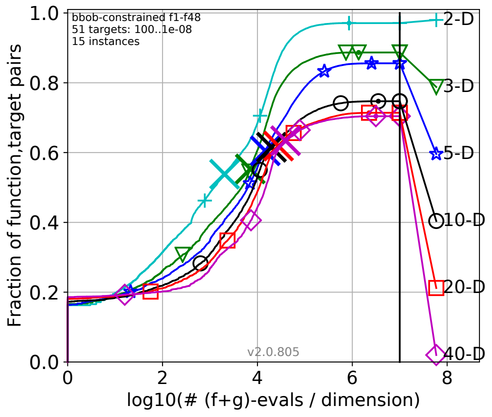
[6]In the BBOB COCO framework one call to the constraint evaluation function yields the values of *all* the constraints at the given query point.



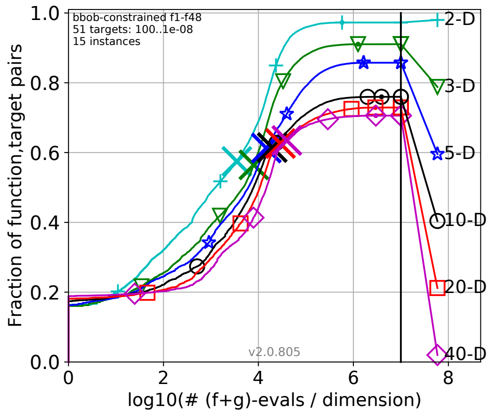
(a) Ray-ES with the Standard LineSearch (non-linear perturbations turned off in the BBOB COCO constrained suite).



(b) Ray-ES with the Standard LineSearch (non-linear perturbations turned on in the BBOB COCO constrained suite).



(c) Ray-ES with the Modified LineSearch (non-linear perturbations turned off in the BBOB COCO constrained suite).



(d) Ray-ES with the Modified LineSearch (non-linear perturbations turned on in the BBOB COCO constrained suite).

Fig. 1: Bootstrapped empirical cumulative distribution of the number of objective function and constraint evaluations divided by dimension for 51 targets with target precision in $10^{[-8..2]}$ for all functions of the BBOB COCO constrained suite aggregated for the dimensions 2, 3, 5, 10, 20, and 40. The horizontal axis shows the $\log_{10}$ of the sum of objective function and constraint evaluations. The vertical axis shows the proportion of target objective function values reached with the given number of objective function and constraint evaluations (horizontal axis). The crosses indicate the medians of the sum of objective function and constraint evaluations of instances that did not reach the most difficult target.

implementations provided by the respective authors were used[7] (adapted for the BBOB COCO framework). The algorithms for the comparison were run with default parameters.

1) Linear Constraints without Non-Linear Perturbations: Figure 2a shows the ECDF plots comparing the considered approaches on the BBOB COCO constrained suite with the non-linear perturbations turned off. I.e., the approaches conSaDE, εDEag, the newly proposed Ray-ES with the Standard LineSearch (called "rayes" in the plot), and the newly proposed Ray-ES with the Modified LineSearch (called "rayes dif" in the plot) are shown.

[7]εDEag: http://www.ints.info.hiroshima-cu.ac.jp/%7Etakahama/download/eDEa-2010.0430.tar.gz, conSaDE: http://web.mysites.ntu.edu.sg/epnsugan/PublicSite/Shared%20Documents/Codes/2006-CEC-Const-SaDE.rar.

2) Linear Constraints with Non-Linear Perturbations: Figure 2b shows the ECDF plots comparing the considered approaches on the BBOB COCO constrained suite with the non-linear perturbations turned on. I.e., the approaches conSaDE, εDEag, the Ray-ES with the Standard LineSearch (called "rayes" in the plot), and the Ray-ES with the Modified LineSearch (called "rayes dif" in the plot) are shown.

We can see that the non-linear perturbations are handled by the DE algorithms as well as the Ray-ESs. The performance is similar for both cases (with and without non-linear perturbations). The DE algorithms clearly outperform the Ray-ES for small dimensions. For dimension 40 we see that the Ray-ES achieves more targets than the εDEag. The performance of the conSaDE is the best for dimension 40. An in-depth analysis revealed that the main reason for the superior performance of conSaDE can be attributed to the local search algorithm fmincon (Matlab) used. If this is switched off, conSaDE exhibits inferior performance compared to the Ray-ES for higher dimensions. A detailed look at the performance plots for the different functions shows the following. The performance of both Ray-ES variants decreases with higher number of constraints. In particular for the case with $6 + 3N$ constraints. In this case the number of targets that are reached with the Ray-ESs is low. This trend is not visible for the DE algorithms.

Having a closer look at Figure 2, there is the remarkable observation that the Ray-ES does not show markedly performance differences between the linear and the non-linear test-bed. This is in contrast to the DE versions benchmarked where the non-linear cases exhibit certain performance degradations. Given this peculiarity of the Ray-ES one might speculate whether further algorithmic improvements regarding the performance of the linear case transfers also to the non-linear cases.

One weakness of the current Ray-ES approach is due to the fixed choice of the ray origin. In our experiments, this origin has been chosen equal to the feasible starting point provided by the BBOB COCO benchmark environment. Especially in the case of a high number of constraints, this special ray origin might introduce a bias in the search process. A more elaborated version of the Ray-ES should evolve the ray origin as well.

V. Conclusion

We have shown how the Ray-ES can be applied to constrained optimization problems. The Ray-ES was first introduced for dealing with a specially constructed function class called HappyCat. It has been tested on different constrained test problems and compared to DE approaches. It does not perform as well as certain elaborated DE algorithms that rely - at least partially - on sophisticated non-evolutionary local optimization routines. Nevertheless, the approach is a simple alternative. It is also worth noting that the Ray-ES presented in this paper is intentionally kept simple. There is potential for extension in future work. One possible extension is a more sophisticated line search. Another idea is to evolve not only the ray direction but also the ray origin. Furthermore, in this paper we have only considered isotropic mutations. In future work, it is of interest to analyze the Ray-ES with a kind of covariance matrix adaptation. Moreover, as we have only presented empirical results, a theoretical analysis is a topic for future work.

Acknowledgment

This work was supported by the Austrian Science Fund FWF under grant P29651-N32.

References

[1] H.-G. Beyer and S. Finck, "HappyCat–a simple function class where well-known direct search algorithms do fail," *Parallel Problem Solving from Nature-PPSN XII*, pp. 367–376, 2012.

[2] H.-G. Beyer and H.-P. Schwefel, "Evolution strategies–a comprehensive introduction," *Natural Computing*, vol. 1, no. 1, pp. 3–52, 2002.

[3] N. Hansen and A. Ostermeier, "Completely derandomized self-adaptation in evolution strategies," *Evolutionary Computation*, vol. 9, no. 2, pp. 159–195, Jun. 2001.

[4] N. Hansen, S. D. Müller, and P. Koumoutsakos, "Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES)," *Evolutionary Computation*, vol. 11, no. 1, pp. 1–18, 2003.

[5] R. Storn and K. Price, "Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces," *Journal of Global Optimization*, vol. 11, no. 4, pp. 341–359, 1997.

[6] V. L. Huang, A. K. Qin, and P. N. Suganthan, "Self-adaptive differential evolution algorithm for constrained real-parameter optimization," in *IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 2006, pp. 17–24.

[7] T. Takahama and S. Sakai, "Constrained optimization by the ε constrained differential evolution with an archive and gradient-based mutation," in *IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 2010, pp. 1–9.

[8] A. Banks, J. Vincent, and C. Anyakoha, "A review of particle swarm optimization. Part I: background and development," *Natural Computing*, vol. 6, no. 4, pp. 467–484, Dec 2007.

[9] A. Banks, J. Vincent, and C. Anyakoha, "A review of particle swarm optimization. Part II: hybridisation, combinatorial, multicriteria and constrained optimization, and indicative applications," *Natural Computing*, vol. 7, no. 1, pp. 109–124, Mar 2008.

[10] S. Finck, N. Hansen, R. Ros, and A. Auger, *COCO Documentation, Release 15.03*. [Online]. Available: http://coco.gforge.inria.fr/

[11] H.-G. Beyer, *The Theory of Evolution Strategies*, ser. Natural Computing Series. Springer, 2001.

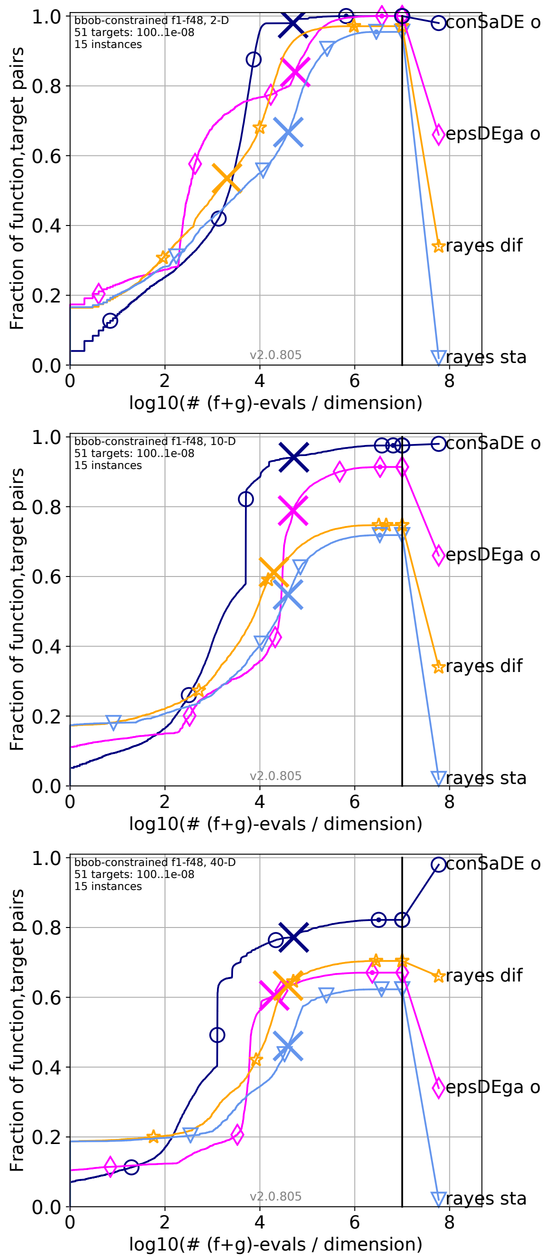
[12] D. Brockhoff, N. Hansen, O. Mersmann, R. Ros, D. Tusar, and T. Tusar, *COCO Code Repository*. [Online]. Available: http://github.com/numbbo/coco

[13] D. Brockhoff, N. Hansen, O. Mersmann, R. Ros, D. Tusar, and T. Tusar, *COCO Documentation Repository*. [Online]. Available: http://github.com/numbbo/coco-doc

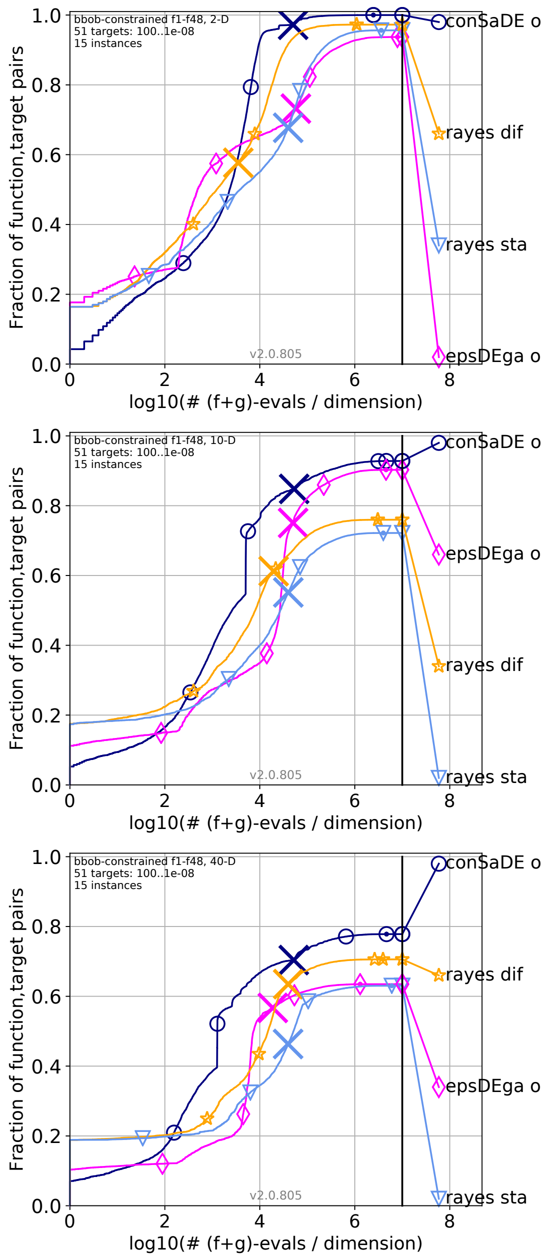
[14] G. Wu, R. Mallipeddi, and P. N. Suganthan, "Problem definitions and evaluation criteria for the CEC 2017 competition on constrained real-parameter optimization," 2017. [Online]. Available: http://www3.ntu.edu.sg/home/epnsugan/index_files/CEC2017/CEC2017.htm

[15] R. Mallipeddi and P. N. Suganthan, "Problem definitions and evaluation criteria for the CEC 2010 competition on constrained real-parameter optimization," 2010. [Online]. Available: http://www3.ntu.edu.sg/home/epnsugan/index_files/CEC10-Const/CEC10-Const.htm

[16] J. J. Liang, T. P. Runarsson, E. Mezura-Montes, M. Clerc, P. N. Suganthan, C. A. Coello Coello, and K. Deb, "Problem definitions and evaluation criteria for the CEC 2006 competition on constrained real-parameter optimization," 2010. [Online]. Available: http://www3.ntu.edu.sg/home/epnsugan/index_files/CEC10-Const/CEC10-Const.htm



(a) Non-linear perturbations turned off in the BBOB COCO constrained suite.



(b) Non-linear perturbations turned on in the BBOB COCO constrained suite.

Fig. 2: Bootstrapped empirical cumulative distribution of the number of objective function and constraint evaluations divided by dimension for 51 targets with target precision in $10^{[-8..2]}$ for all the functions and dimension of the BBOB COCO constrained suite: comparison of all the approaches. The horizontal axis shows the $\log_{10}$ of the sum of objective function and constraint evaluations. The vertical axis shows the proportion of target objective function values reached with the given number of objective function and constraint evaluations (horizontal axis). The crosses indicate the medians of the sum of objective function and constraint evaluations of instances that did not reach the most difficult target.