Benchmarking several strategies to update the penalty parameters in AL-CMA-ES on the bbob-constrained testbed

Preprint version

Paul Dufossé
Institut Polytechnique de Paris
Inria Paris-Saclay, CMAP
Paris area, France

Asma Atamna
Institute for Neural Computation
Department for Computer Science
Ruhr-University Bochum
Bochum, Germany

ABSTRACT

In this paper, we benchmark several versions of a population-based evolution strategy with covariance matrix adaptation, handling constraints with an Augmented Lagrangian fitness function. The versions only differ in the strategy to adapt the penalty parameter of the fitness function. We compare the resulting algorithms, AL-CMA-ES, with random search and Powell's derivative-free COBYLA on the recently released bbob-constrained test suite for constrained continuous optimization in dimensions ranging from 2 to 40. The experimental results allow identifying classes of problems where one algorithm is more advantageous to use. They also reveal features of the merit function used for performance assessment and in particular situations where even on simple problems the targets can be hard to meet for algorithms based on Lagrange multipliers.

CCS CONCEPTS

• **Computing methodologies** → **Continuous space search**.

KEYWORDS

Benchmarking, Black-box optimization, constrained optimization

1 INTRODUCTION

This paper compares four variants of the Augmented Lagrangian Covariance Matrix Adaptation Evolution Strategy (AL-CMA-ES) on the `bbob-constrained` testbed. It also reports data obtained running random search and Powell's Constrained Optimization BY Linear Approximations (COBYLA) algorithm [16] on the same test functions for baseline comparison.

Augmented Lagrangian (AL) methods handle the original constrained optimization problem by sequentially solving unconstrained sub-problems, optimizing an AL fitness function. This function consists of the Lagrangian function of the constrained problem augmented with a penalty term. Dynamic parameters are updated after each sub-problem is approximately solved. Initially introduced by Hestenes [14] and Powell [16], AL constraint handling was first combined with Evolution Strategies (ES) by Arnold and Porter [1], where the authors analyze a $(1+1)$-ES with AL on a sphere function with a single linear constraint and an update for the AL penalty parameters was proposed. This update was then adopted by Atamna et al. [2], who applied AL constraint handling to a $(\mu/\mu_\omega, \lambda)$-CMA-ES [7] for the first time. Later on, they extended the algorithm to handle multiple constraints [3] and analyzed the convergence of a step-size adaptive $(\mu/\mu_\omega, \lambda)$-ES with AL on sphere and ellipsoid objectives under multiple active linear constraints.

More recently, Dufossé and Hansen [6] conducted an extensive study of AL approaches for CMA-ES determining the influence of hyperparameters, individual adaptive penalty parameters and meta-modeling on a set of low-dimensional problems. They propose a new default parameter setting for the AL hyperparameters, thereby improving upon the previous work [1–3] on some problems. Additionally, the authors identified failure cases of the AL method and showed how surrogate linear modeling of the constraints helps to overcome these difficulties.

The objective of this work is to provide an extensive comparison of the different strategies to update the penalty parameters in AL-CMA-ES, which are implemented in the pycma package [8], following the carefully designed test problems and methodology of COCO [11] for constrained optimization [5].

The paper is organized as follows: in section 2 we present the common framework of AL-CMA-ES and the different variants we consider in the experiments. Section 3 details the experimental setup such as parameters settings, heuristics for initialization of adaptive parameters and how we perform restarts. We specify the hardware and report timings in section 4. Finally, selected results are discussed in section 5.

2 ALGORITHM PRESENTATION

Recall that we consider the following optimization problem

$$\underset{\boldsymbol{x}\in\mathbb{R}^n}{\text{minimize}}\ f(\boldsymbol{x}) \quad \text{subject to } g_k(\boldsymbol{x}) \leq 0 \text{ for } k = 1, \ldots, m \tag{1}$$

by minimizing the dynamic Augmented Lagrangian (AL) function defined as

$$H(\boldsymbol{x}, \boldsymbol{\gamma}, \boldsymbol{\omega}) = f(\boldsymbol{x}) + \sum_{k=1}^{m} \begin{cases} \gamma_k g_k(\boldsymbol{x}) + \frac{\omega_k}{2} g_k^2(\boldsymbol{x}) & \text{if } g_k(\boldsymbol{x}) \geq \frac{-\gamma_k}{\omega_k} \\ -\frac{\gamma_k^2}{2\omega_k} & \text{otherwise} \end{cases} \tag{2}$$

where $\boldsymbol{\gamma} \in \mathbb{R}^n$ and $\boldsymbol{\omega} \in \mathbb{R}^n$ are respectively the Lagrangian and penalty parameters with $\gamma_k \geq 0$ and $\omega_k > 0$ for all $k = 1, \ldots, m$. They are adapted at each iteration of the ES[1] and we expect $\boldsymbol{\gamma}$ to converge to Lagrange multipliers associated with a (at least) local optimum if the Karush-Kuhn Tucker conditions [15] are satisfied. We call AL-CMA-ES the combination of a $(\mu/\mu_\omega, \lambda)$-CMA-ES optimizing the *dynamic fitness* function H.

[1] Depending on the context, we may or may not use the time index for $\boldsymbol{\gamma}^{(t)}$ and other variables, where t is the iteration counter.

Algorithm 1 AL-$(\mu/\mu_\omega, \lambda)$-CMA-ES

Require: $\boldsymbol{m}^{(0)} \in \mathbb{R}^n$, $\sigma^{(0)} \in \mathbb{R}^+$, default CMA-ES parameters, standard AL parameters $\chi > 1$, $d_\gamma > 0$ and eventually other parameters specific to ω-adaptation

1: $\boldsymbol{\gamma}^{(0)} = \mathbf{0} \in \mathbb{R}^m$
2: $\boldsymbol{\omega}^{(0)} = \mathbf{1} \in \mathbb{R}^m$
3: $t = 0$
4: **while** not happy **do**
5: Ask population $\mathcal{X}^{(t)}$
6: $\mathcal{X}^{(t)} \leftarrow \mathcal{X}^{(t)} \cap \{\boldsymbol{m}^{(t)}\}$
7: Tell with $H(\boldsymbol{x}, \boldsymbol{\gamma}^{(t)}, \boldsymbol{\omega}^{(t)})$ for $\boldsymbol{x}$ in $\mathcal{X}^{(t)}$
8: **for** $k = 1, \ldots, m$ **do** ▷ Loop over constraints
9: **if** $\gamma_k^{(t)} + \omega_k^{(t)} g_k(\boldsymbol{m}^{(t+1)}) > 0$ **then**
10: $\gamma_k^{(t+1)} = \max[0, \gamma_k^{(t)} + \frac{\omega_k^{(t)}}{d_\gamma} g_k(\boldsymbol{m}^{(t+1)})]$
11: **if** condition-to-increase(k) is satisfied **then**
12: increase $\omega_k^{(t)}$ to $\omega_k^{(t+1)}$
13: **if** condition-to-decrease(k) is satisfied **then**
14: decrease $\omega_k^{(t)}$ to $\omega_k^{(t+1)}$
15: $t \leftarrow t + 1$

Algorithm 1 presents the general algorithmic framework for AL-CMA-ES in an ask-and-tell interface. All variants of AL-CMA-ES we consider in this paper obey the framework of Algorithm 1 and differ only by the strategies to adapt the penalty parameters. Lines 11 to 14 provide a generic update scheme of the penalty parameters for each constraint.

The classical update scheme [1–3] is to increase ω_k by a factor $\chi^{1/4}$ if

$$\omega_k g_k^2(\boldsymbol{m}^{(t+1)}) < k_1 \frac{\left|H(\boldsymbol{m}^{(t+1)}, \boldsymbol{\gamma}^{(t)}, \boldsymbol{\omega}^{(t)}) - H(\boldsymbol{m}^{(t)}, \boldsymbol{\gamma}^{(t)}, \boldsymbol{\omega}^{(t)})\right|}{n} \tag{3}$$

or if

$$k_2 \left|g_k(\boldsymbol{m}^{(t+1)}) - g_k(\boldsymbol{m}^{(t)})\right| < \left|g_k(\boldsymbol{m}^{(t)})\right| \tag{4}$$

and otherwise decrease ω_k by a factor χ. We have that $\omega_k^{(t)} \neq \omega_k^{(t+1)}$ for all $k = 1, \ldots, m$ and $t \geq 0$, that is, at each iteration, each penalty parameter is either increased or decreased.

The influence of hyperparameters k_1 and χ on convergence speed and robustness is studied by Dufossé and Hansen [6]. They investigate the algorithm when $k_1 = 10$ and $\chi = 2^{1/\sqrt{n}}$ compared to the previously proposed settings of $k_1 = 3$ and $\chi = 2^{1/5n}$. The latter algorithm is the version 1 of AL-CMA-ES considered in this work. Both settings are used in the different versions considered in this work, but the algorithms also vary by different conditions to trigger the update of $\boldsymbol{\omega}$.

We now introduce some useful quantities to describe versions 2, 3 and 4 of AL-CMA-ES that we benchmark. Consider the archive $\mathcal{A}$ of all points sampled by the evolution strategy and $\mathcal{A}(\rho)$ the last ρ points of $\mathcal{A}$. In practice, the size of $\mathcal{A}$ is bounded by the greatest value of ρ used to query the archive. If $\rho > \lambda$ then the archive accumulates samples from previous iterations.

The version 2 increases ω_k by $\chi^{1/4}$ if either eq. (3) or eq. (4) is satisfied *and* if additionally

$$\begin{cases} \Phi_k(\rho) \geq \tau^+ \\ \quad \text{or} \\ \Phi_k(\rho) \leq 1 - \tau^+ \end{cases} \quad \text{and} \quad g_k(\boldsymbol{m}^{(t+1)}) g_k(\boldsymbol{m}^{(t)}) > 0 \ , \tag{5}$$

where $\Phi_k(\rho)$ is the ratio of points satisfying the k-th constraint in the archive of size ρ

$$\Phi_k(\rho) = \frac{\#\{\boldsymbol{x} \in \mathcal{A}(\rho); g_k(\boldsymbol{x}) \leq 0\}}{\rho} \ , \tag{6}$$

$\tau^+ = 0.6$ defines the default threshold setting and $\rho = \text{int}(1 + \sqrt{n})$, ρ is eventually increased by 1 to be even. Otherwise, the penalty parameter is only decreased by χ if

$$1 - \tau^- < \Phi_k(\rho) < \tau^- \ , \tag{7}$$

with $\tau^- = 0.95$ and $\rho = n + 20$.

The versions 3 and 4 don't consider anymore the conditions of eqs. (3) and (4) to update the penalty parameter. They also share the same setting for $\rho = 5 + n$ and the same factors to update ω_k.

This multiplicative factor is not constant anymore over iterations. Let s_k be 1 if condition-to-increase(k) is true and -1 if condition-to-decrease(k) is true and $d_k^{(t)} = \sum_{t'=t_0}^{t} s_k^{(t')}$ where t_0 is the last time s_k has changed, forgetting iterations where no condition is triggered. Then d_k starts at 1 (respectively -1) and increases (resp. decreases) each time the same condition is satisfied and is reset when the other condition is satisfied. The increase of ω_k follows

$$\omega_k^{(t+1)} = \omega_k^{(t)} \chi^{(1+C)/4} \tag{8}$$

and the decrease

$$\omega_k^{(t+1)} = \omega_k^{(t)} \chi^{-(1-C)/4} \tag{9}$$

where $C = 4 s_k \min\left[T, \max[s_k d_k - T, 0]\right]/T$ with $T = 2 + n$. Hence "The value [of C] is zero for the first [T] same changes and increases linearly for the next [T] same changes and then stays at the threshold value as long as the change does not flip sign."[2]

For version 3, the penalty ω_k is increased if either $g_k(\boldsymbol{m}^{(t+1)}) > 0$ or $\Phi_k(\rho) > \tau^+ = 0.95$ and decreased only if $\Phi_k(\rho) < \tau^- = 0.9$ and $g_k(\boldsymbol{m}^{(t+1)}) < 0$.

For version 4, it is increased either if $g_k(\boldsymbol{x}) > 0$ for all $\boldsymbol{x} \in \mathcal{A}(2n)$ (i.e. $\Phi_k(2n) = 0$) or if $g_k(\boldsymbol{m}^{(t+1)}) > -\gamma_k^{(t)}/\omega_k^{(t)}$ and $\Phi_k(\rho) > \tau^+ = 0.95$. It is decreased if $g_k(\boldsymbol{m}^{(t+1)}) < -\gamma_k^{(t)}/\omega_k^{(t)}$ and $\Phi_k(2n) < 1$.

Table 1 provides a summary of all hyperparameters implied in the different strategies to adapt the penalty parameters throughout the optimization process.

3 EXPERIMENTAL PROCEDURE

We run AL-CMA-ES from the pycma package[3] [8] using the AL fitness implemented in the constraints_handler module. All hyperparameters for CMA-ES are set to the default values provided in the used package version. In this section, we provide more details about the experimental setup related to constraint handling and baseline algorithms COBYLA and random search. Each AL version

[2] See the documentation of CountLastSameChanges in the constraints_handler module of the pycma package.
[3] https://github.com/CMA-ES/pycma/releases/tag/r3.2.2

Table 1: Hyperparameters used in the different versions of AL-CMA-ES implemented in pycma. If a parameter is not used in a version, it is replaced with a cross (×) and if it has different values for increase and decrease conditions, they are specified with respectively up (↗) and down (↘) arrows.

Version	Hyperparameters						
	Existing				Newly introduced		
	χ	d_γ	k_1	k_2	ρ	τ^+	τ^-
1	$2^{1/5n}$	5	3	5	×	×	×
2	$2^{1/\sqrt{n}}$	5	10	5	$\simeq 1+\sqrt{n}$(↗) $n+20$(↘)	0.6	0.95
3 and 4	$2^{1/5n}$	5	×	×	$5+n$	0.95	0.9

is given a budget multiplier of 5×10^4 meaning the sum of objective and constraint evaluations must not exceed $5 \times 10^4 n$ where n is the problem dimension.

Initialization heuristic. The Lagrangian and penalty parameters are set based on f- and g- values of sampled points in the first population, $\{(f(\boldsymbol{x}), g(\boldsymbol{x})),\ \boldsymbol{x} \in \mathcal{X}^{(0)}\}$. Compute Δf and $\Delta g_k, \Delta g_k^2$ for $k = 1, \ldots, m$ where Δf is the interquartile range of $\{f(\boldsymbol{x}), \boldsymbol{x} \in \mathcal{X}^{(0)}\}$, and set

$$\gamma_k^{(0)} = \frac{\Delta f}{n\Delta g_k + 10^{-11}(\Delta f + 1)} \tag{10}$$

$$\omega_k^{(0)} = \frac{2\Delta f}{5n\left(\Delta g_k + 10^{-6}\Delta g_k^2 + 10^{-11}(\Delta f + 1)\right)} \tag{11}$$

where in both cases the term $10^{-11}(\Delta f + 1)$ is to avoid division by 0.

Restarts with increasing population size. The stopping criteria are numerous and kept to default from pycma. Only tolstagnation is disabled because it is pointless in the case of a dynamic fitness function. If any criterion is met before the allowed budget is exhausted, the algorithm restarts according to the Increasing Population size framework (IPOP-CMA-ES) [4]. The parameters of the AL fitness function are not reset.

Baseline data. In addition to AL-CMA-ES variants, we include other algorithms for baseline comparison. The random search rs uniformly samples points in $-[5, 5]^n$ and evaluates both the objective and the constraints at each point. This dataset can be found in the official COCO data archive and the budget multiplier is 2×10^5. We also run fmin_cobyla from the SciPy [18] optimization library which is a wrapper around Powell's Fortran code [16]. The budget multiplier is 10^4 and we set the initial trust-region radius rhobeg to 1, the constraint tolerance catol to 10^{-8} and the minimum radius rhoend (stopping criterion) to 10^{-10} .

4 TIMING

We monitored the elapsed time of all algorithms while running on the entire bbob-constrained test suite [5] for the given budgets according to Hansen et al. [12]. We did not monitor the CPU time. There may be a discrepancy between the two for AL-CMA-ES, as the pycma package relies on NumPy [13] routines for linear algebra spanning multiple cores. For COBYLA, the core code is written in Fortran so the elapsed and CPU times also depend on the compiler.

Table 2: Timing results for AL-CMA-ES (AL), COBYLA and random search in milliseconds.

Dimension	2	3	5	10	20	40
AL	0.35	0.29	0.33	0.60	0.99	3
COBYLA	0.022	0.016	0.023	0.055	0.14	0.59
RS	0.034	0.039	0.058	0.11	0.22	0.51

The code was run on a Linux machine with 64 Intel(R) Xeon(R) CPUs (E5-2683 v4 @ 2.10GHz), 64 cores and 64 GB RAM. The time per function evaluation is monitored as in example_experiment2.py for dimensions 2, 3, 5, 10, 20 and 40 and reported in Table 2. Because we split the optimization processes between different instances into batches, there is one time measurement per batch and AL version. We display in Table 2 the median value obtained across the different batches and versions. The relative sample differences[4] between the maximum and minimum elapsed times are respectively of 34, 18, 21, 12, 8 and 24 percent for dimensions 2, 3, 5, 10, 20, 40 and show no significant bias towards a specific AL version. Indeed the computational effort to adapt the AL fitness is small compared to CMA-ES.

5 RESULTS AND DISCUSSION

Results from conducted experiments following the COCO performance assessment methodology [5, 9, 12] on the bbob-constrained benchmark functions [5] are presented in Figures 1 to 5. The experiments were performed with COCO [11], version 2.6.2, the plots were produced with version 2.6.1.8.

The **expected running time (ERT)**, reported in Figures 1 and 2, depends on a given target value, $\tau_t = f_{\text{opt}} + \Delta\tau$, and is computed over all relevant trials as the sum of function and constraint evaluations executed during each trial while the best value of the merit function [5] has not reached τ_t, summed over all trials and divided by the number of trials that actually reached τ_t [5, 10, 17]. **Statistical significance** is tested with the rank-sum test for a given target $\Delta\tau_t$ using, for each trial, either the number of needed function and constraints evaluations to reach $\Delta\tau_t$ (inverted and multiplied by −1), or, if the target was not reached, the best $\Delta\tau$-value achieved, measured only up to the smallest number of overall (function + constraints) evaluations for any unsuccessful trial under consideration.

Unsurprisingly, random search does not perform very well on almost all problems but on the linear slope problem f13 in 20-D (Figure 3) and, to a lesser extent, on the Discus problem f25. In fact, while we do not enter the flat region of the linear slope objective function, the solution set of f13 is a hyperplane, because there is only 1 constraint which boundary is also the level set of the underlying linear function (see Dufossé et al. [5] for a mathematical characterization of the solution set). Such argument can be roughly extended to the Discus problem, whose level sets are similar to a linear function on a global scale. Beyond these cases, which are

[4] Absolute value of the difference divided by the maximum value across all measurements in the sample.

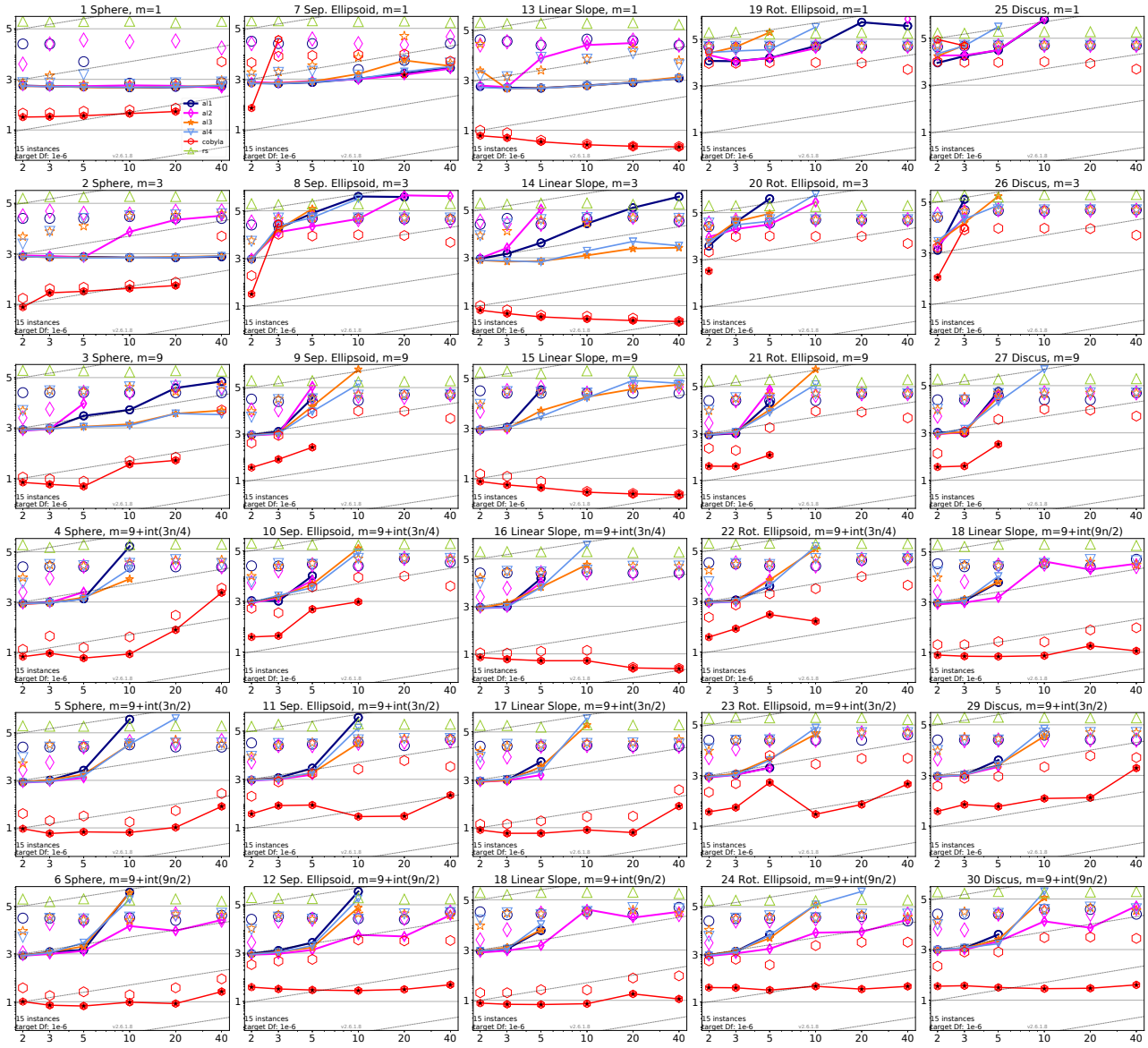

Figure 1: Expected running time (ERT in number of evaluations as $\log_{10}$ value), divided by dimension for target function value 10^{-6} versus dimension. Slanted grid lines indicate quadratic scaling with the dimension. Different symbols correspond to different algorithms given in the legend of f_1. Light symbols give the maximum number of evaluations from the longest trial divided by dimension. Black stars indicate a statistically better result compared to all other algorithms with $p < 0.01$ and Bonferroni correction number of dimensions (six). Legend: ○: al1, ◇: al2, ⋆: al3, ▽: al4, ⬡: cobyla, △: rs.

hard to avoid considering the merit function used, random search serves as a sanity check for any reasonable algorithm.

On the sphere problem with 1 constraint f1, Figure 1 reveals that the number of evaluations required to reach the target with a precision value of 10^{-6} scales linearly with dimension. COBYLA is about 10 times faster than AL in all dimensions but in 40-D where it does not reach the hardest targets. Considering ECDFs in Figure 3, we see that all AL versions solve all targets with near-equal performance. On sphere problems with more than one constraint, we see performance plateaus for AL algorithms. In particular, on f2 in 20-D, al2 solves 90% of the targets within 2×10^4 evaluations but still needs up to 10^6 evaluations to reach the remaining last 10%. It solves only 80% of the targets on f3, 70% on f4 and f5. This effect is reverted on f6 as al2 solves (almost) all targets whereas other versions stagnate around 80% of the targets. The same effect is observed on some other problems in 20-D, like the linear slope f13 to f18 and bent cigar f31 to f37. In the latter case, for f33, f34, f35, the al3 and al4 versions seem less affected and solve all targets. A

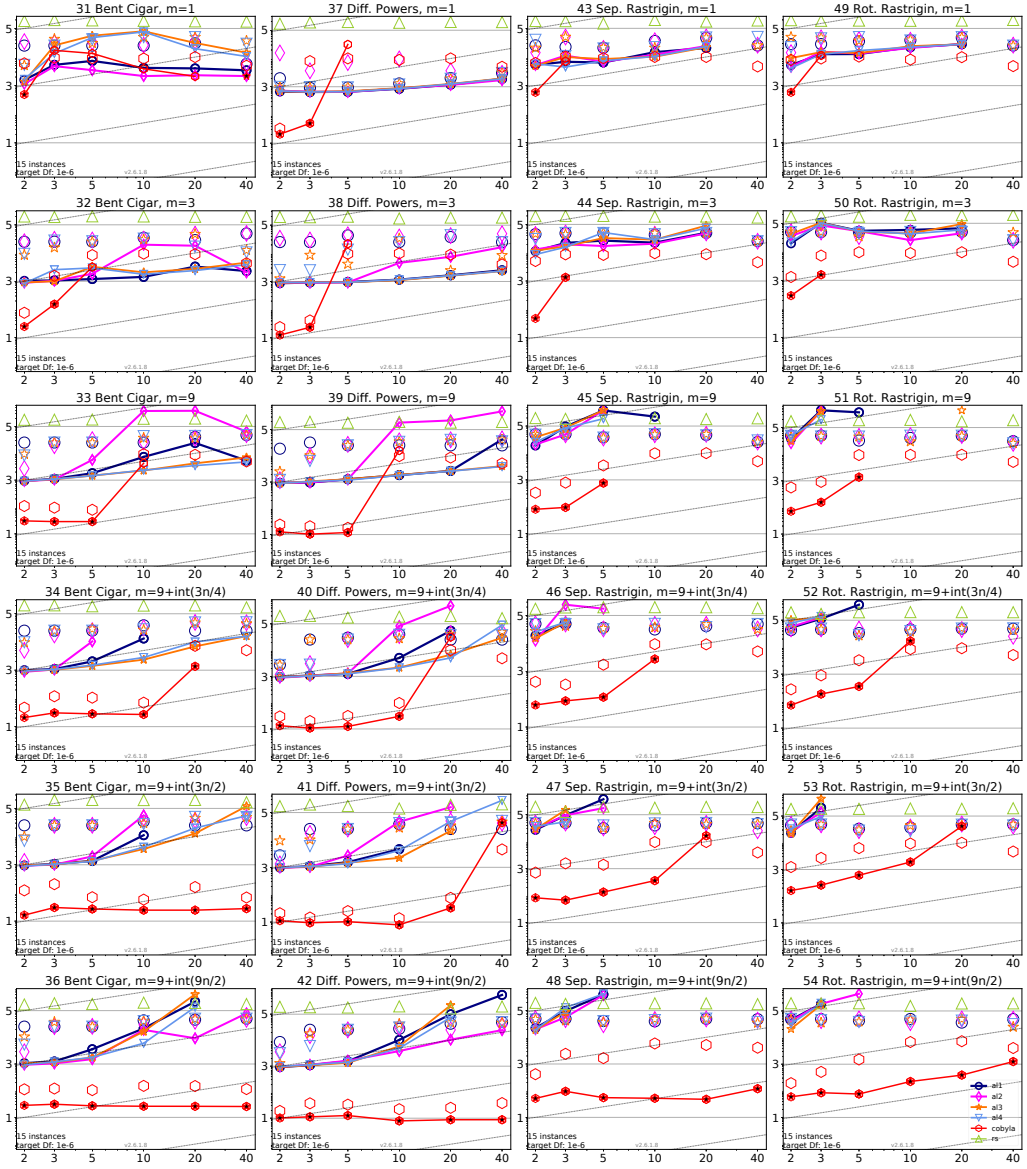

Figure 2: Expected running time (ERT in number of evaluations as $\log_{10}$ value), divided by dimension for target function value 10^{-6} versus dimension. Slanted grid lines indicate quadratic scaling with the dimension. Different symbols correspond to different algorithms given in the legend of f_{54}. Light symbols give the maximum number of evaluations from the longest trial divided by dimension. Black stars indicate a statistically better result compared to all other algorithms with $p < 0.01$ and Bonferroni correction number of dimensions (six). Legend: ○: al1, ◇: al2, ⋆: al3, ▽: al4, ⬡: cobyla, △: rs.

possible reason for these plateaus is the triggering of some stopping criterion in CMA-ES before the last target is reached. It is also clear that the effect varies with the AL version considered since the stopping criteria are set to the same values.

Investigating early stopping of al1 on f18 in 20-D ($m = 99$) several times, we assert the following scenario: the ES without restart stops after $\sim 4 \times 10^4$ because the tolfun = 10^{-11} stopping criterion is triggered. The euclidean distance from the optimal solution to the ES mean is $\sim 10^{-8}$ and indicates convergence. While the merit function value is very close to the final target, some constraints are slightly violated, hence don't satisfy the feasibility condition to trigger the target. Note that the fmin_con method of the pycma package also implements a repair method consisting of minimizing the sum of the constraints' violations squared via the post_optimization boolean argument. The repair method runs after CMA-ES has optimized the AL fitness, starting from the best obtained solution with the adapted step-size and covariance matrix. This second stage is typically fast and we can expect the new solution to improve the

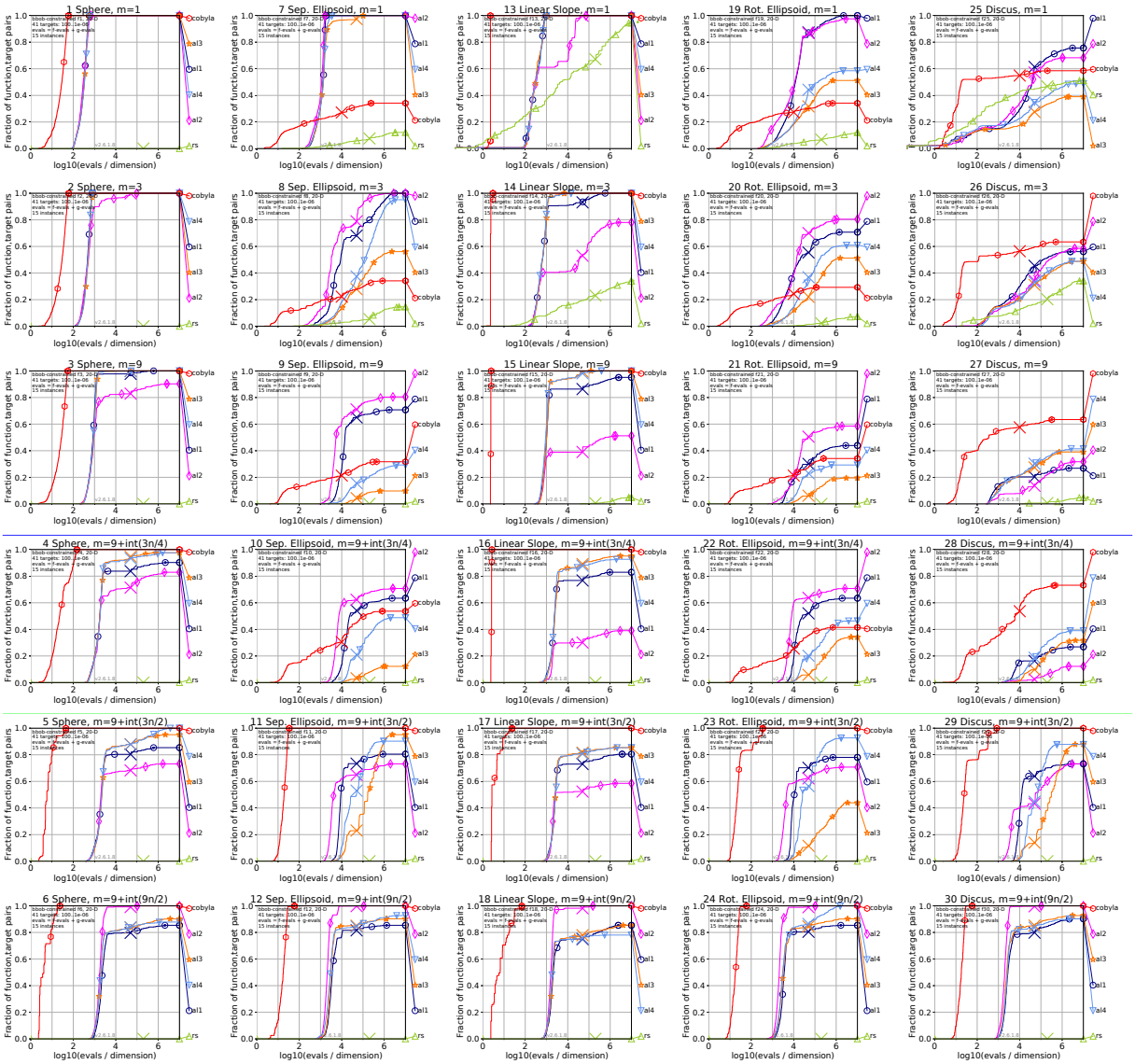

Figure 3: Empirical cumulative distribution of simulated (bootstrapped) runtimes, measured in number of evaluations, divided by dimension (evals/DIM) for the 41 targets $10^{[-6..2]}$ in dimension 20 and problem IDs 1 to 30. Problems below the blue line have more constraints than dimension, and problems below the green line have more active constraints than dimension.

merit function. In these experiments, we disabled the repair method and it is of further interest to quantify its influence on the number of targets solved, all other parameters being equal.

On the same problem, we also observe the Lagrangian parameters to converge to values spanning multiple orders of magnitude, between 2 and 10^{-3}, whereas in the merit function[5] used to trigger targets, coefficients playing a similar role are all set to 1 [5]. This discrepancy between the Lagrangian function and the merit function reveals what can be seen as both a feature of the test suite and a defect of algorithms based on Lagrange multipliers where the scaling of the constraints defines the precision at which they are satisfied. Decreasing funtol to 10^{-12}, AL-CMA-ES does not converge and it is most likely that the algorithm is affected by numerical noise.

As expected, COBYLA solves the linear slope problems f13 to f18 with very few evaluations (less than $10n$ in almost all cases and all dimensions, see Figure 1). Indeed, these problems have no non-linear transformation hence the linear surrogate model of COBYLA is exact.

On ellipsoid problems, we observe that all AL versions solve the separable ellipsoid with 1 constraint f7 within roughly $10^3 n$

[5]We recall the merit function is $\tilde{f}(\boldsymbol{x}) = \max[f(\boldsymbol{x}), f^\star] + \sum_k \max[g_k(\boldsymbol{x}), 0]$ where $f^\star$ is the optimal value and $k = 1, \ldots, m$.

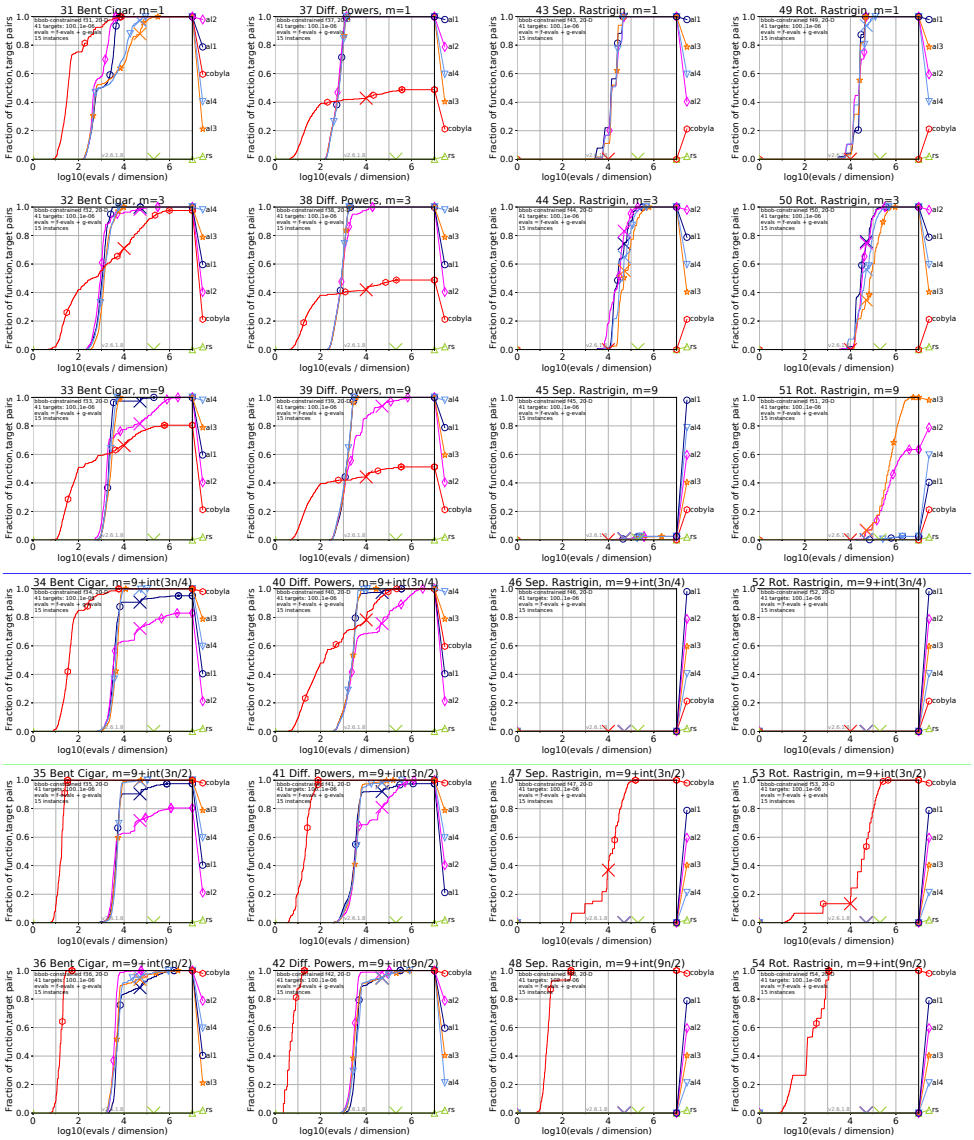

Figure 4: Empirical cumulative distribution of simulated (bootstrapped) runtimes, measured in number of evaluations, divided by dimension (evals/DIM) for the 41 targets $10^{[-6..2]}$ in dimension 20 and problem IDs 31 to 54.

evaluations for all dimensions n, though this slightly increases, from $9 \times 10^2 n$ in 5-D to $2 \times 10^3 n$ in 40-D (see Figure 1). Comparing the separable with the rotated ellipsoid ECDFs in Figure 3, al1 and al2 require 10 times more evaluations to solve the latter problem with 1 constraint, and al3, al4 are much slower (again roughly another factor of 10 before the budget is exhausted). This ranking of AL versions generalize to ellipsoid problems with more constraints except when $m = 9 + \text{int}(9n/2)$ where all versions show equal performance before al1, al3 and al4 reach the plateaus. The ECDFs of COBYLA are the same for the separable and rotated ellipsoid problems, reaching 30% of the targets within $10^4 n$ evaluations when $n = 20$.

On the global picture, aggregating data for problems with the same number of constraints, Figure 5 reveals that in all cases COBYLA is 10 to 100 times faster than AL-CMA-ES to solve the low tier of targets, which coincides with small budgets, say below $10^3 n$. On problems where there remain degrees of freedom, i.e. the number of active constraints is less than the dimension, AL-CMA-ES is competitive for large budgets, and in particular for challenging objective functions where the linear surrogate model of COBYLA is a poor fit, like ellipsoid and different powers problems with $m \in \{1, 3, 9\}$. Where there is no degree of freedom, COBYLA efficiently solves all problems except the Rastrigin function f47 and f53 with $m = 9 + \text{int}(3n/2)$ constraints.

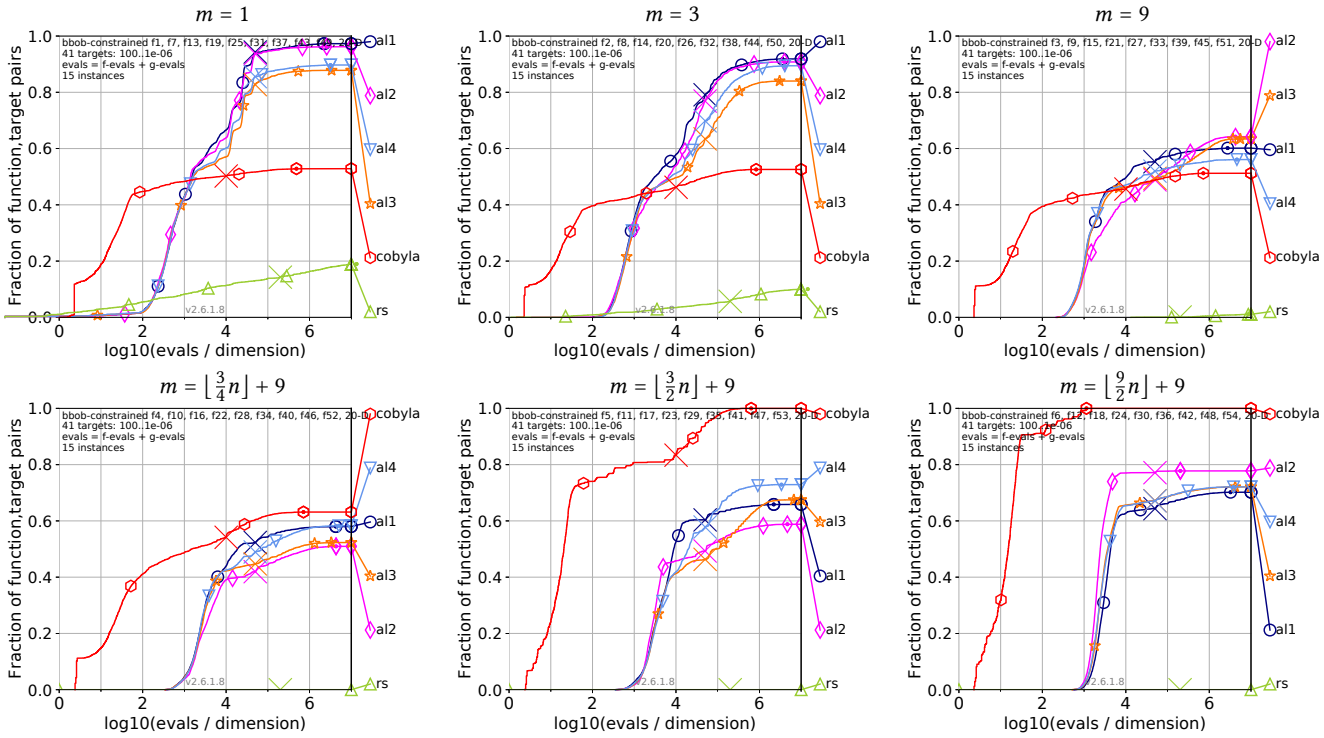

Figure 5: Bootstrapped empirical cumulative distribution of the number of evaluations divided by dimension (evals/DIM) for 41 targets with target precision in $10^{[-6..2]}$ for all functions and subgroups in 20-D. The functions are grouped by number of constraints.

Finally, Figure 5 shows the two algorithms are rather complementary, and suggests that running a portfolio algorithm composed of COBYLA and AL-CMA-ES[6] would be a good strategy in a black-box scenario.

[6]Running the former first as it is faster and in particular if the problem has many constraints.

REFERENCES

[1] Dirk V. Arnold and Jeremy Porter. 2015. Towards an Augmented Lagrangian Constraint Handling Approach for the $(1+1)$-ES. In *Genetic and Evolutionary Computation Conference*. ACM Press, 249–256.

[2] Asma Atamna, Anne Auger, and Nikolaus Hansen. 2016. Augmented Lagrangian Constraint Handling for CMA-ES—Case of a Single Linear Constraint. In *Proceedings of the 14th International Conference on Parallel Problem Solving from Nature*. Edinburgh, United Kingdom, 181 – 191. https://doi.org/10.1007/978-3-319-45823-6_17

[3] Asma Atamna, Anne Auger, and Nikolaus Hansen. 2017. Linearly Convergent Evolution Strategies via Augmented Lagrangian Constraint Handling. In *The 14th ACM/SIGEVO Workshop on Foundations of Genetic Algorithms (FOGA XIV)*. Copenhagen, Denmark, 149 – 161. https://doi.org/10.1145/3040718.3040732

[4] Anne Auger and Nikolaus Hansen. 2005. A restart CMA evolution strategy with increasing population size. In *2005 IEEE Congress on Evolutionary Computation*, Vol. 2. 1769–1776. https://doi.org/10.1109/CEC.2005.1554902

[5] Paul Dufossé, Asma Atamna, Philippe R Sampaio, Dimo Brockhoff, Nikolaus Hansen, and Anne Auger. 2022. Building scalable test problems for benchmarking constrained optimizers. *?* ? (2022), ?

[6] Paul Dufossé and Nikolaus Hansen. 2021. Augmented Lagrangian, Penalty Techniques and Surrogate Modeling for Constrained Optimization with CMA-ES. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '21)*. Association for Computing Machinery, New York, NY, USA, 519–527. https://doi.org/10.1145/3449639.3459340

[7] Nikolaus Hansen. 2006. The CMA Evolution Strategy: A Comparing Review. In *Towards a New Evolutionary Computation: Advances in the Estimation of Distribution Algorithms*, Jose A. Lozano, Pedro Larrañaga, Iñaki Inza, and Endika Bengoetxea (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 75–102. https://doi.org/10.1007/3-540-32494-1_4

[8] Nikolaus Hansen, Youhei Akimoto, and Petr Baudis. 2019. CMA-ES/pycma on Github. Zenodo, DOI:10.5281/zenodo.2559634. https://doi.org/10.5281/zenodo.2559634

[9] Nikolaus Hansen, Anne Auger, Dimo Brockhoff, Dejan Tušar, and Tea Tušar. 2016. COCO: Performance Assessment. *ArXiv e-prints* arXiv:1605.03560 (2016).

[10] Nikolaus Hansen, Anne Auger, Steffen Finck, and Raymond Ros. 2012. *Real-Parameter Black-Box Optimization Benchmarking 2012: Experimental Setup*. Technical Report. INRIA. http://numbbo.github.io/gforge/bbob2012-downloads

[11] Nikolaus Hansen, Anne Auger, Olaf Mersmann, Tea Tušar, and Dimo Brockhoff. 2016. COCO: A Platform for Comparing Continuous Optimizers in a Black-Box Setting. *ArXiv e-prints* arXiv:1603.08785 (2016).

[12] Nikolaus Hansen, Tea Tušar, Olaf Mersmann, Anne Auger, and Dimo Brockhoff. 2016. COCO: The Experimental Procedure. *ArXiv e-prints* arXiv:1603.08776 (2016).

[13] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. 2020. Array programming with NumPy. *Nature* 585, 7825 (Sept. 2020), 357–362. https://doi.org/10.1038/s41586-020-2649-2

[14] Magnus R. Hestenes. 1969. Multiplier and Gradient Methods. *Journal of Optimization Theory and Applications* 4, 5 (1969), 303–320. https://doi.org/10.1007/BF00927673

[15] Harold W. Kuhn and Albert W. Tucker. 1951. Nonlinear programming. In *Proceedings of the Second Berkeley Symposium on Mathematical Statistics and Probability*. University of California Press, Berkeley, USA, 481–492.

[16] Michael J. D. Powell. 1969. A Method for Nonlinear Constraints in Minimization Problems. In *Optimization*, R. Fletcher (Ed.). Academic Press, 283–298.

[17] Kenneth Price. 1997. Differential evolution vs. the functions of the second ICEO. In *Proceedings of the IEEE International Congress on Evolutionary Computation*. IEEE, Piscataway, NJ, USA, 153–157. https://doi.org/10.1109/ICEC.1997.592287

[18] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. 2020. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods* 17 (2020), 261–272. https://doi.org/10.1038/s41592-019-0686-2