Evolving the Structure of Evolution Strategies

Sander van Rijn, Hao Wang, Matthijs van Leeuwen, Thomas Bäck
Natural Computing Group
LIACS, Leiden University
Niels Bohrweg 1, 2333 CA
Leiden, The Netherlands
Email: {s.j.van.rijn, h.wang, m.van.leeuwen, t.h.w.baeck}@liacs.leidenuniv.nl

***Abstract*—Various variants of the well known Covariance Matrix Adaptation Evolution Strategy (CMA-ES) have been proposed recently, which improve the empirical performance of the original algorithm by structural modifications. However, in practice it is often unclear which variation is best suited to the specific optimization problem at hand. As one approach to tackle this issue, algorithmic mechanisms attached to CMA-ES variants are considered and extracted as functional *modules*, allowing for combinations of them. This leads to a configuration space over ES structures, which enables the exploration of algorithm structures and paves the way toward novel algorithm generation. Specifically, eleven modules are incorporated in this framework with two or three alternative configurations for each module, resulting in $4\,608$ algorithms. A self-adaptive Genetic Algorithm (GA) is used to efficiently evolve effective ES-structures for given classes of optimization problems, outperforming any classical CMA-ES variants from literature. The proposed approach is evaluated on noiseless functions from BBOB suite. Furthermore, such an observation is again confirmed on different function groups and dimensionality, indicating the feasibility of ES configuration on real-world problem classes.**

I. Introduction

Evolutionary Algorithms (EAs) such as Genetic Algorithms (GAs) [14] and Evolution Strategies (ESs) [26] have been studied for decades, leading to the many variants proposed in the literature [5]. The performance of all these algorithms depends not only on the specific optimization task, but also on appropriately tuning the algorithm's parameters such as population size and mutation rate. As a result, the ideas of online tuning [9] and automated parameter optimization have been proposed [4]. These were later reinforced by the realization that an increase in performance for some problem instances will necessarily cause a decrease in performance for some other instances. This implies that the potential performance improvement that can be gained from optimizing parameters is always limited by the algorithm that is chosen. To limit the bias imposed by the choice of a specific (evolutionary) algorithm, researchers have proposed to instead *evolve the structure of an EA itself* (e.g., [17], [16], [27]).

The class of optimizers derived from the Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [13] are the preferred optimization method for many real-valued black-box problems, and are therefore also the scope of this paper. As discussed in more detail shortly, many variations of the CMA-ES have been separately introduced and discussed in the literature. However, only few combinations of these variations have been empirically tested and compared. As a result, in practice it is often unclear which variation is best suited to the optimization task at hand. This leads to the three main questions in this work:

1) Can we define a modular and extensible CMA-ES framework that allows to unify the many variations that have been introduced in the literature?
2) Within the framework defining a large number of CMA-ES variations, how to determine an efficient ES structure, given limited function evaluation budget?
3) Are there any novel variations, i.e., combinations of methods that have been proposed in the literature, that outperform the known variants?

Approach and contributions Firstly, a modular and extensible CMA-ES framework is proposed based on the original CMA-ES [13], for adapting the structure of Evolution Strategies in particular. A number of independent, functional *modules* are extracted from various existing ES variants. By allowing each of these modules to be activated independently, a set of $4\,608$ so-called *ES-structures* can be instantiated, many of which have never been considered before. A detailed overview of the framework and all selected modules is given in Section III.

Secondly, a metaheuristic is proposed to search in the configuration space containing ES-structures. In Section IV a simple yet effective genetic encoding scheme is used, facilitating a mutation-only and self-adaptive GA optimization. Although it is possible to deploy an exhaustive search (brutal force) on such a combinatorial configuration space, the time complexity of exhaustive search would grow polynomially with increasing number of ES modules and makes it computationally infeasible in practice. Additionally, Section IV-A investigates how we can robustly evaluate evolution strategies. The proposed approach combines a stable fitness measure—based on *Fixed Cost Error* (FCE) and *Estimated Running Time* (ERT)—with a statistical estimation of the number of runs required to reliably compare different strategies.

Finally, the third contribution consists of an extensive empirical evaluation to investigate the potential of the CMA-ES configuration framework and metaheuristic used for the configuration. The experiments described in Section V, using the noiseless optimization functions from BBOB [12], show that the meta-GA consistently converges quickly and produces results that are on par with the best possible configurations found by brute force search, needing only 5% of the evaluations. Furthermore, the effectiveness of various modules is shown on each function group and dimensionality, indicating that searching for a suitable ES structure is a better solution than always resorting to a single 'default' configuration.

978-1-4799-7492-4/15/$31.00

II. Related Work

The idea of evolving the structure of Evolutionary Algorithms (EAs) is first motivated in [18], using the Genetic Programming (GP) technique. Multiple attempts have been made for this purpose, e.g. meta-evolution on graph-based programs [17], [16]. In those studies, varying numbers of meta-levels are used, each responsible for evolving the program used in the level below and potentially itself. The *Push* language [27] was later dedicated to enable autoconstructive evolution whereby a population of programs is expected to produce other programs.

Later approaches focused more on applying evolution within the existing structure of an EA. In [28] GP is used to create new mutation, recombination and selection operators, from which a standard EA structure is constructed. Oltean *et al.* use existing operators as the genes to create new structures in [24], [22], [23], while tree-based structures of multiple EAs with different parameters are evolved in [21] by Martin *et al*.

The Grammatical Evolution (GE) approach by Lourenço *et al.* [20] is highly related to this work. The structure of an EA is represented as a context free grammar with parameters and operators as terminals. Treating an EA as a collection of atomic operators in a fixed framework allows arbitrary combinations, similar to our proposed approach.

III. CMA-ES Framework

This section introduces the modular and extensible CMA-ES framework. Short summaries of all considered modules are first given in Section III-A, after which Section III-B introduces the modular framework. Implementation details are described in Section III-C.

A. ES Variations

Eleven possible modules are considered in total, nine of which have two available options, and the remaining two have three options. This results in $2^9 \cdot 3^2 = \mathbf{4\,608}$ different ES-structures. For each module, a brief description is given below.

1) **Active Update:** The update of covariance matrix C is normally only done by taking the most successful mutations into account. The Active Update, introduced by Jastrebski *et al.* [15], adapts the covariance matrix using the negative factor based on the least successful individuals, in addition to the standard covariance matrix update.
2) **Elitism:** Both (μ, λ)- and $(\mu+\lambda)$-strategy are proposed together in evolutionary algorithms. In this work, elitism is considered as an alternative.
3) **Mirrored Sampling:** A technique to ensure more evenly spaced sampling of the search space is Mirrored Sampling by Brockhoff *et al.* [7]. Half of the mutation vectors are still sampled from the normal distribution, but every other mutation vector is the mirror image of the previous random vector.
4) **Orthogonal Sampling:** A later addition to Mirrored Sampling was Orthogonal Sampling by Wang *et al.* [29]. The desired number of samples is first drawn from the normal distribution. The Gram-Schmidt process is then used to orthonormalize the set of vectors.
5) **Sequential Selection:** Without parallel execution, all λ individuals are first evaluated in order, and then selection is applied. The sequential selection method proposed by Brockhoff *et al.* [7] immediately compares the function value of each newly evaluated individual to the best found so far, and does not evaluate any additional offspring individuals when an improvement has been found.
6) **Threshold Convergence:** Becoming stuck in a local optimum is a common problem when using an ES. Piad *et al.* propose Threshold Convergence [25] as a method of forcing the evolution to stay in an exploratory phase for longer, by requiring mutation vectors to reach a length threshold. This threshold then decreases after every generation to slowly transition into local search.
7) **Two-Point Step-Size Adaptation (TPA):** The step size σ of the CMA-ES is adapted after every generation according to the evolution path, which incorporates the latest successful individuals. Hansen *et al.* proposed TPA [10], which reserves two individuals from the λ offspring. These are used to evaluate two mutations after selection and recombination has taken place: one with a longer, the other with a shorter version of the weighted average mutation vector belonging to the μ selected individuals. Which of these two results in a lesser function value, determines whether the step-size should increase or decrease.
8) **Pairwise Selection:** Mirrored Sampling can cause a bias in the length of mutation vectors, as two mirrored vectors will (partially) cancel each other out in recombination. Pairwise Selection was introduced by Auger *et al.* [1] to prevent this. In this paper, the best offspring is first selected from each mirrored pair. The regular selection operator is then applied to all offspring that were selected in this previous step.
9) **Recombination Weights:** In the CMA-ES, recombination is performed with the following weight vector:

$$w_i = \log(\mu + \frac{1}{2}) - \frac{\log(i)}{\sum_j w_j}$$

for $i = 1, \ldots, \mu$. Alternative weights are the arithmetic mean $w_i = \frac{1}{\mu}$.
10) **Quasi-Gaussian Sampling:** Samples are not necessarily uniformly drawn from the normal distribution. Alternatively, the vectors can be drawn from a quasi-random uniform sequence, which are then transformed to a Gaussian distribution, as proposed in [3]. As quasi-random sequences, the *Sobol* and *Halton* sequences can be used.
11) **Increasing Population Size (IPOP):** Restarting an ES can be done when no improvements have been found in recent generations. Auger *et al.* proposed an increasing population scheme IPOP [2] to use the remaining function evaluations more effectively after a restart. Later, Hansen *et al.* introduced the bi-population (BIPOP) [11] variation which alternates between a larger and smaller population size.

Note that although pairwise selection and orthogonal sampling were introduced in combination with mirrored sampling, each can be activated independently in this framework.

B. ES Framework

To easily allow the combination of all ES-variations listed in Section III-A, we introduce a modular framework based on the CMA-ES (see Algorithm 1). It is designed such that a module can be activated by replacing a function or passing an additional variable. Any endogenous variables of the CMA-ES and its variations are abstracted into a single global *params* object (lines 9, 14) that is accessible from all other functions.

Default values are available in literature for all relevant parameters, whether belonging to the standard CMA-ES or to any particular variant. These defaults are only used when no other values are specified at initialization (line 6).

Algorithm 1 Modular CMA-ES Framework

```
1:  options ← which modules are active
2:  // Local restart loop
3:  while not terminate do
4:      t ← 0
5:      x̄ ← randomly generated individual
6:      SetParameters(init-params)
7:      // ES execution loop
8:      while not terminate local do
9:          params ← Initialize(init-params)
10:         x⃗ ← Mutate(x̄, options)            // Sampler, Threshold
11:         f⃗ ← Evaluate(x⃗, options)           // Sequential
12:         P^(t+1) ← Select(x⃗, f⃗, options)    // Elitism, Pairwise
13:         x̄ ← Recombine(P^(t+1), options)  // Weights
14:         UpdateParams(params, options)    // Active, TPA
15:         t ← t + 1
16:     end while
17:     AdaptParams(init-params)             // (B)IPOP
18: end while
```

The variable functions are the mutation (line 10), selection (line 12), recombination (lines 13) and parameter update (line 14). Here, the variability is shown by the added *options* argument in all function calls. The sampler is a special case that merges three variations: Quasi-Gaussian sampling replaces the regular Gaussian sampling that is used as *base-sampler*. Mirrored sampling and Orthogonal sampling are added on top of the base-sampler instead.

Sequential selection (line 11) is performed in the abstracted *Evaluate* function that otherwise acts as a simple call-through to the evaluation function. The local restart criteria checked in line 8 are all those from [2], [11], and population size adaptations for (B)IPOP are performed in line 17.

Each module can be activated independently of all others. Furthermore, new variations on existing modules can easily be created, and new modules can be added without rewriting the entire algorithm.

C. Implementation Details

Our aim is to allow any possible combination of the modules listed in Section III-A such that the resulting ES will run with minimal need of checking dependencies between variants, and without causing runtime errors. Two of the considered modules have to be adapted before they can be used in this way, because they were proposed for different evolution strategies.

- **Sequential Selection:** Originally intended to be used in $(1, \lambda)$-strategies, a delay in the cut-off is introduced to ensure evaluation of at least μ individuals when $\mu > 1$. This represents a more robust solution than accepting $< \mu$ individuals and adapting all following calculations.
- **Threshold Convergence:** In the original paper by Piad *et al.* [25], Threshold Convergence is used in a regular (μ, λ)-ES. The threshold is applied to the mutation vector after it has been scaled by the step size σ. If this method is equally applied in our framework, the benefit of the CMA-ES is lost for small mutations, because the covariance matrix C scales mutations in different directions differently. Instead, the threshold is applied to the randomly sampled vector, before it is used in any further calculations. This forces the mutation vector to have a minimal length, without losing benefits of the covariance matrix C.

Furthermore, selection modules can cause problems when they require more than μ individuals for the selection process. Descriptions of the encountered issues and our solutions are given below.

- **Pairwise Selection and Sequential Selection:** For pairwise selection to return μ individuals, a selection must be made from at least 2μ individuals. This causes a problem when sequential selection is allowed to stop the generation after μ individuals. To solve this, the cut-off point for sequential selection is artificially increased to 2μ. If $\lambda < 2\mu$, λ is also increased to 2μ.
- **Pairwise Selection and TPA:** TPA reserves two individuals from the λ offspring, preventing them from being used for selection and recombination. This leaves the ES with $\lambda_{\text{eff}} = \lambda - 2$ individuals. When pairwise selection is used and $\lambda = 2\mu$, we are one pair short of being able to select μ individuals. In this case, μ is set to $\lambda_{\text{eff}}/2$.
- **Pairwise Selection, Sequential Selection and TPA:** When pairwise selection, sequential selection and TPA are all active, the cut-off point for sequential selection is based on λ_{eff}.

IV. Evolving ES Structures

As mentioned in Section III-A, the modular framework (Algorithm 1) can be used to instantiate many different ES-structures. There is no interest in the performance of every individual combination, both because there are far too many and because most will perform poorly. Instead, it is far more interesting to determine the best performing ES-structure for different *classes* of optimization problems.

For a GA tasked with this optimization problem, a valid way of comparing ESs is required. The considerations to be taken into account in establishing a robust comparison are examined in Section IV-A. Next, Section IV-B explains the encoding that is used during the search process. Details of the GA used can be found in Section IV-C.

A. Evaluating Evolution Strategies

Let f be some (black-box) function to be minimized, i.e. the aim is to approach $\boldsymbol{x}_{\text{opt}}$ as closely as possible, where $\boldsymbol{x}_{\text{opt}}$ is defined by $f(\boldsymbol{x}_{\text{opt}}) \leq f(\boldsymbol{x})\ \forall \boldsymbol{x}$. Now, let $\mathbb{O}$ be a set of stochastic continuous optimization methods, e.g. Evolution

Strategies (ESs). To keep things simple, the aim is to define a quality measure based on the output of $f(\boldsymbol{x})$ to compare any two of these optimization methods.

Let $O \in \mathbb{O}$ be some optimization method of interest. Then we define $y^{O,f} = f(\boldsymbol{x}^O_{\text{opt}})$[1], where $\boldsymbol{x}^O_{\text{opt}}$ is the best instance of $\boldsymbol{x}$ that was found by a run of optimizer O.

Although the optimizers in question are stochastic, a well-tuned optimizer will on average result in much lower values for y^O than a poorly tuned optimizer. By treating a single y^O as a *sample* from the distribution of possible outcomes, the mean $\overline{y}^O = \frac{1}{n}\sum_{i=1}^{n} y_i^O$ of multiple runs can be used as a more stable quality measure because of the Central Limit Theorem, given a large enough sample size n.

A value for this parameter n must be chosen as low as possible to reduce computational effort. But, as n increases, the standard error $s^O = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i^O - \overline{y}^O)^2}$ associated with $\overline{y}^O$ will decrease. This can be used to calculate the *uncertainty* of a comparison between any two optimizers.

Let $A, B \in \mathbb{O}$ be two optimizers to be compared. When $\overline{y}^A < \overline{y}^B$, we say that A performs better than B. However, there is a non-zero probability that A and B are at least indistinguishable in terms of quality. This probability, denoted here as $P(A \equiv B)$, will be used as an uncertainty indication for the comparison $P(A \equiv B) = 2(1 - cdf(t))$, where t is the Welch's *t-test*

$$t = \frac{|\ \overline{y}^B - \overline{y}^A\ |}{s_e} \tag{1}$$

and the standard error s_e is calculated as $s_e = \sqrt{\frac{(s^A)^2 + (s^B)^2}{n}}$ Furthermore, *cdf* is the cumulative distribution function of the *t-distribution* with $2n - 2$ degrees of freedom.

To limit the number of variables for these calculations, the relative distances $d = (\overline{y}^B - \overline{y}^A)/\overline{y}^A$ are considered, where $\overline{y}^A < \overline{y}^B$. The standard errors are proportionally set to $s^B = (1 + d)s^A$. Specifically, this reduces Equation 1 to $t = d/s_e$, and establishes the relationship between the parameter n and the uncertainty of comparisons between any two optimizers A and B. The experiments that were performed to determine a useful value for this n are described in Section V-B.

B. Encoding

Table I provides a summary of the ES modules considered in this paper in the same order as introduced in Section III-A. By choosing integers to represent the different modules and listing them in the specified order, the structure of an ES can be represented as a list of integers $\vec{r} = r_1 r_2 \ldots r_m$, where m is the number of available module choices. The resulting representations range from **00000000000** (default CMA-ES), to **11111111122** (CMA-ES with all modules activated).

Decoding a given representation $\vec{r}$ works as follows: For each integer r_i in the representation $\vec{r}$, find the ES module i in Table I, and use the option indicated by r_i. For example, the representation $\vec{r} =$ **01100000100** represents the non-default option for ES modules 2, 3 and 9: *elitism*, *mirrored sampling* and *pairwise selection*. In other words: A (μ+λ)-mirrored-CMA-ES with pairwise selection.

[1]The function f will be omitted from the notation of $y^{O,f}$ in the following, as comparisons between different functions are meaningless.

TABLE I: *Overview of the available ES modules studied in this paper.* For most of these modules the only required options are *off* and *on*, encoded by the values 0 and 1. For quasi-Gaussian sampling and increasing population, the additional option is encoded by the value 2. The entries in row 9, recombination weights, specify the formula for calculating each weight w_i.

#	Module name	0 (default)	1	2
1	Active Update	off	on	-
2	Elitism	(μ, λ)	$(\mu+\lambda)$	-
3	Mirrored Sampling	off	on	-
4	Orthogonal Sampling	off	on	-
5	Sequential Selection	off	on	-
6	Threshold Convergence	off	on	-
7	TPA	off	on	-
8	Pairwise Selection	off	on	-
9	Recombination Weights	$\log(\mu+\frac{1}{2}) - \frac{\log(i)}{\sum_j w_j}$	$\frac{1}{\mu}$	-
10	Quasi-Gaussian Sampling	off	Sobol	Halton
11	Increasing Population	off	IPOP	BIPOP

C. Genetic Algorithm

A mutation only, self-adaptive GA according to Kruisselbrink *et al.* [19] is used as optimizer for the ES-structure (see Algorithm 2). Crossover is omitted to reduce the number of exogenous parameters of the GA. An individual in the GA consists of an ES-structure $\vec{r}$ (previously described in Section IV-B) and the self-adaptive mutation rate p_{m}. This algorithm was chosen because of its fast and reliable convergence, as shown in [19].

Algorithm 2 $(1, \lambda)$-self-adaptive GA

1: $t \leftarrow 0$
2: $P^{(0)} \leftarrow$ generate individual $\vec{I}$, randomly
3: **while** not terminate **do**
4: **for** $i = 1$ to λ **do** *// Create λ offspring*
5: $(\vec{r}_i, p_{\text{m},i}) = \vec{I}_i \leftarrow \text{copy}(P^{(t)})$
6: $p_{\text{m},i} \leftarrow \text{mutate\_}p(p_{\text{m},i})$ *// Update mutation rate*
7: $\vec{r}_i \leftarrow \text{mutate}(\vec{r}_i, p_{\text{m},i})$ *// Update ES structure*
8: *// with mutation rate $p_{m,i}$*
9: $f_i \leftarrow \text{evaluate}(\vec{r}_i)$
10: **end for**
11: $P^{(t+1)} \leftarrow \vec{I}_{1:\lambda}$, select single best from λ
12: $t \leftarrow t + 1$
13: **end while**

The mutation of $\vec{r}$ (line 7) occurs by changing each value r_j in $\vec{r}$ with probability $p_{\text{m},i}$. The value r_j is flipped between zero and one if there are two available options. For the cases where there are three options, a random 50/50 choice is made from the two remaining options, ensuring that a value selected for mutation, is always actually changed.

V. Experiments

The performed experiments and their results are described in this section. The setup for all experiments is listed in Section V-A. Next, the process by which the number of runs per ES was determined is described in Section V-B, with the final results following in Section V-C.

A. Setup

All 24 noiseless functions of the well-known *black-box optimization benchmark* (BBOB) suite [12] are used in 2, 3, 5, 10 and 20-D (dimensions), for a total of 120 experiments. As target values are known, *Fixed Cost Error* (FCE) and *Estimated Running Time* (ERT) values can be calculated for every ES. The combination of these two values will be referred to as the *fitness* of an ES. The ERT values are more informative, but can only be calculated if a target value is reached by at least one of the independent runs. Comparisons between ESs are therefore initially done only by comparing ERT values. If an ERT value is only available for one of two ESs, that ES is declared to be better. Only when both ESs do not have an ERT value available, is the FCE value used for the comparison. The BBOB default target value of 10^{-8} is used for these experiments.

A (1,12)-GA with a budget of 240 ES-structure evaluations is used, based on initial experiments with various population and budget sizes. Every run of an ES in turn is given a budget of 10^3D function evaluations. Our framework is written in Python using the *mpi4py* package [8] and was run on the DAS-4 cluster [6], allowing parallelization of both the twelve individuals per generation of the GA and the number of independent runs n (determined in Section V-B) per ES.

A brute force search over all possible ES-structures with default parameter values from literature is performed to evaluate convergence of the search towards the best ES possible within the framework. Although this brute force search is at the limit of what is computationally practical, it has a large benefit for the GA runs. Instead of running the encoded ES again for every individual the GA has to evaluate, the associated ERT/FCE values can simply be retrieved from storage. This vastly reduces the additional time spent on running the GA and allows us to use the average results from 30 runs for each experiment.

B. Uncertainty of Comparisons Between ESs

For these initial experiments, a set of 256 independent runs for around 4 000 ES-structures is used. The used ESs are generated by the GA described in Section IV-C, spread out over a representative sample of test functions in multiple dimensionalities. When ERT values were available for both ESs, or if only FCE values were available, the distance is easily calculated by $|\overline{y}^A - \overline{y}^B|$. When only one ERT value was available, the distance is calculated between the FCE of the ES without ERT, and the target value. As the absolute resulting values can differ a lot between different functions and dimensionalities, these distances are calculated only between ES results for each function/dimensionality combination.

First, an empirical distribution of relative distances is obtained from these preliminary runs. For the 40% smallest distances, $d \leq 1$ holds, while $d \geq 100$ is the case for the 30% largest distances. Due to the large spread of these distances, the uncertainty will be calculated for the distances at 5% intervals according to the cumulative distribution.

Next, 100-fold subsampling is used to simulate having run each ES only $2 \leq n \leq 256$ times. The standard error is calculated for every sample, and averaged over all samples. Small subsamples can be used as accurate representations of expected samples, but for (much) larger samples, this accuracy is lost as the samples always converge to the mean and variance of the set that is sampled from.

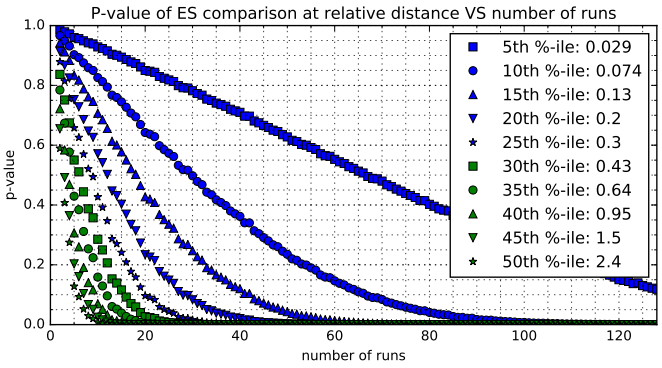


Fig. 1: Uncertainty upper bound of comparisons between ESs for the $x\%$ smallest distances d

Figure 1 shows how the uncertainty decreases with the number of runs. For the 15-20% smallest distances, there is still a relatively high uncertainty at a large number of runs. As the relative distances are only 20%, such high uncertainty is to be expected. For greater relative distances, the uncertainty drops rapidly once a sample size $n \geq 10$ is used.

In this case, we care about the comparison uncertainty because it is of importance to the automated optimization performed by the GA. For all large differences, we want to be very sure that the better one is always chosen, but for increasingly smaller differences, the GA is effectively only building a set of similarly performing ESs. For this reason, we choose to accept $\leq 5\%$ uncertainty for the largest 80% of all distances as sufficient. Combined with the DAS-4 architecture accommodating 16 parallel processes per computation node, a value of $n = 32$ is used for all further experiments.

C. Results

A GA can effectively evolve improving ES structures using our framework. The average runtime of a full brute force run for a single function-dimensionality combination is *between four and five hours* on the DAS-4 cluster. Calculating from this, a single GA run will on average last around 15 minutes.

The evolved ES-structures are often (much) better when comparing the ERT and FCE values of ES-structures found by the GA to those of some standard ES configurations. For example: the ERT values of IPOP-CMA-ES are on average 3.9 times higher than that of structures found by the GA and at best is only outperformed by 20%. Because we are more interested in what kind of structures the GA has produced, further results can be found in the extended version of this paper[2].

Figure 2 illustrates that all runs of the GA show convergence in their budget of 20 generations. The behavior of the GA for eight of the "easier" functions can be seen in Figure 2a. For these functions, most of the possible ES-structures are able to consistently reach the target value of 10^{-8}, and the GA is able to improve on the ERT over time.

For functions that are more difficult for an ES to optimize, behavior similar to that in Figure 2b is seen. Not all ES

[2]see https://arxiv.org/abs/1610.05231

TABLE II: *Ranking of GA-found ESs.* The *Fitness* row lists the cumulative percentage out of 120 experiments in which the average ERT/FCE values of the ESs found by the 30 runs of the GA reaches at least given rank when placed among all results from the BF runs, sorted by fitness.

Rank	1	2	3	4–5	6–9	10–17	18+
Fitness (%)	0.0	12.5	37.5	73.33	92.5	99.17	100

runs are able to reach the target value, so improvements in the FCE measure can be seen alongside ERT improvements. When none of the runs reach the target value, an ERT value is unavailable. This occurs most often in the 10- and 20-dimensional experiments. Note that the FCE measure is not strictly decreasing when an ERT measure is available, as FCE values are ignored in the comparison between ERT values.

Figure 2c shows convergence for functions where the GA only achieves minimal improvements. Whereas the FCE for high-dimensional cases shows improvement in Figure 2b, this is much less the case for these functions.

A ranking of the aggregated results by the 30 runs of the GA among all BF runs is shown in Table II. This ranking overwhelmingly positive: the aggregated results always rank among the top-20 out of the $4\,608$ possible structures. Although the 0% at rank one indicates that the GA was unable to find the best possible solution in all of the 30 runs, the 12.5% at rank two shows that the GA often finds the best possible solution.

Having established that most GA-found results correspond to the best ES-structures found by BF, Table III shows the relative activation of the eleven available modules over all experiments. These results again show a lot of similarity between the GA and BF runs, with minor differences. Overall, some modules are only activated in few cases such as alternative recombination weights (15–20%) and threshold convergence (23–29%), while others seem to be activated more evenly such as Orthogonal Sampling and Elitism with total activation percentages around 50%. Most successful are Mirrored Sampling (58%), Increasing Population (78%) and Quasi-Gaussian Sampling (84%).

Separating the activation percentages by function subgroups provides some additional insight. For most subgroups, the percentages correspond quite well to the aggregated values, but the third and fourth subgroups (F10–F14 and F15–F20) show interesting behavior. Table III illustrates this with highlighted outlying values. All higher than average activation percentages occur with the third subgroup of unimodal functions with high conditioning. Especially Elitism with Threshold Convergence seems to be very effective for this group of functions. Meanwhile, most of the lower than average percentages are for the fourth subgroup of multi-modal functions with adequate global structure. Elitism, Threshold Convergence and Two-Point step-size Adaptation are almost never active in the best performing ESs for these functions. This is almost a complete opposite of the results for the third subgroup.

Table IV separates the activation percentages by problem dimensionality. This highlights a trend of increasing or decreasing performance for some of the available modules. Active, Elitism and Threshold Convergence for example are selected more often for lower dimensionality problems, while Mirrored and Orthogonal Sampling are increasingly activated for problems with higher dimensionality. Combined with the consistently high activation percentages of Quasi-Gaussian sampling, these results suggest that especially in high-dimensional problems, the success of an ES depends on its ability to uniformly sample the neighboring search space.

A particular case to examine is the correlation between Mirrored Sampling and Pairwise Selection, because of their paired introduction to reduce sampling bias. With exception of the third subgroup, Mirrored Sampling is active in 60% or more of all cases. Pairwise selection however is only active in 20–25% of the experiments, or only one third as often. A similar effect is seen also in the third subgroup, where both modules are only activated half as often as in the other subgroups, maintaining the ratio between them.

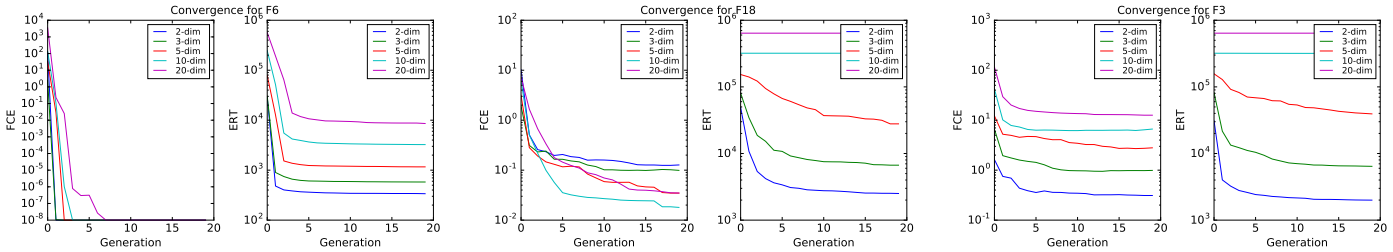
VI. Conclusions and Outlook

By extracting structural features from different CMA-ES variations, a modular framework for running new ES-structures is created, in which the structures (configurations) of CMA-ES are optimized for different optimization problems. ES-modules considered in this work are implemented in a way that they can be activated independently, with minimal dependency checking. In addition, this framework is also extensible for other modules that are not considered in this research.

Based on the empirical study using the well-known BBOB function suite, is it clear that the proposed approach on ES structure configuration, exploiting a self-adaptive genetic algorithm, consistently produces results that are comparable to the top 0.5% of the best results from the brute force search. The GA achieves such a result on 240 ES-structures, constituting only 5% of the whole configuration space, showing the advantage of the GA when more ES structure modules are incorporated into the framework. Furthermore, note that in real applications where function evaluations are costly, the overhead on determining the fitness of ES structures could grow drastically. In this case, the proposed GA-based search is more computational tractable than the exhaustive search.

Across function groups and problem dimensions, the analysis of the selected ES-structures clearly confirms that different modules or ES-variants may excel in some cases, at the cost of reduced performance for other cases. Overall, the most successful modules are Increasing Population and Quasi-Gaussian Sampling. Note that (B)IPOP is chosen very often, even with the limited budget of 10^3D function evaluations. Elitism, Two-Point Step-Size Adaptation and Threshold Convergence perform best for unimodal functions with high conditioning. Alternatively, Elitism, Active update and Sequential Selection are recommended for low dimensionality problems, while for increasingly higher dimensionalities, use of Mirrored and Orthogonal Sampling give the most likely increase in performance.

In the future research, the impact of each ES module can be analyzed by constructing a data-driven model (e.g. decision tree) on the ES structures and their performance measures. The most influential ES module on a particular function class can be related to the landscape features (e.g. convexity, ruggedness) of this function. Such relations could help to verify theoretical



(a) Convergence for F6, similar to 1, 2, 5, 8, 10, 11 and 14.

(b) Convergence for F18, similar to 7, 9 13, 16, 17, 23 and 24.

(c) Convergence for F3, similar to 4, 12, 15, 19, 20, 21 and 22.

Fig. 2: *GA convergence*. The above graphs show the average rate of convergence during the optimization of ES-structures by 30 runs of the GA for different BBOB optimization functions. Both the FCE and ERT of the best found structure over time is shown. Results for the optimization process in all five dimensionalities have been plotted per function. As the graphs for groups of several functions are similar, an example has been chosen as representative for each group. The group of functions that a single graph represents is listed below the graph.

TABLE III: *Module activation percentages by function subgroup.* This table lists the frequency of activated algorithmic modules among the experimental cases, obtained by both all 30 runs per experiment of the Genetic Algorithm (GA) and Brute Force search (BF). Results are separated by the BBOB function subgroups. Except for the second subgroup that consists of four functions, five functions make up each subgroup, for a total of 20 or 25 experiments. For the bottom two rows, the two slash-separated values indicate which of the two choices for the module was chosen. **Bold** values indicate a relatively **high** activation percentage for a module in a particular subgroup, while a ***bold italic*** value indicates a relatively ***low*** activation percentage.

	F1–F5		F6–F9		F10–F14		F15–F19		F20–F24		Average	
Module name	GA	BF	GA	BF	GA	BF	GA	BF	GA	BF	GA	BF
Active Update	33	36	11	20	**40**	**44**	19	20	29	32	27.0	30.8
Elitism	41	44	45	45	**70**	**72**	***12***	***12***	54	48	44.3	44.2
Mirrored Sampling	61	60	62	60	***39***	***40***	73	68	59	60	58.4	57.5
Orthogonal Sampling	61	68	54	45	49	44	56	64	51	52	54.1	55.0
Sequential Selection	40	36	39	45	45	48	***21***	***12***	30	44	34.8	36.7
Threshold Convergence	26	36	17	25	**53**	**56**	***2***	***4***	15	24	22.6	29.2
TPA	38	44	35	45	**53**	**56**	***9***	***4***	21	28	31.1	35.0
Pairwise Selection	24	20	23	20	***6***	***4***	20	24	35	20	21.3	17.5
Recombination Weights	16	20	***7***	***5***	**30**	**32**	20	24	15	12	17.9	19.2
Sobol/Halton	51/32	48/32	55/32	45/45	45/47	40/56	52/35	56/40	42/38	48/44	48.8/36.8	47.5/41.7
IPOP/BIPOP	42/35	52/32	37/37	30/25	28/25	36/20	53/46	40/60	34/57	36/56	38.9/39.9	39.2/39.2

TABLE IV: *Module activation percentages by dimensionality.* This table lists the frequency of activated algorithmic modules among the experimental cases, obtained by both all 30 runs per experiment of the Genetic Algorithm (GA) and Brute Force search (BF). For the bottom two rows, the two slash-separated values indicate which of the two choices for the module was chosen. **Bold** values indicate a relatively **high** activation percentage for a module in a particular subgroup, while a ***bold italic*** value indicates a relatively ***low*** activation percentage.

	2D		3D		5D		10D		20D		Average	
Module name	GA	BF	GA	BF	GA	BF	GA	BF	GA	BF	GA	BF
Active Update	**53**	**67**	40	50	25	17	5	8	13	13	27.0	30.8
Elitism	**72**	**75**	**66**	**71**	42	46	23	13	19	17	44.3	44.2
Mirrored Sampling	***36***	***33***	50	42	66	71	**72**	**75**	**68**	**67**	58.4	57.5
Orthogonal Sampling	38	37	35	33	60	75	**74**	**75**	63	54	54.1	55.0
Sequential Selection	47	46	42	46	29	25	31	33	24	33	34.8	36.7
Threshold Convergence	35	42	38	50	19	29	***11***	***13***	***10***	***13***	22.6	29.2
TPA	41	46	45	50	30	42	22	21	18	17	31.1	35.0
Pairwise Selection	21	21	27	25	18	8	16	8	24	25	21.3	17.5
Recombination Weights	24	25	16	21	18	25	18	17	14	8	17.9	19.2
Sobol/Halton	47/43	33/54	51/38	54/42	27/63	21/71	55/26	67/25	63/14	63/17	48.8/36.8	47.5/41.7
IPOP/BIPOP	35/39	38/42	34/44	25/46	38/39	46/33	37/51	38/54	50/27	50/21	38.9/39.9	39.2/39.2

hypothesis on the optimization problem or even gain more knowledge on the problem. The proposed framework should be extended with other modules, which make the configuration space even larger. In addition, this framework should be tested on some problem class that are summarized from complex real-world optimization problems.

VII. Acknowledgments

The authors would like to thank the reviewers for their helpful comments.

References

[1] A. Auger, D. Brockhoff, and N. Hansen. Mirrored sampling in evolution strategies with weighted recombination. In *Proceedings of the 13th Annual Conference Companion on Genetic and Evolutionary Computation*, GECCO '11, pages 861–868. ACM, 2011.

[2] A. Auger and N. Hansen. A restart CMA evolution strategy with increasing population size. In *Evolutionary Computation, 2005. The 2005 IEEE Congress on*, volume 2, pages 1769–1776. IEEE, 2005.

[3] A. Auger, M. Jebalia, and O. Teytaud. Algorithms (x, sigma, eta): quasi-random mutations for evolution strategies. In *Artificial Evolution: 7th International Conference, Revised Selected Papers*, pages 296–307. Springer, 2006.

[4] T. Bäck. *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*. Oxford University Press, Oxford, UK, 1996.

[5] T. Bäck, C. Foussette, and P. Krause. *Contemporary Evolution Strategies*. Natural Computing Series. Springer Berlin Heidelberg, 2013.

[6] H. Bal, D. Epema, C. de Laat, R. van Nieuwpoort, J. Romein, F. Seinstra, C. Snoek, and H. Wijshoff. A Medium-Scale Distributed System for Computer Science Research: Infrastructure for the Long Term. *Computer*, 49(5):54–63, May 2016.

[7] D. Brockhoff, A. Auger, N. Hansen, D. V. Arnold, and T. Hohm. Mirrored Sampling and Sequential Selection for Evolution Strategies. In R. Schaefer, C. Cotta, J. Kołodziej, and G. Rudolph, editors, *Parallel Problem Solving from Nature, PPSN XI: 11th International Conference, Kraków, Poland, September 11-15, 2010, Proceedings, Part I*, pages 11–21, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[8] L. Dalcn, R. Paz, and M. Storti. MPI for Python. *Journal of Parallel and Distributed Computing*, 65(9):1108–1115, 2005.

[9] J. J. Grefenstette. Optimization of Control Parameters for Genetic Algorithms. *IEEE Transactions on Systems, Man, and Cybernetics*, 16(1):122–128, Jan. 1986.

[10] N. Hansen. CMA-ES with Two-Point Step-Size Adaptation. *CoRR*, abs/0805.0231, 2008.

[11] 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, pages 2389–2396. ACM, 2009.

[12] N. Hansen, S. Finck, R. Ros, and A. Auger. Real-Parameter Black-Box Optimization Benchmarking 2009: Noiseless Functions Definitions. Research Report RR-6829, INRIA, 2009.

[13] N. Hansen and A. Ostermeier. Adapting arbitrary normal mutation distributions in evolution strategies: The covariance matrix adaptation. In *Evolutionary Computation (CEC), 1996 IEEE Congress on*, pages 312–317. IEEE, 1996.

[14] J. H. Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*, volume viii. U Michigan Press, Oxford, England, 1975.

[15] G. Jastrebski, D. V. Arnold, et al. Improving evolution strategies through active covariance matrix adaptation. In *Evolutionary Computation (CEC), 2006 IEEE Congress on*, pages 2814–2821. IEEE, 2006.

[16] W. Kantschik, P. Dittrich, M. Brameier, and W. Banzhaf. Empirical Analysis of Different Levels of Meta-Evolution. In *Evolutionary Computation (CEC), 1999 IEEE Congress on*, volume 3, page 2093 Vol. 3, 1999.

[17] W. Kantschik, P. Dittrich, M. Brameier, and W. Banzhaf. Meta-Evolution in Graph GP. In R. Poli, P. Nordin, W. B. Langdon, and T. C. Fogarty, editors, *Genetic Programming*, number 1598 in Lecture Notes in Computer Science, pages 15–28. Springer Berlin Heidelberg, May 1999.

[18] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA, 1992.

[19] J. Kruisselbrink, R. Li, E. Reehuis, J. Eggermont, and T. Bäck. On the Log-Normal Self-Adaptation of the Mutation Rate in Binary Search Spaces. In *Proceedings of the 13th Annual Conference Companion on Genetic and Evolutionary Computation*, GECCO '11, pages 893–900. ACM, 2011.

[20] N. Lourenço, F. Pereira, and E. Costa. Evolving Evolutionary Algorithms. In *Proceedings of the 14th Annual Conference Companion on Genetic and Evolutionary Computation*, GECCO '12, pages 51–58, New York, NY, USA, 2012. ACM.

[21] M. A. Martin and D. R. Tauritz. Evolving Black-box Search Algorithms Employing Genetic Programming. In *Proceedings of the 15th Annual Conference Companion on Genetic and Evolutionary Computation*, GECCO '13, pages 1497–1504, New York, NY, USA, 2013. ACM.

[22] M. Oltean. Evolving evolutionary algorithms for function optimization. In K. C. (et al), editor, *The 7th Joint Conference on Information Sciences*, volume 1, pages 295–298, North Carolina, Sept. 2003. Association for Intelligent Machinery.

[23] M. Oltean. Evolving Evolutionary Algorithms using Linear Genetic Programming. *Evolutionary Computation*, 13(3):387–410, 2005.

[24] M. Oltean and C. Groşan. Evolving Evolutionary Algorithms Using Multi Expression Programming. In W. Banzhaf, J. Ziegler, T. Christaller, P. Dittrich, and J. T. Kim, editors, *Advances in Artificial Life*, number 2801 in Lecture Notes in Computer Science, pages 651–658. Springer Berlin Heidelberg, Sept. 2003.

[25] A. Piad-Morffis, S. Estevez-Velarde, A. Bolufe-Rohler, J. Montgomery, and S. Chen. Evolution strategies with thresheld convergence. In *Evolutionary Computation (CEC), 2015 IEEE Congress on*, pages 2097–2104, May 2015.

[26] I. Rechenberg. *Evolutionsstrategie – Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Frommann-Holzboog-Verlag, 1973.

[27] L. Spector and A. Robinson. Genetic Programming and Autoconstructive Evolution with the Push Programming Language. *Genetic Programming and Evolvable Machines*, 3(1):7–40, Mar. 2002.

[28] J. Tavares, P. Machado, A. Cardoso, F. B. Pereira, and E. Costa. On the Evolution of Evolutionary Algorithms. In M. Keijzer, U.-M. O'Reilly, S. Lucas, E. Costa, and T. Soule, editors, *Genetic Programming*, number 3003 in Lecture Notes in Computer Science, pages 389–398. Springer Berlin Heidelberg, Apr. 2004.

[29] H. Wang, M. Emmerich, and T. Bäck. Mirrored Orthogonal Sampling with Pairwise Selection in Evolution Strategies. In *Proceedings of the 29th Annual ACM Symposium on Applied Computing*, pages 154–156. ACM, 2014.

Appendix

TABLE V: *Common ES Variants* A selection of ten common ES variants is listed here, as referred to in Section IV-B.

Variant	Representation
CMA-ES	00000000000
Active CMA-ES	10000000000
Elitist CMA-ES	01000000000
Mirrored-pairwise CMA-ES	00100001000
IPOP-CMA-ES	00000000001
Active IPOP-CMA-ES	10000000001
Elitist Active IPOP-CMA-ES	11000000001
BIPOP-CMA-ES	00000000002
Active BIPOP-CMA-ES	10000000002
Elitist Active BIPOP-CMA-ES	11000000002