Per-instance configuration of the modularized CMA-ES by means of classifier chains and exploratory landscape analysis

Prager, R.P.; Traumann, H.; Wang, H.; Bäck, T.H.W.; Kerschke, P.

Per-Instance Configuration of the Modularized CMA-ES by Means of Classifier Chains and Exploratory Landscape Analysis

1st Raphael Patrick Prager
Statistics and Optimization
University of Münster
Münster, Germany
raphael.prager@wi.uni-muenster.de

2nd Heike Trautmann
Statistics and Optimization
University of Münster
Münster, Germany
trautmann@wi.uni-muenster.de

3rd Hao Wang
LIP6
Sorbonne Université
Paris, France
hao.wang@lip6.fr

4th Thomas H. W. Bäck
Leiden Institute of Advanced Computer Science
University of Leiden
Leiden, The Netherlands
t.h.w.baeck@liacs.leidenuniv.nl

5th Pascal Kerschke
Statistics and Optimization
University of Münster
Münster, Germany
kerschke@uni-muenster.de

***Abstract*—In this paper, we rely on previous work proposing a modularized version of CMA-ES, which captures several alterations to the conventional CMA-ES developed in recent years. Each alteration provides significant advantages under certain problem properties, e.g., multi-modality, high conditioning. These distinct advancements are implemented as modules which result in $4\,608$ unique versions of CMA-ES. Previous findings illustrate the competitive advantage of enabling and disabling the aforementioned modules for different optimization problems. Yet, this modular CMA-ES is lacking a method to automatically determine when the activation of specific modules is auspicious and when it is not. We propose a well-performing instance-specific algorithm configuration model which selects an (almost) optimal configuration of modules for a given problem instance. In addition, the structure of this configuration model is able to capture inter-dependencies between modules, e.g., two (or more) modules might only be advantageous in unison for some problem types, making the orchestration of modules a crucial task. This is accomplished by chaining multiple random forest classifiers together into a so-called Classifier Chain based on a set of numerical features extracted by means of Exploratory Landscape Analysis (ELA) to describe the given problem instances.**

***Index Terms*—Single-Objective Continuous Optimization, Algorithm Configuration, Exploratory Landscape Analysis, Modularized CMA-ES, Classifier Chains**

I. Introduction

The field of optimization is focused by various research areas, such as operations research, evolutionary computation as well as computer science and mathematics in general. A variety of different strategies to tackle such optimization problems have emerged in the last decades. These approaches range from methods specifically designed for a single problem instance, to local search methods, and so-called general-purpose algorithms such as evolutionary algorithms. Recently, efforts are undertaken to move from single algorithms to more complex algorithmic 'tool boxes'. The goal of such endeavours is to provide a framework from which a multitude of different algorithms can be realized.

The intrinsic nature of all these algorithms and modularized frameworks is that they usually expose a variety of different hyperparameters which one can tamper with. Setting those to appropriate values for specific problem instances directly impacts the performance of the algorithm in question and therefore a desirable outcome. This undertaking is typically known as *algorithm configuration (AC)* [1], [2].

Closely related to algorithm configuration is the field of algorithm selection (AS). While past works propose algorithms which are dedicated to achieve good results on specific problem domains in general, recent trends rely on using a set of performance-complementary algorithms $\mathcal{A}$. Depending on a problem instance i at hand (out of a set I of instances of a problem P), a selection strategy $S(i)$ is responsible to choose the most suitable algorithm $A \in \mathcal{A}$ for any problem instance i. Exemplary strategies are parallel algorithm portfolios, algorithm schedules, or an automated selection procedure aided by machine learning (ML) models [3]. In general, all these formerly mentioned strategies strive to select an algorithm for every instance anew. This approach is also known as per-instance algorithm selection.

In contrast, the subject of AC is to find for a single algorithm A an optimal configuration $\mathbf{c} = (c_1, c_2, ..., c_n)$ of the hyperparameters algorithm A exhibits. Depending on the context, this can either be performed for a whole set of problem instances P (per-set AC) or for a single instance i (per-instance AC). However, realistically algorithms exhibit a multitude of hyperparameters of different types which can

978-1-7281-2547-3/20/$31.00 ©2020 IEEE

range from binary parameters to discrete or even continuous parameter values resulting in possibly an infinite amount of configurations. This factor separates the fields of AS and AC methodologically, meaning techniques employed in AS are rarely used in the research stream of AC [4]–[6]. Other techniques such as racing strategies are more prominent [7]. Yet, the downside of these techniques is that they require significantly more computational effort and computation time.

In this work, we attempt to bridge the gap between these two rather similar fields by employing machine learning techniques, usually applied in the AS community, on an AC problem. We are particularly interested in optimally configuring modularized algorithm frameworks. In essence, we strive to address the following research questions:

1) How well can a framework of algorithm components compete with prominent algorithms when configured appropriately?
2) Which challenges in this context arise for machine learning models in AC and what means are at our disposal to solve them?

In the remainder of this paper, we propose an approach, capable of configuring a modularized version of the most prominent meta-heuristic for single-objective continuous optimization, i.e., the *Covariance Matrix Adaption Evolution Strategy (CMA-ES)* [8]. Merits of this modular framework especially lie in its flexibility and timeliness. While it is based on the original CMA-ES (introduced by [9]), it also encompasses advances made in recent years each with its own hyperparameters. Thereby, it increases the configuration space of the conventional CMA-ES substantially. Pivotal for such an automated configuration procedure is knowledge about the problem instance prior to the actual application of the modular CMA-ES. As such, we incorporate Exploratory Landscape Analysis (ELA) – which has proven itself successfully in that regard numerous times – in our approach [10], [11]. These problem instance features serve as an input for the machine learning model. To be more precise, a so-called Classifier Chain (CC) is used because it is inherently well equipped to deal with obstacles of AC scenarios. In the end, the CC is able to predict a high-performing configuration of the modular CMA-ES for a given problem instance by means of ELA which is competitive with alternative state-of-the art optimizers.

The structure of this paper is as follows. First, we cover the basic components of our work. In other words, we introduce the modularized CMA-ES framework, Exploratory Landscape Analysis, and Classifier Chains in Section II. Thereafter, we discuss the experimental setup to generate necessary data which is used to train our model. Section IV provides an in-depth analysis of the different facets of our experiment including a discussion of the potential performance gains of the modularized CMA-ES version. This is followed by an evaluation of our model's performance. The last portion of Section IV compares our approach to common state-of-the art solvers in terms of performance. Finally, we summarize our findings w.r.t. our research questions, point out current limitations, and hint on future avenues to explore in Section V.

TABLE I
OVERVIEW OF CMA-ES MODULES AND THEIR OPTIONS [8].

#	Module name	0	1	2
1	Active Update [12]	off	on	-
2	Elitism	(μ, λ)	$(\mu + \lambda)$	-
3	Mirrored Sampling [13]	off	on	-
4	Orthogonal Sampling [14]	off	on	-
5	Sequential Selection [15]	off	on	-
6	Threshold Convergence [16]	off	on	-
7	Two-Point Step-Size Adaptation [17]	off	on	-
8	Pairwise Selection [14]	off	on	-
9	Recombination Weights	$\log(\mu + \frac{1}{2}) - \frac{\log(i)}{\sum_j w_j}$	$\frac{1}{\mu}$	-
10	Quasi-Gaussian Sampling [18]	off	Sobol	Halton
11	Increasing Population [19], [20]	off	IPOP	BIPOP

II. BACKGROUND

A. Modular CMA-ES Framework

Many different variants of the prominent solver CMA-ES have been proposed over the past two decades. Some advancements strive to accelerate the evolutionary search on simple problems such as the work of [15]. Others introduce an alternative approach to the step-size adaption of CMA-ES [17] to varying population sizes [19]. Each of these alterations offers some advantages for problems with specific characteristics. In other words, in some situations it is auspicious to use one specific variant over any other. Inspired by these circumstances, [8] devised an approach to utilize the merits offered by several of these variations. In their work, they considered eleven alterations, so-called modules, to the conventional CMA-ES. These are consolidated into a single implementation – the modular CMA-ES framework – which can be configured to resemble a multitude of different CMA-ES variants proposed in recent years. Nine of these modules offer binary options, either 'active' or 'inactive', whereas the last two offer trinary options. This results in $2^9 \cdot 3^2 = 4\,608$ distinct configurations. A brief overview of these different modules as well as their respective source and their options is given in Table I.

While some modules, e.g., (B)IPOP and Elitism, have a positive effect on a large variety of problems, other modules, such as Threshold Convergence, exhibit the opposite behavior and impede the performance on many problem instances. However, while such negative impact on the performance of a chosen configuration can be observed, [21] point out that this is not a definite indication that the respective module is inferior in principle. In fact, their findings illustrate how the activation of two or more modules in unison can become auspicious for the optimization process whereas the activation of only one of these modules becomes a hindrance. These findings illustrate that different modules do not only have different effects depending on the given problem instance but also strongly interact with each other. Hence, the decision to activate a certain module should not only be based on the problem instance at hand, it also should be considered which other modules will be activated or respectively remain inactive.

B. Exploratory Landscape Analysis

The fields of AC and AS both heavily rely on some sort of information about the underlying problem instance to make an informed decision on which configuration or algorithm to choose. Depending on the problem domain, different approaches exist [3]. In the domain of single-objective continuous black-box optimization, arguably the most prominent one is *Exploratory Landscape Analysis (ELA)* [22]. Essentially, it allows to compute numerical features, which in turn act as numerical surrogates of the different problem characteristics. Exemplary characteristics are the degree of multimodality, global structure, and variable scaling. The numerical features proposed by [22] are organized in six feature sets: curvature, convexity, levelset, local search, meta models, and y-distribution where the majority of features can be computed from a so called initial design.

In essence, this initial design is a small sample of the problem instance, either randomly sampled or by employing more sophisticated strategies, e.g., Latin Hypercube Sampling. The vast majority of ELA features requires not more than $50D$ samples to provide a reliable estimation, where D is the problem dimensionality [23]. A few features, e.g., the local search feature set, however, require additional function evaluations on top of the initial design.

Further advancements in this area extended and augmented the initial ELA features proposed by [22], i.e. the cell-mapping features [24], information content features [25], as well as the nearest better clustering features [26]. The most comprehensive collection of ELA features can be found in the R-package `flacco` [27], [28].

C. Classifier Chains

In the vast majority of published works, the building block of automated algorithm selectors is constituted of a supervised learning algorithm [3], [29]. Since both, AC and AS, share many similarities, we utilize approaches which have proven their worth in the field in AS for our purpose. Typically, AS can be treated as classification as well as regression task and the same can be applied in the context of AC. When it is considered a classification task, the input variables are the ELA features of the problem instances and the prediction is an algorithm A out of the set of algorithms $\mathcal{A}$.

A different avenue is taken in case of regression-based AS. Here, an option is to build a ML model for every algorithm in the set $\mathcal{A}$. The input variables remain the same as in a classification task. However, here the prediction is the performance (based either on time or number of function evaluations). Given a problem instance i, all features are extracted and passed to n models, where $n = |\mathcal{A}|$. The algorithm which exhibits the 'best' predicted performance is selected [3].

Within this work, we explore a different promising avenue. We still remain in the area of a classification scenario. Yet since we have $4\,608$ possible output classes, a conventional multi-class ML model underperforms due to the large number of possible classes. A more favorable approach is to predict whether a module should be active or inactive under certain

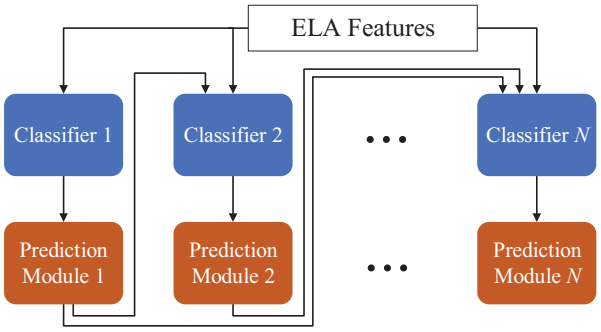

Fig. 1. Schematic representation of a Classifier Chain.

ELA features. This transforms the multi-class classification with over a thousand classes to a multi-label classification with the options of modules as possible classes, e.g., 'active' or 'inactive'. Technically, this problem is still a multi-class problem since the modules 'Increasing Population' and 'Quasi-Gaussian Sampling' offer three possibles values.

In multi-label classification, each problem instance can be associated with more than one label in contrast to multi-class classification where the labels are mutually exclusive [30]. In this context, one label refers to the option value of one module (of a CMA-ES configuration). To be able to predict a configuration for given ELA features, a chain of 'base' classifiers – a so-called *Classifier Chain (CC)* – is used. Decision trees, random forests, and radial SVMs are considered as possible base classifiers. Such a CC transforms the multi-label problem with N labels into N binary (or trinary) classification problems. Hence, each module is classified by a different binary/trinary base classifier. Yet, instead of being independent from each other, each classifier takes the output of all previous classifiers (as well as the ELA features) into consideration, thus forming a chain of classifiers. Consequently, correlations between modules can be taken into account [30]. An intrinsic limitation to CCs poses the order of base classifiers. While some interdependencies between modules can be modelled, it is not possible to model all interactions. Meaning, the first base classifier will solely make its prediction based on the ELA features. The subsequent base classifier on the other hand takes the prediction of the first base classifier into account. Given this, the internal order of base classifier can become a major contributor to the CC's performance.

Despite this limitation, a CC in general serves the purpose of modelling the interaction between modules which is of utmost importance. This is evident in the findings of [21] as well as in the later parts of this paper. While a CC cannot capture all interaction between modules, it allows us at least to model the most influential dependencies between them. The workings of such a CC in this application scenario are depicted in Fig. 1. The input for all base classifiers in this CC are the ELA features of a given problem instance i. The first base classifier predicts whether the module it is responsible for should be turned on or not. Its prediction (as well as the ELA features) are then used as an input for the second classifier which in

turn predicts the option for the module this second classifier is responsible for. Note that the last base classifier takes all previous predictions into account, while the first operates only on the ELA features.

III. Experimental Setup

In order to implement the aforementioned algorithm configurator, two distinct experiments (serving different needs) are conducted. The first experiment is tasked with creating the necessary training data for the algorithm configurator, i.e., the modular CMA-ES framework, with all $4\,608$ candidate configurations, is applied to a large, representative set of problem instances. Adherent to the standard of the evolutionary optimization community, the Black-Box Optimization Benchmark (BBOB) is used to provide a performance assessment of the different configurations [31]. All 24 noise-less functions $F = \{1, 2, .., 24\}$ in dimensions $D = \{2, 3, 5, 10\}$ are considered. For each function f the first five instances $I = \{1, 2, ..., 5\}$ are used which leads to 480 problems instances in total. Note that we define a problem instance as the triplet $i := (F, D, I)$. Due to the stochastic nature of evolutionary algorithms, each candidate configuration is applied five times to gain a more stable estimation of a candidate configuration's performance. As performance indicator we use the *Expected Running Time (ERT)* which is well established in the community [32], [33]:

$$ERT = \frac{1}{s} \sum_j FE_j, \tag{1}$$

where j denotes the different repetitions of a configuration c on a problem instance i, FE is the number of function evaluations required to reach the target value F_{target} (cf. Table II), and s is the number of successful runs in terms of reaching F_{target}. In cases, where F_{target} is not reached in any of the j repetitions, i.e., $s = 0$, we impute the performance using the PAR10 score [34]. This results in a performance dataset comprising the ERT for each configuration c related to each problem instance i. Further chosen parameters of this experiment are detailed in Table II.

Pivotal for our approach are descriptive features which characterize a problem instance prior to optimization. As elaborated in Section II-B, ELA provides the necessary means. In this work, we rely on a small subset of all ELA features. This restriction simply results from the fact that some features, like the cell mapping features, are practically infeasible in $10D$, or incur additional cost, e.g., local search features. Hence, we use only the following features: 3 y-distribution, 18 levelset, 9 meta, 16 dispersion, 4 information content, and 5 nearest better clustering features as well as the problem dimensionality. For each of the 480 problem instances, the ELA features are sampled 10 times using a sample size of $50D$ obtained with Latin Hypercube Sampling. We use the R-package `flacco` to compute all features of interest [27]. Thereafter, these results are averaged using the median. This allows us to obtain a more reliable estimation and reduces stochastic interference.

As mentioned in Section II-C, the order of classifiers within a CC is a decisive factor when it comes to the model's performance. Hence, within the second experiment we evaluated all possible sequences of base classifiers. Since we have base classifiers for each module, this results in $11! = 39\,916\,800$ different possible arrangements within a CC. We refrain to endeavour in the realm of parameter tuning at this point and postpone this until we have identified a small subset of promising CCs. This is mainly done to reduce the already immense computational effort required. In addition, instead of using more computationally intensive model assessment methods like cross validation, we use all BBOB functions in all considered dimensions but only the instances 1-4 as training data. The fifth instance of each function-dimension pairing is used to evaluate our model in this first experiment. This results in 384 problem instances in the training set and 96 instances in the validation set V. As base classifiers, we employ random forest using the Python package `scikit-learn` [35].

Furthermore, ERT values in general are highly influenced by the problem dimensionality, i.e., an instance in a higher dimension tends to have a substantially larger ERT value than in a lower dimension. Hence, the selection of a sub-optimal configuration for higher dimensional instances will have a larger effect compared to lower dimensional instances. Consequently, CCs will focus more on learning problems in higher dimensional space and disregard the ones of lower dimensionality. To mitigate such unwanted effects, we use a relative ERT value, in short relERT [11]. To be more precise, we assess the performance as follows: for each $i \in V$, we obtain the configuration c_i (the prediction of the CC in question) and its corresponding ERT value. Thereafter, we identify the configuration c_i^* which exhibits the lowest ERT value on this instance i and compute the performance metric $\text{relERT}_{c_i} = \text{ERT}_{c_i}/\text{ERT}_{c_i^*}$. Thereby, we obtain an indication of how much worse a selected configuration c_i is compared to the best configuration c_i^* on problem instance i. A relERT of 1 specifies a perfect selection. After computing the relERT for each $i \in V$, we average this set of individual values by using the arithmetic mean which allows to asses the performance of a particular CC fairly.

IV. Results and Discussion

A. Performance Analysis of all Configurations

Typical in the field of algorithm selection is to chose a baseline algorithm also known as the single best solver (SBS) [11], [34]. In this case, the SBS refers to the CMA-ES configuration which exhibits the overall best performance, i.e., the performance of every configuration is averaged over all functions, dimensions, and instances using the arithmetic mean. The configuration in possession of the lowest ERT is denoted as SBS. On average, the best performing CMA-ES configuration is $c = (0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 2)$ with a relERT of 11.09. In other words, the SBS is on average 11 times slower compared to the hypothetical case where always the best configuration for a given problem instance would be selected. In regards to the SBS's configuration, most of

TABLE II
(1) VALUES OF TERMINATION CRITERIA, (2) CONVENTIONAL CMA-ES EXOGENOUS STRATEGY PARAMETERS, AND (3) CMA-ES MODULE SPECIFIC PARAMETERS. THE LINES SEPARATING THE TABLE INDICATE TO WHICH OF THE FORMER THREE TOPICS A PARAMETER BELONGS.

Parameter	Value	Description
B	$10\,000 \cdot D$	Number of maximum function evaluations allowed in a single run where D denotes the problem dimension. B is referred to as budget.
$\triangle p$	10^{-2}	Precision value.
F_{target}	$F_{opt} + \triangle p$	Target value a successful run has to reach. F_{target} exhibits a slightly worse fitness (thus is larger) than F_{opt}.
λ	$4+\lfloor 3 \cdot \log(D) \rfloor$	Size of offspring population where D denotes the problem dimension
μ	$\lfloor \lambda/2 \rfloor$	Size of population which are used in the adaptation strategies of CMA-ES.
$\mathbf{m}$	$\mathcal{U}(-4, 4)^d$	Each element of the initial mean vector is sampled uniformly within the interval $[-4, 4]$.
C	$\mathbf{I}$	The initial covariance matrix C is the identity matrix $\mathbf{I}$ of size $D \times D$ where D denotes problem dimensionality.
σ	1	Value of the initial mutation step size.
T	0.2	Initial value of module 'Threshold Convergence'
Decay Factor	0.995	Decay factor of the threshold value T
α'	0.5	Test width α' used for module 'Two-Point Step-Size Adaptation'

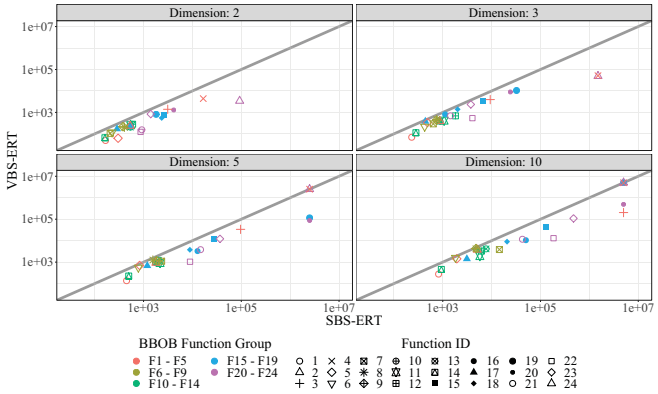

Fig. 2. Performance gap of the SBS and VBS.

the modules are inactive with the following exceptions. The module 'Mirrored Sampling' is active, 'Quasi-Gaussian Sampling' uses the Sobol sequence to sample mutation vectors, and 'Increasing Population' uses the BIPOP restart scheme.

In contrast to the SBS, the virtual best solver (VBS) is the best known algorithm for all specific optimization problems individually. Here, this is limited to all possible configurations, meaning only CMA-ES configurations are considered as potential VBS. Consequently, the VBS for a specific BBOB function-dimension-instance pairing is the best performing configuration which is not necessarily (and here most likely not) the SBS. Thus, the VBS can be perceived as an intangible optimization algorithm that always achieves the best possible performance on any problem instance [34]. We already used the notion of the VBS in the previous section without explicitly referring to it as such. There it is used as the denominator to compute the relERT. The performance contrast of the SBS compared to the VBS is illustrated in Fig. 2. Points on the diagonal refer to an identical ERT of the SBS and VBS on a given problem instance. The larger the distance to this diagonal becomes horizontally (parallel to the x-axis), the worse the SBS performs compared to the VBS.

On F6-F14, i.e., BBOB function groups 2 and 3, the respective points are located in the lower left quadrant of the figures in all dimensions. SBS and VBS require substantially less function evaluations to solve these problem instances. While there is a performance gap between those two, it is not as significant as for the other BBOB function groups. The performance of the SBS diverges peculiarly on functions belonging to the last two BBOB function groups as well as on the first function group. Selecting the respective VBS and therefore a different CMA-ES configuration is by far more auspicious than using the SBS. Based on these findings, we argue that the development of an algorithm configurator indeed has merit. By selecting the VBS instead of the SBS the evolutionary search can be accelerated by orders of magnitude on some instances.

B. Performance Evaluation of 11! CCs

As shown by [21], the performance of a configuration c is largely dependent on how well different modules complement each other. Two questions are of foremost interest. The first one investigates whether modules interact with each other at all. In cases where an interaction is present, the second question is how to efficiently exploit this behaviour. Fig. 3 provides the performance distribution of all 11! distinct sequences of CCs. The lower performance bound is given by the VBS, depicted as a green line. The large range of differently performing CCs highlights the presence of strong module interactions. The best-performing CC is able to achieve a relERT of 8.76 whereas the worst-performing CC is on average 114.31 times slower than the VBS. In fact, the majority of CCs perform worse than the SBS (colored in blue with an relERT of 11.09). However, 6 156 out of the 39 916 800 provide a better performance in general, without any hyperparameter tuning of the base classifiers so far. This large performance range highlights the strength of interaction between modules and the effect it can have on the overall competitiveness of a CC.

To enhance this promising subset of CCs further, we select the three most promising CCs with the following internal order of modules (cf. first column of Table I):

1) $CC_1 = (4, 8, 10, 6, 1, 2, 3, 11, 5, 7, 9)$
2) $CC_2 = (9, 2, 8, 5, 1, 4, 6, 7, 3, 10, 11)$
3) $CC_3 = (8, 6, 7, 3, 10, 9, 11, 5, 4, 1, 2)$

These particular CCs are subjected to further tuning. This tuning procedure is undertaken using `irace` [7]. In particular, we tune the following hyperparameters of the random forest base classifiers: $p :=$ (`n_estimators`,

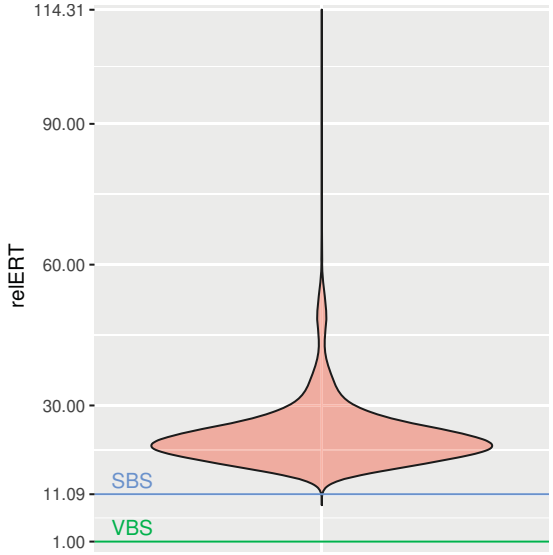

Fig. 3. Performance range of different CCs before tuning.

`max_Features`, `max_depth`, `min_samples_split`, `min_samples_leaf`, `bootstrap`).

After successfully applying parameter tuning, we use CC_1 as our final model because it demonstrates the best performance. The values of the tuned hyperparameters (of the random forests) are $p = (944, \text{sqrt}, 20, 2, 2, \text{False})$, where the option 'sqrt' sets `max_Features` to the square root of the number of input features. The resulting value is rounded down.

C. Performance Comparison of the Algorithm Configurator

In this step, we use leave-one-out cross validation to assess our model. Meaning, we train n CCs on $n-1$ training samples, where $n = |P|$. The left out problem instance in each of these n CCs is used to predict the performance of that particular CC [36]. This gives a more reliable estimation of our model's performance than the initial evaluation of all 11! CCs which used a simple hold-out strategy (cf. Section III).

As a baseline, we use two well-known variants of the CMA-ES, the conventional CMA-ES [9] and the IPOP-CMA-ES [19], as well as the SBS[1]. Further, we compare not only to the baseline algorithms but also to two other prominent and powerful solvers: the Brent-STEP algorithm with quadratic interpolation [37] and a hybrid CMA-ES version called HCMA [38]. The ulterior motive here is to gain insights on how well our approach competes with conventional CMA-ES variants and other well-established algorithms. Brent-STEP provides an exemplary deterministic algorithm whereas the HCMA – which resulted as SBS in a previous comprehensive AS study [11] – serves as representative of more sophisticated state-of-the-art algorithms. In particular, the HCMA is constituted of a local search component and incorporates a surrogate model making it an ample contender for our approach. Noteworthy is that both, Brent-STEP and HCMA, cannot be realized within the modular CMA-ES framework.

[1]Configurations used for the baseline algorithms:

- SBS: $c = (0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 2)$
- conventional CMA: $c = (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)$
- IPOP-CMA-ES: $c = (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1)$

TABLE III
PERFORMANCE COMPARISON USING RELERT, AGGREGATED BY PROBLEM DIMENSIONALITY (D) AND BBOB FUNCTION GROUP (FUNCTION).

		Baseline			CC		COCO	
D	Function	SBS	CMA	IPOP	no	with	HCMA	BRENT
2	01-05	110.12	401.30	106.11	42.09	48.25	0.55	0.27
	06-09	20.65	17.30	3.75	6.56	8.53	2.80	7920.81
	10-14	2.96	3.69	5.09	1.77	3.27	1.53	3065.59
	15-19	6.81	417.56	5.95	21.16	22.11	5.16	2478.98
	20-24	116.69	372.84	106.89	37.88	39.42	76.85	3664.40
	all	52.73	251.92	47.30	22.53	24.97	17.98	3238.73
3	01-05	27.93	161.67	19.96	46.90	48.33	6.46	0.09
	06-09	5.30	296.04	2.88	15.75	16.97	1.80	4284.56
	10-14	2.98	3.54	3.05	1.70	2.99	1.24	3007.97
	15-19	6.65	330.75	7.17	10.40	10.91	4.27	1946.15
	20-24	48.43	245.90	48.41	35.12	35.83	34.61	2846.85
	all	18.80	203.89	16.85	22.23	23.26	10.01	2339.31
5	01-05	51.80	38.33	27.47	4.68	5.76	0.57	0.06
	06-09	5.14	320.47	2.36	2.69	3.37	1.41	3130.59
	10-14	2.23	2.44	2.44	1.61	2.59	0.79	2129.87
	15-19	9.35	223.61	10.75	7.59	7.88	3.04	1445.89
	20-24	44.44	1161.38	180.26	10.89	11.20	8.73	1078.98
	all	23.32	350.44	46.42	5.61	6.28	2.97	1491.52
10	01-05	45.53	12.46	12.47	11.64	12.62	0.05	0.05
	06-09	5.26	454.39	1.79	1.77	2.16	1.01	1967.78
	10-14	1.99	2.51	2.27	2.24	3.00	0.58	1477.10
	15-19	3.88	316.59	15.80	8.56	8.79	2.93	1120.49
	20-24	31.62	434.65	175.75	19.60	19.75	23.51	783.99
	all	18.17	235.36	43.27	9.05	9.56	5.81	1032.47
all	01-05	58.84	153.44	41.50	26.33	28.74	1.91	0.12
	06-09	9.09	272.05	2.69	6.69	7.76	1.75	4325.93
	10-14	2.54	3.04	3.21	1.83	2.96	1.04	2420.13
	15-19	6.67	322.13	9.92	11.93	12.42	3.85	1747.88
	20-24	60.30	553.69	127.83	25.87	26.55	35.93	2093.55
	all	28.26	260.40	38.46	14.86	16.02	9.19	2025.51

Table III summarizes performance of our model in addition to the aforementioned other solvers. The first two columns provide information about the considered problem instances, e.g., the first row gives a performance indication of the different solvers only on 2-dimensional problems of the first BBOB group (F1-F5). The performance values are captured using the relERT. Values < 1 state that a better performance is accomplished compared what it is possible with the modular CMA-ES framework under a perfect selection, i.e., VBS's performance. The performance of the conventional CMA-ES, IPOP-CMA-ES, and SBS serve as baseline. The two columns 'no' and 'with' represent the final CC's performance without and with the additional costs of ELA features (which are $50D$). The last two columns serve as an external validation mechanism to evaluate how competitive our devised model is compared to other well-established solvers. Note that the performance data of the external algorithms are provided by the COCO platform[2]. However, the comparison is not entirely fair since the results submitted to the COCO platform do not require any repeated runs on a problem instance whereas our results are replicated five times. Therefore, the results of the

[2]https://coco.gforge.inria.fr

other solvers might be more prone to stochastic interference.

To facilitate a fair comparison between the SBS and the CCs, we assess the SBS performance using an approach similar to the leave-one-out procedure. That is, for each left-out problem instance out of all problems $v \in P$, we identify the SBS_v $\forall i \in P \backslash \{v\}$. The ERT and relERT of the SBS_v is then derived by applying it to the problem instance v. This means, that the SBS can differ for each problem instance v.

Overall, i.e., across all dimensions and functions, the single best configuration of the modularized CMA-ES version is superior to the conventional CMA-ES and IPOP-CMA-ES. This not only might propose a slightly adapted CMA-ES version but also highlights the potential of the modular CMA-ES framework to provide novel algorithms in general, which were not realized before. On the other hand, the VBS shows a remarkable performance in contrast to the CMA-ES and IPOP-CMA-ES. In addition to that, the other solvers HCMA and BRENT are also not capable to outperform the VBS overall. If that was the case, they would attain an relERT < 1. Rather, the VBS is superior by a large margin compared to the best competing solver (HCMA).

The final CC is able to outperform the SBS almost on every BBOB function group and thereby contributes to the current state of research by means of the modularized CMA-ES framework. A notable exception is that the SBS pulls ahead on functions 15 to 19 (BBOB function group 4). However, considering the BBOB suite in its entirety the CC with additional ELA feature costs accelerates the search by a factor of $28.26/16.02 = 1.76$, i.e., roughly 43.31% less function evaluations are required to solve a problem instance on average.

When assessing Table III, the performance disparity on the BBOB function group 1, comprising functions 1 to 5, becomes particularly visible. Our devised approach struggles to achieve a competitive performance on these (rather simple) problem instances. While the CC is capable of beating the SBS through almost all dimensions on this particular group, it falls behind the HCMA and BRENT. Two possible reasons may be the cause of this behaviour. First, these BBOB functions are the only separable functions and relatively easy to solve by any standards compared to the remaining BBOB functions. This leads to an imbalance in training samples for our model and thereby limits the learning effects for this type of problem instances. This explains the gap between the VBS and CC on this function group. However, it is also evident that the modularized CMA-ES framework is inherently limited, i.e., it is lacking a local search component. Algorithms which incorporate a local search method, such as the HCMA and BRENT, naturally outperform even the VBS of the modularized CMA-ES on this function group. A different behaviour can be observed on functions 20 to 24 which can be considered as the most difficult ones to solve. Here, our configuration approach is capable of achieving a similar or even a significantly better performance than HCMA. This might hint that some modules of the modularized CMA-ES framework are especially beneficial on these complex functions.

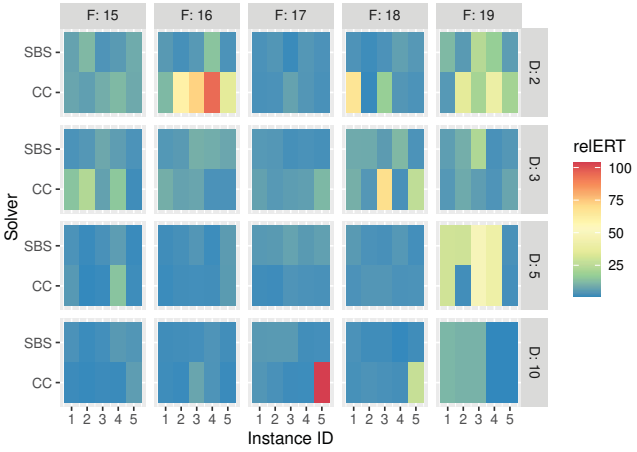

Fig. 4. Performance (in terms of the relative ERT) of the SBS and CC (with costs for ELA features), shown for the functions of BBOB function group 4.

Despite the dominance of the CC over the SBS in general, it is apparent that our model lacks behind the SBS's performance on BBOB function group 4 (F15-F19). A more detailed account of this observation is given in Fig. 4. Even when scrutinizing the performance on each function in isolation, our model is inferior to the SBS in most of the cases. Especially F16 in dimension 2 provides a challenge to our CC consistently. To this point, we lack an insightful explanation for this circumstance and aim to analyze this in future experiments.

V. Conclusion

In this paper, we propose a sophisticated approach which is capable of automatically configuring a modularized version of the CMA-ES for a given problem instance in single-objective continuous optimization based on numerical instance features.

We moreover address important challenges of per-instance algorithm configuration. By using machine learning in terms of classifier chains to determine the best configuration of the modular CMA-ES framework, we introduce a novel solution to interdependencies between algorithms' hyperparameters.

This approach consistently outperforms the single best solver, and its performance is even unmatched by some prominent state-of-the-art solvers on a small subset of BBOB functions. Across all functions and dimensionalities, our approach is able to improve the SBS by roughly 40 percent. This underlines the potential of such modular algorithm frameworks in general and provides a very promising perspective onto possible gains by extending the modular CMA-ES framework even further, e.g., by incorporating a local search module. This could even be enhanced by configuring the CMA-ES hyperparameters of the different modules specifically.

Nevertheless, it is important to mention that this approach is currently limited to discrete parameters. We plan to extend this research to parameters of arbitrary type, e.g., the test width of the module 'Two-Point Step-Size Adaptation' which is a continuous parameter. Furthermore, other research – such as [10], [11] – suggest that a small subset of ELA features

is descriptive enough and additional features may even be disturbing. Hence, feature selection is also a promising area to explore. Lastly, the work of [39] illustrates the extraordinary performance gain when switching between configurations during the evolutionary search. We will thus explore the potential of online algorithm configuration in future studies.

ACKNOWLEDGMENT

The authors acknowledge support by the *European Research Center for Information Systems (ERCIS)*.

REFERENCES

[1] K. Eggensperger, M. Lindauer, and F. Hutter, "Pitfalls and Best Practices in Algorithm Configuration," Journal of Artificial Intelligence Research, vol. 64, pp. 861 – 893, 2019.

[2] C. Huang, Y. Li, and X. Yao, "A Survey of Automatic Parameter Tuning Methods for Metaheuristics," IEEE Transactions on Evolutionary Computation, vol. 24, no. 2, pp. 201–216, 2020.

[3] P. Kerschke, H. H. Hoos, F. Neumann, and H. Trautmann, "Automated Algorithm Selection: Survey and Perspectives," Evolutionary Computation (ECJ), vol. 27, no. 1, pp. 3 – 45, 2019.

[4] F. Hutter, Y. Hamadi, H. H. Hoos, and K. Leyton-Brown, "Performance Prediction and Automated Tuning of Randomized and Parametric Algorithms," in International Conference on Principles and Practice of Constraint Programming. Springer, 2006, pp. 213 – 228.

[5] N. Belkhir, J. Dréo, P. Savéant, and M. Schoenauer, "Feature Based Algorithm Configuration: A Case Study with Differential Evolution," in PPSN XIV. Springer, 2016, pp. 156 – 166.

[6] N. Belkhir, J. Dréo, P. Savéant, and M. Schoenauer, "Per Instance Algorithm Configuration of CMA-ES with Limited Budget," in Proc. of the Genetic and Evol. Comp. Conf. ACM, 2017, pp. 681 —- 688.

[7] M. López-Ibáñez, J. Dubois-Lacoste, L. Pérez Cáceres, M. Birattari, and T. Stützle, "The irace Package: Iterated Racing for Automatic Algorithm Configuration," Oper. Research Perspectives, vol. 3, pp. 43 – 58, 2016.

[8] S. van Rijn, H. Wang, M. van Leeuwen, and T. Bäck, "Evolving the Structure of Evolution Strategies," in 2016 IEEE Symposium Series on Computational Intelligence (SSCI), 2016, pp. 1–8.

[9] N. Hansen, The CMA Evolution Strategy: A Comparing Review. Springer, 2006, pp. 75 – 102.

[10] B. Bischl, O. Mersmann, H. Trautmann, and M. Preuß, "Algorithm Selection Based on Exploratory Landscape Analysis and Cost-Sensitive Learning," in Proceedings of the 14th Annual Conference on Genetic and Evolutionary Computation. ACM, 2012, pp. 313 – 320.

[11] P. Kerschke and H. Trautmann, "Automated Algorithm Selection on Continuous Black-Box Problems by Combining Exploratory Landscape Analysis and Machine Learning," Evolutionary Computation (ECJ), vol. 27, no. 1, pp. 99 – 127, 2019.

[12] G. A. Jastrebski and D. V. Arnold, "Improving Evolution Strategies through Active Covariance Matrix Adaptation," in 2006 IEEE Int'l Conference on Evolutionary Computation, 2006, pp. 2814 – 2821.

[13] A. Auger, D. Brockhoff, and N. Hansen, "Mirrored Sampling in Evolution Strategies with Weighted Recombination," in Proc. of the 13th Annual Conf. on Genetic and Evol. Comp. ACM, 2011, pp. 861–868.

[14] H. Wang, M. Emmerich, and T. Bäck, "Mirrored Orthogonal Sampling with Pairwise Selection in Evolution Strategies," in Proc. of the 29th Annual ACM Symp. on Applied Comp. ACM, 2014, pp. 154–156.

[15] D. Brockhoff, A. Auger, N. Hansen, D. V. Arnold, and T. Hohm, "Mirrored Sampling and Sequential Selection for Evolution Strategies," in PPSN XI. Springer, 2010, pp. 11 – 21.

[16] A. Piad-Morffis, S. Estévez-Velarde, A. Bolufé-Röhler, J. Montgomery, and S. Chen, "Evolution Strategies with Thresheld Convergence," in IEEE Congress on Evolutionary Computation, 2015, pp. 2097 – 2104.

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

[18] O. Teytaud and S. Gelly, "DCMA, yet Another Derandomization in Covariance-Matrix-Adaptation," in Proc. of the 9th Annual Conf. on Genetic and Evolutionary Computation. ACM, 2007, pp. 955–963.

[19] A. Auger and N. Hansen, "A Restart CMA Evolution Strategy with Increasing Population Size," in 2005 IEEE Congress on Evolutionary Computation, vol. 2. IEEE, 2005, pp. 1769 – 1776.

[20] 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. ACM, 2009, pp. 2389 – 2396.

[21] S. van Rijn, H. Wang, B. van Stein, and T. Bäck, "Algorithm Configuration Data Mining for CMA Evolution Strategies," in Proc. of the Genetic and Evolutionary Computation Conference. ACM, 2017, pp. 737–744.

[22] O. Mersmann, B. Bischl, H. Trautmann, M. Preuss, C. Weihs, and G. Rudolph, "Exploratory Landscape Analysis," in Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation. ACM, 2011, pp. 829 – 836.

[23] P. Kerschke, M. Preuss, S. Wessing, and H. Trautmann, "Low-Budget Exploratory Landscape Analysis on Multiple Peaks Models," in Proceedings of the 18th Annual Conference on Genetic and Evolutionary Computation. ACM, 2016, pp. 229 – 236.

[24] P. Kerschke, M. Preuss, C. Hernández, O. Schütze, J.-Q. Sun, C. Grimme, G. Rudolph, B. Bischl, and H. Trautmann, "Cell Mapping Techniques for Exploratory Landscape Analysis," in EVOLVE - A Bridge between Probability, Set Oriented Numerics, and Evolutionary Computation V. Springer, 2014, pp. 115 – 131.

[25] M. A. Muñoz, M. Kirley, and S. K. Halgamuge, "Exploratory Landscape Analysis of Continuous Space Optimization Problems Using Information Content," IEEE Trans. on Evol. Comp., vol. 19, no. 1, pp. 74–87, 2015.

[26] P. Kerschke, M. Preuss, S. Wessing, and H. Trautmann, "Detecting Funnel Structures by Means of Exploratory Landscape Analysis," in Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation. ACM, 2015, p. 265–272.

[27] P. Kerschke and H. Trautmann, "Comprehensive Feature-Based Landscape Analysis of Continuous and Constrained Optimization Problems Using the R-package flacco," in Applications in Statistical Computing. Springer, 2019, pp. 93 – 123.

[28] C. Hanster and P. Kerschke, "flaccogui: Exploratory Landscape Analysis for Everyone," in Proc. of the 19th Annual Conference on Genetic and Evolutionary Computation Companion. ACM, 2017, pp. 1215 – 1222.

[29] L. Kotthoff, Algorithm Selection for Combinatorial Search Problems: A Survey. Springer International Publishing, 2016, pp. 149–190.

[30] J. Read, B. Pfahringer, G. Holmes, and E. Frank, "Classifier Chains for Multi-Label Classification," Machine Learning, vol. 85, no. 3, pp. 333 – 359, 2011.

[31] N. Hansen, S. Finck, R. Ros, and A. Auger, "Real-Parameter Black-Box Optimization Benchmarking 2009: Noiseless Functions Definitions," INRIA, Research Report RR-6829, 2009. [Online]. Available: https://hal.inria.fr/inria-00362633

[32] A. Auger and N. Hansen, "Performance Evaluation of an Advanced Local Search Evolutionary Algorithm," in 2005 IEEE Congress on Evolutionary Computation, vol. 2. IEEE, 2005, pp. 1777–1784.

[33] T. Bartz-Beielstein, C. Doerr, J. Bossek, S. Chandrasekaran, T. Eftimov, A. Fischbach, P. Kerschke, M. López-Ibáñez, Manuel, K. M. Malan, J. H. Moore, B. Naujoks, P. Orzechowski, V. Volz, M. Wagner, and T. Weise, "Benchmarking in Optimization: Best Practice and Open Issues," arXiv preprint arXiv:2007.03488, 2020.

[34] B. Bischl, P. Kerschke, L. Kotthoff, T. M. Lindauer, Y. Malitsky, A. Fréchette, H. H. Hoos, F. Hutter, K. Leyton-Brown, K. Tierney, and J. Vanschoren, "ASlib: A Benchmark Library for Algorithm Selection," Artificial Intelligence, vol. 237, pp. 41 – 58, 2016.

[35] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," Journal of Machine Learning Research, vol. 12, pp. 2825 – 2830, 2011.

[36] T. Hastie, R. Tibshirani, and J. Friedman, The Elements of Statistical Learning: Data Mining, Inference, and Prediction. Springer, 2009.

[37] P. Pošík and P. Baudiš, "Dimension Selection in Axis-Parallel Brent-STEP Method for Black-Box Optimization of Separable Continuous Functions," in Proc. of the 2015 Annual Conference on Genetic and Evolutionary Computation Companion. ACM, 2015, pp. 1151 —- 1158.

[38] I. Loshchilov, M. Schoenauer, and M. Sèbag, "Bi-population cma-es agorithms with surrogate models and line searches," in Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation Companion. ACM, 2013, pp. 1177 —- 1184.

[39] D. Vermetten, S. van Rijn, T. Bäck, and C. Doerr, "Online Selection of CMA-ES Variants," in Proceedings of the Genetic and Evolutionary Computation Conference. ACM, 2019, pp. 951 —- 959.