Towards an Adaptive CMA-ES Configurator

Sander van Rijn[1(✉)], Carola Doerr[2], and Thomas Bäck[1]

[1] LIACS, Leiden University, Niels Bohrweg 1, 2333 CA Leiden, The Netherlands
{s.j.van.rijn,t.h.w.baeck}@liacs.leidenuniv.nl
[2] Sorbonne Université, CNRS, Laboratoire d'Informatique de Paris 6, Paris, France
Carola.Doerr@mpi-inf.mpg.de

Abstract. Recent work has shown that significant performance gains over state-of-the-art CMA-ES variants can be obtained by a recombination of their algorithmic modules. It seems plausible that further improvements can be realized by an adaptive selection of these configurations. We address this question by quantifying the potential performance gain of such an online algorithm selection approach. In particular, we study the advantage of structurally adaptive CMA-ES variants on the functions F1, F10, F15, and F20 of the BBOB test suite. Our research reveals that significant speedups might be possible for these functions. Quite notably, significant performance gains might already be possible by adapting the configuration only once. More precisely, we show that for the tested problems such a single configuration switch can result in performance gains of up to 22%. With such a significant indication for improvement potential, we hope that our results trigger an intensified discussion of online structural algorithm configuration for CMA-ES variants.

Keywords: Continuous black-box optimization · CMA-ES
Online algorithm configuration

1 Introduction

Black-box optimization algorithms are an important field of research due to their direct applicability to many problems and their long success history. Many different algorithms are continuously being developed, each with its own performance characteristics to make it better suited to certain problems or problem classes.

Such optimizers typically use online adaptation of parameters such as step size or population size. A common one is the CMA-ES by Hansen *et al.* [5] and its (B)IPOP derivatives [1,7]. This adaptation behavior allows a single algorithm to behave differently at different points in time during the optimization process, usually trying to transition from *exploration* to *exploitation*.

When using specific optimizers that are tailored to the specific features of the optimization problem at hand, the efficiency is typically improved. Such

A. Auger et al. (Eds.): PPSN 2018, LNCS 11101, pp. 54–65, 2018.
https://doi.org/10.1007/978-3-319-99253-2_5

algorithms provide improved convergence rate and/or quality of the final solution found for these problems, but often perform worse than average on other problems. This limits their general applicability and makes it less likely that non-specialists will be aware that the tailored algorithm exists as an option for their particular problem. Furthermore, this also means that many algorithmic variants are not commonly combined with each other, as they are only compared with standard algorithms upon their introduction.

In an effort to explore more of this underlying algorithm combination space, a modular CMA-ES implementation for eleven such variants has been proposed in [12]. This framework easily allows for 4 608 different combinations to be tested and compared, as has been done in [12,13].

In this paper we consider the potential of *adaptive structural configurations*. That is, we regard an online selection of the most suitable of the 4 608 configurations. In a nutshell, adaptive structural configurations allow to use at every stage of the optimization process the configuration that is most suitable for it. This way, we can in particular switch from a configuration that performs well during the earlier stages of optimization to one that is more suitable in the later parts. Imagine for example the optimization process of the multi-modal Rastrigin function. The search has to avoid many local optima at first, but effectively only has to solve the Sphere problem once the area of the global optimum has been identified.

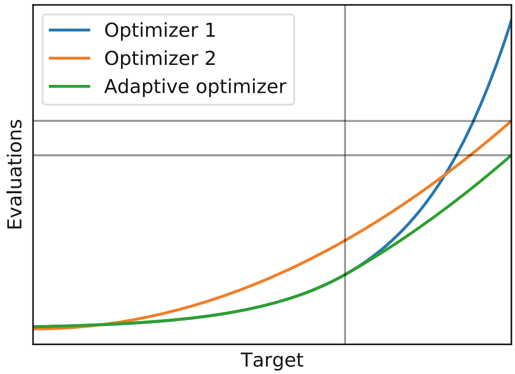

Fig. 1. *Sketch of adaptive configuration.* Sketched performance profiles of two optimizers and an adaptive performance profile based on the profiles of optimizers 1 and 2. The vertical line indicates the transition from using optimizer 1's behavior to that of optimizer 2. Horizontal lines indicate the required budget to reach the final target for optimizer 2 and the adaptive optimizer. The gained improvement is the difference between these two lines.

Specifically we analyze the potential speed-up that can be gained by simulating such configuration switches using the convergence history of 4 608 CMA-ES-based algorithms. Figure 1 illustrates how we can simulate the potential improvement of using adaptive configurations based on existing performance data. In

Sect. 2 we give a short introduction of the modular CMA-ES framework that is used to create the 4 608 CMA-ES configurations. Section 3 defines the procedure applied to calculate the potential speed-up, with Sect. 4 discussing the results. Finally, Sect. 5 concludes this paper and lists some potential avenues for future research.

2 Modular CMA-ES

The *modular CMA-ES framework* [12] has been developed to facilitate the testing and exploration of arbitrary combinations of algorithmic variations of the CMA-ES. In the following, such algorithmic variations are called *configurations*. Each configuration is built-up of eleven different *modules* that can be enabled or disabled independently of each other. As two of the possible modules have three options rather than two, this allows for $2^9 \cdot 3^2 = 4\,608$ different possible configurations. A list of the modules used in this framework is provided in Table 1. For any further implementation details, see [12].

Table 1. *Overview of the available ES modules in the modular CMA-ES framework.* For most of these modules the only available 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 [10]	off	on	-
2	Elitism	(μ, λ)	$(\mu + \lambda)$	-
3	Mirrored Sampling [4]	off	on	-
4	Orthogonal Sampling [14]	off	on	-
5	Sequential Selection [4]	off	on	-
6	Threshold Convergence [11]	off	on	-
7	TPA [6]	off	on	-
8	Pairwise Selection [2]	off	on	-
9	Recombination Weights	$\frac{\log(\mu+\frac{1}{2})-\log(i)}{\sum_j w_j}$	$\frac{1}{\mu}$	-
10	Quasi-Gaussian Sampling [3]	off	Sobol	Halton
11	Increasing Population [1,7]	off	IPOP	BIPOP

3 Data Processing

3.1 Generation and Pre-processing of the Data

We quantify the theoretical potential of adaptive configuration selection for the four functions F1, F10, F15, and F20 from the BBOB benchmark suite [9], and for each of these functions we focus on dimensions 5 and 20. Each function is chosen to represent one of the four subgroups from the BBOB suite. For generating and preprocessing the performance data, the following steps are executed:

Step 1: Generation of runtime data for each of the 4 608 configurations. For each ($F =$ function, $d =$ dimension) pair we collect running time data from 5 independent runs on 5 different instances, resulting in a total of 25 runs. The BBOB test suite stores the function value of a best-so-far search point after every improvement. The allocated budget is $10^4 \cdot d$ per run. Since we collect data for each of the 4 608 configurations, this gives us about 4 GB of running time data for each (F, d) pair.

Step 2: Computation of average hitting times AHT. Following the logic of BBOB, we compute an average hitting time for each of the $\Gamma = 51$ target precisions $\{10^{2.0}, 10^{1.8}, \ldots, 10^{-7.8}, 10^{-8.0}\}$, defined as $\phi_i = 10^{2-(i-1)\cdot 0.2}$, $i \in \{1, 2, \ldots, \Gamma\}$. For every function F, dimension d, configuration C, and target precision ϕ this is done as follows: Let $0 \leq s \leq 25$ be the number of *successful* runs, i.e., the number of runs of configuration C that have reached the target precision ϕ on the (F, d) pair. The first point in time in which the function has been at most ϕ is referred to as the first hitting time. When $s = 25$, i.e., when all runs have been successful, the average hitting time $\text{AHT}(F, d, C, \phi)$ is simply the average of the hitting times. As we will usually fix the function F and dimensionality d, we write $\text{AHT}(C, \phi)$ as a shorthand for $\text{AHT}(F, d, C, \phi)$. When $s < 25$, at least one of the runs was not able to reach ϕ and the AHT is set to ∞. The reasons for this are explained in Sect. 3.3.

Step 3: Computation of *segmented* average hitting time sAHT. We next compute from the $\text{AHT}(C, \phi)$ values an indicator for the performance of the different configurations in the function value *segments* $S_i = (\phi_{i-1}, \phi_i]$. For each $i \in \{1, \ldots, \Gamma\}$ let ϕ_i be the i-th target precision, with $\phi_0 = \infty$ such that Segment $S_1 = (\infty, 10^2)$. This *segmented* average hitting time of configuration C in segment S_i for function, dimension pair (F, d) is defined as

$$\text{sAHT}(C, i) = \text{AHT}(C, \phi_i) - \text{AHT}(C, \phi_{i-1}) \tag{1}$$

i.e., difference in average hitting times. We can assume that $\text{sAHT}(C, i) \geq 0$ since the average hitting time is monotonically increasing by construction:

$$\forall i : \text{AHT}(C, \phi_i) \leq \text{AHT}(C, \phi_{i+1}).$$

3.2 Constructing Optimal Adaptive Configurations

The full convergence behavior of a configuration C towards final target ϕ_Γ can be defined based on the sAHT measure defined above:

$$\text{AHT}(C, \phi_\Gamma) = \sum_{i=1}^{\Gamma} \text{sAHT}(C, i). \tag{2}$$

Maximally Adaptive. We can adapt Eq. (2) to replace any static configuration C by a sequence of configurations $\mathbf{C} = \{C_1, C_2, \ldots, C_\Gamma\}$:

$$\mathrm{AHT}(\mathbf{C}, \phi_\Gamma) = \sum_{i=1}^{\Gamma} \mathrm{sAHT}(C_i, i). \tag{3}$$

Under the assumption that the performance of C_i on section S_i is independent of C_i's behavior up to target ϕ_{i-1}, we can then pick this sequence $\mathbf{C}$ such that

$$\mathbf{C}_{\mathrm{opt}} = \{C_i \mid i \in \{1, \ldots, \Gamma\}, C_i = \arg\min_C\, (\mathrm{sAHT}(C, i))\} \tag{4}$$

then we can guarantee by construction that

$$\mathrm{AHT}(\mathbf{C}_{\mathrm{opt}}, \phi_\Gamma) \leq \mathrm{AHT}(\mathbf{C}, \phi_\Gamma) \tag{5}$$

for any sequence $\mathbf{C}$ of configurations.

In other words, this means that we consider $\mathbf{C}_{\mathrm{opt}}$ to be an algorithm that chooses its internal configuration for each segment S_i to be precisely the configuration C_i that has the smallest sAHT for that segment.

Single Split. Naturally, this maximally adaptive algorithm is practically quite infeasible for various reasons. However, by restricting our choice of $\mathbf{C}$ we can create a more feasible alternative that still outperforms any single original configuration C.

Let $\mathbf{C}$ be a sequence $(C_1, C_2, \ldots, C_\Gamma)$ such that: $C_i = C_1\ \forall i \in \{1, \ldots, s\}$ and $C_i = C_\Gamma\ \forall i \in \{s+1, \ldots, \Gamma\}$, where $1 < s < \Gamma$ is the *split* index. This represents an adaptive approach where a single configuration switch occurs during the runtime, namely when reaching ϕ_s. For a given split s, we can then find C_1 and C_Γ that minimize the AHT for their respective sequence of segments.

By repeating this for all possible splits, an optimal split s with corresponding configurations C_1 and C_Γ can be computed from the dataset. By appropriating the argmin notation, we can write this as follows:

$$(C_1, C_\Gamma, s) = \arg\min_{C_1, C_\Gamma, s} \left(\sum_{i=1}^{s} \mathrm{sAHT}(C_1, \phi_i) + \sum_{i=s+1}^{\Gamma} \mathrm{sAHT}(C_\Gamma, \phi_i) \right) \tag{6}$$

or in terms of AHT:

$$(C_1, C_\Gamma, s) = \arg\min_{C_1, C_\Gamma, s} (\mathrm{AHT}(C_1, \phi_s) + (\mathrm{AHT}(C_\Gamma, \phi_\Gamma) - \mathrm{AHT}(C_\Gamma, \phi_s))) \tag{7}$$

In words: we consider an adaptive algorithm that is split at the end of segment S_i, such that configuration C_1 performs best for $S_1, \ldots, S_i$ and C_Γ performs best for $S_{i+1}, \ldots, S_\Gamma$ and their combined AHT is minimal again. In the worst case scenario, $C_1 = C_\Gamma$ and we do not get any improvement.

3.3 Discarding Partially Successful Configurations

As mentioned in Sect. 3.1, we deliberately choose to ignore any configuration that was not successful in *all* 25 runs for the desired target value ϕ. This absolves us of the associated uncertainty at the cost of reducing the effective size of our dataset. Otherwise, we would run into problems with the previously defined (s)AHT measure using the two most likely options of dealing with unsuccessful runs: ignoring and penalizing.

Ignoring Unsuccessful Runs. If we were to ignore missing values caused by unsuccessful runs and simply average over all available individual hitting times, we can easily end up with negative sAHT values, as shown in Fig. 2.

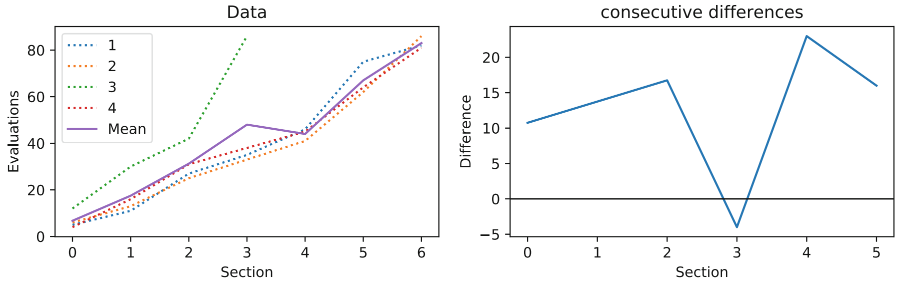

Fig. 2. *Example of non-monotonicity causing negative sAHT values.* The original, monotonically increasing data, consists of four runs, one of which is not successful for targets 4 and up. If an average is taken of all valid data points, the unsuccessful run increases the average for targets 0–3, but the average drops once the unsuccessful run is no longer taken into account. The difference, or sAHT, between targets 3 and 4 becomes negative in this case.

ERT/Simulated AHT. Methods such as *simulated AHT* or *Estimated Running Time (ERT)* as defined in [8] have been introduced to incorporate these unsuccessful runs into the AHT. These methods artificially increase the hitting time by substituting the unsuccessful runs with (multiples of) the maximum evaluation budget to account for the uncertainty of success. Although this is very useful when considering a configuration's performance from start to finish, the penalty can be ignored when we only consider the performance between given targets.

Figure 3 shows how this can happen. In this figure, two otherwise equal optimizers 1 and 2 are penalized for an unsuccessful run, each after reaching a different intermediate target. When applying the construction of Eq. (6), the optimal point to switch will be at the vertical line, i.e. *before* optimizer 1 is penalized, but *after* optimizer 2 is. This way, the penalty is completely avoided by the adaptive configuration sequence. The resulting AHT then indicates that a large performance improvement is possible, while this not the case.

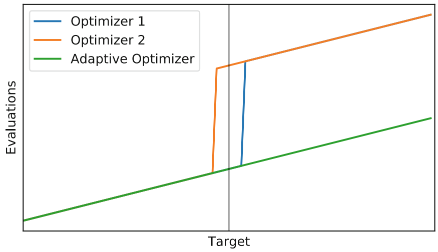

Fig. 3. *Adaptive optimizer omits penalties for unsuccessful runs.* This figure shows a sketch of two optimizers that both incur a penalty for having unsuccessful runs, shown by a jump in the number of evaluations. The vertical line indicates a single configuration switch. As the penalties occur at either side of the switch, the adaptive optimizer avoids the penalty.

4 Results

4.1 Maximally Adaptive

Results for the maximally adaptive strategy are listed in Table 2, where the relative improvement is calculated as 1 − Adaptive/Static. *Static* refers to a non-adaptive configuration $\mathbf{C} = (C_1, C_2, \ldots, C_\Gamma)$ where $C_1 = C_2 = \ldots = C_\Gamma$. They show improvements ranging from 11 up to 47%. It must be noted, however, that, given the budget of $10^4 \cdot d$, 5d F20, 20d F15 and 20d F20 have not reached the final target of 10^{-8}. The listed results are only up to the listed target ϕ.

Although the results are impressive, they seem unreliable. After all, CMA-ES performs rather well on the sphere function as-is, so an expected speed-up of 47% for 5d F1 is unrealistic. The explanation for this lies in the fact that CMA-ES are still stochastic processes. If we have a number of configurations that perform similarly over the entire runtime, some local variance is to be expected. As the size of each segment S_i decreases, we end up cherry-picking sAHT values that are small by chance rather than because of inherent information they contain.

Having said this, these results still provide an upper bound for the improvement that may be obtained by making better use of the available structural configuration space.

4.2 Single Split

Table 3 lists the results when only considering a single configuration switch. Potential improvement ranges from 3 up to 22% in this case. The reached targets are the same as for the maximally adaptive method. Convergence behavior of these results is shown in Figs. 4 and 5.

For the easiest F1 functions, these improvements are much smaller than in the maximally adaptive setting: 6.9% versus 47.1% and 3% versus 16.6% for 5d and 20d F1, respectively. This much lower improvement combined with the convergence behavior as shown in Fig. 4 supports the explanation that these

Table 2. *Results of only successful configurations.* The *Static* and *Adaptive* columns indicate the AHT values. Column ϕ lists the smallest target values for which the shown AHT values were obtained. Relative improvement r is calculated as 1−Adaptive/Static.

d	F	ϕ	Static	Adaptive	r	Static C
5	1	$10^{-8.0}$	412.00	218.05	0.471	00 110 011 010
5	10	$10^{-8.0}$	1437.08	832.00	0.421	11 000 110 022
5	15	$10^{-8.0}$	14812.72	9220.80	0.378	00 110 011 011
5	20	$10^{-0.2}$	14535.16	10951.92	0.247	01 010 101 022
20	1	$10^{-8.0}$	1269.05	1058.45	0.166	00 110 110 021
20	10	$10^{-8.0}$	16565.48	11135.00	0.328	00 001 000 010
20	15	$10^{0.6}$	45279.08	26249.16	0.420	00 111 000 001
20	20	$10^{0.2}$	14476.76	12829.40	0.114	00 010 001 022

improvements for F1 are most likely due to random variance. When comparing the representations for 20d F1 for example, we can see that the best normal configuration is the same as that for the second part of the optimization process, and that the configuration for the first part only differs in the final module: IPOP instead of BIPOP, which should not make difference when optimizing the sphere function F1.

For more difficult functions such as 5d F10 and F15 however, the single split method already shows a large potential improvement of over a third to a half of the upper bound established by the maximally adaptive method. The improvement is clearly visible in the convergence behavior. The convergence of the first part follows that of a rather successful configuration, which does not perform best overall, while for the second part, the behavior of a different configuration is followed. This second configuration may take longer to exhibit this beneficial search behavior, but because of the switch, we can disregard this initial delay and use its good performance in the latter half of the optimization process to (significantly) beat the best static configuration.

On the other hand, the results for 5d F20, 20d F15 and 20d F20 are not very informative as most target values are not reached. For these functions, even higher budgets are required for CMA-ES-based optimizers to reach the target value of 10^{-8}.

4.3 Discussion

As already noted in Sect. 4.1, because the data has been created by a stochastic process, measurement uncertainty has to be taken into account. Although we have tried to reduce its influence by using an average over 25 runs and discarding any AHT values for which not all 25 were successful, some variance remains. This is very visible in the analysis of F1 (sphere function).

The results of 5d F10 (see Fig. 4) shows a very clear knee-point at the split, after which all three original configurations start performing better, i.e. needing

Table 3. *Results of only successful configurations.* The *Static* and *Adaptive* columns indicate the AHT values. Relative improvement r is calculated as $1 - \text{Adaptive}/\text{Static}$. *Split* indicates the target value ϕ_i after which we switch from one configuration to the other. The final three columns and C_Γ show the representation for the static best configuration and the configurations C_1, C_Γ that make up the first and second part of the adaptive approach, respectively. This representation can be decoded using Table 1.

d	F	Static	Adaptive	r	Split	Static C	C_1	C_Γ
5	1	412.00	383.70	0.069	$10^{-2.2}$	00 110 011 010	10 110 111 020	00 110 011 011
5	10	1437.08	1207.12	0.160	$10^{0.2}$	11 000 110 022	01 001 110 120	01 101 000 012
5	15	14812.72	11524.24	0.222	$10^{0.4}$	00 110 011 011	00 110 000 001	00 001 011 011
5	20	14535.16	11628.36	0.200	$10^{0.0}$	01 010 101 022	01 110 011 011	00 100 001 021
20	1	1269.05	1231.40	0.030	$10^{-3.8}$	00 110 110 021	00 110 110 022	00 110 110 021
20	10	16565.48	15003.72	0.094	$10^{1.8}$	00 001 000 010	00 011 000 011	00 001 101 011
20	15	45279.08	42020.80	0.072	$10^{0.8}$	00 111 000 001	00 111 000 001	00 101 000 011
20	20	14476.76	12942.96	0.106	$10^{0.4}$	00 010 001 022	11 100 111 001	00 010 001 022

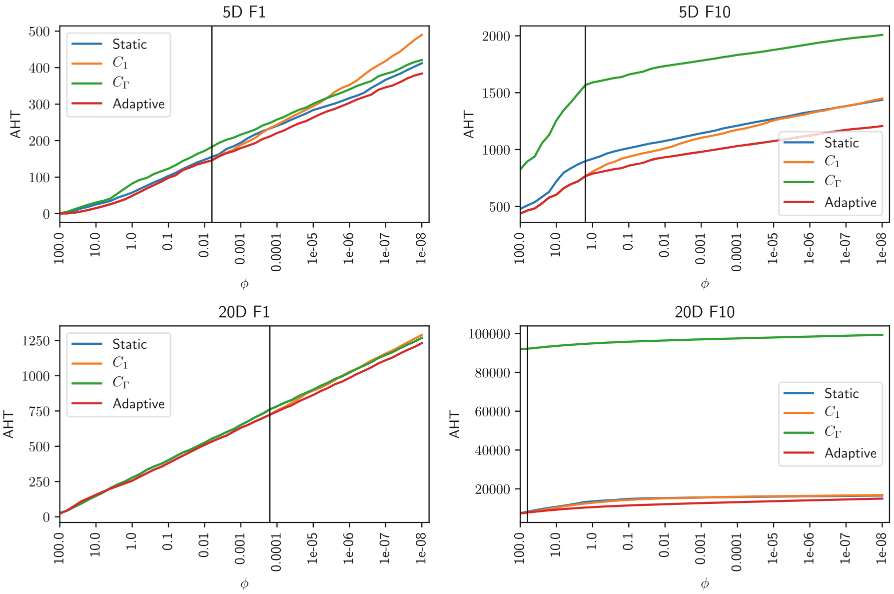

Fig. 4. *Convergence behavior of configurations listed in Table 3.* These plots show $\text{AHT}(F, d, C, \phi)$, the average number of evaluations required for a configuration C to reach a target value ϕ. The vertical line indicates the location of the optimal *split*. The convergence labeled *Adaptive* consist of the behavior of configuration C_1 before the split, and the behavior of configuration C_Γ afterwards. If none of the configurations were successful in all 25 runs, there is no data for those convergence targets ϕ.

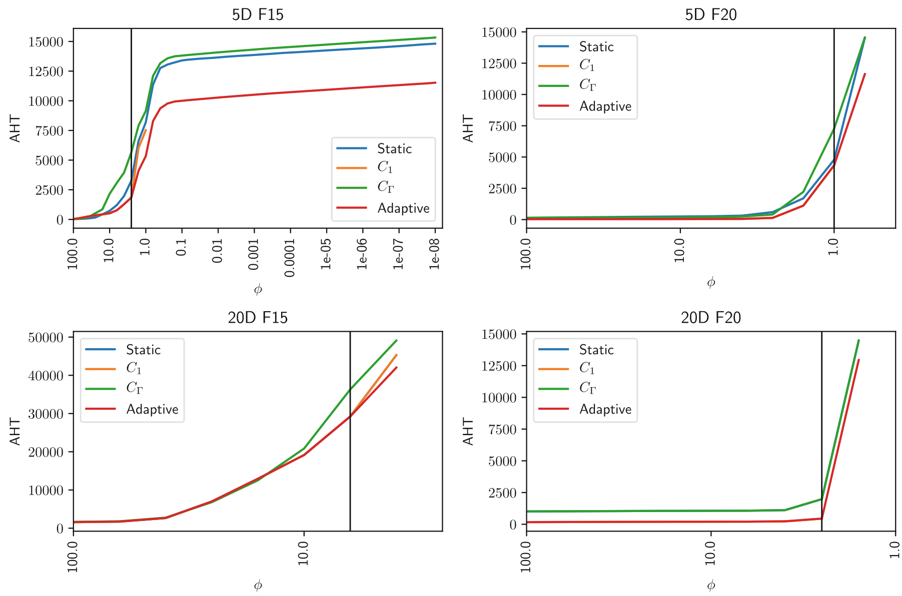

Fig. 5. *Convergence behavior of configurations listed in Table* 3. These plots show AHT(F, d, C, ϕ), the average number of evaluations required for a configuration C to reach a target value ϕ. The vertical line indicates the location of the optimal *split*. The convergence labeled *Adaptive* consist of the behavior of configuration C_1 before the split, and the behavior of configuration C_Γ afterwards. If none of the configurations were successful in all 25 runs, there is no data for those convergence targets ϕ.

fewer evaluations to reach the next targets. This change in performance is not the same however, which is exploited by the adaptive configuration, confirming the intuition motivating this research.

A practical caveat, however, is that a fast but less successful configuration such as 'part 1' in 5d F15 as seen in Fig. 5 may reach its best target value ϕ by exploiting a local optimum that is not necessarily close to the global optimum. In such cases, continuing the search with a different configuration will *not* result in the speed-up demonstrated in this paper, but rather in a slow-down as the algorithm would have to escape the local optimum, if possible at all, before finding the global optimum on its own.

5 Conclusion and Future Work

In this paper we have shown that it is possible to empirically determine upper bounds for the possible speed-up when considering structurally adaptive CMA-ES-based optimization algorithms. The results support the idea that improvements of 5–20% are already viable when switching algorithm configuration only once during the optimization process. We hope this research will inspire further investigation into online structural adaptation of optimization algorithms.

However, the assumption that is required for the presented interpretation (see before Eq. (4)) is non-trivial. As the internal state of CMA-ES-based optimizers is highly dependent on the search history, there is no guarantee that the performance will correctly continue after an actual configuration switch. An important next step is then to run some of the identified adaptive configurations to evaluate their actual performance. This should be done on white-box versions of the used benchmark functions so the switch can be performed at the identified optimal intermediate target. In this paper, we intentionally focused on the data-driven analysis, and investigating the above-mentioned setting will be the next step towards understanding and exploiting adaptive configurations.

Additionally, the process described in this paper can also be further improved with statistical analysis of the (s)AHT values that are used to determine hypothetical performance of the adaptive configurations. E.g. for the sphere function F1, such analysis will give an indication of whether the improvement is due to random variance in the data or due to actual differences in convergence behavior.

Acknowledgements. The authors would like to thank Hao Wang for his participation in the discussions leading up to this work.

References

1. Auger, A., Hansen, N.: A restart CMA evolution strategy with increasing population size. In: 2005 IEEE Congress on Evolutionary Computation, vol. 2, pp. 1769–1776, September 2005. https://doi.org/10.1109/CEC.2005.1554902
2. Auger, A., Brockhoff, D., Hansen, N.: Mirrored sampling in evolution strategies with weighted recombination. In: Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation, GECCO 2011, pp. 861–868. ACM, New York (2011). https://doi.org/10.1145/2001576.2001694
3. Auger, A., Jebalia, M., Teytaud, O.: Algorithms (X, sigma, eta): quasi-random mutations for evolution strategies. In: Talbi, E.-G., Liardet, P., Collet, P., Lutton, E., Schoenauer, M. (eds.) EA 2005. LNCS, vol. 3871, pp. 296–307. Springer, Heidelberg (2006). https://doi.org/10.1007/11740698_26
4. Brockhoff, D., Auger, A., Hansen, N., Arnold, D.V., Hohm, T.: Mirrored sampling and sequential selection for evolution strategies. In: Schaefer, R., Cotta, C., Kołodziej, J., Rudolph, G. (eds.) PPSN 2010. LNCS, vol. 6238, pp. 11–21. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15844-5_2
5. Hansen, N., Ostermeier, A.: Adapting arbitrary normal mutation distributions in evolution strategies: the covariance matrix adaptation. In: Proceedings of IEEE International Conference on Evolutionary Computation, pp. 312–317, May 1996. https://doi.org/10.1109/ICEC.1996.542381
6. Hansen, N.: CMA-ES with Two-Point Step-Size Adaptation. arXiv:0805.0231 [cs], May 2008
7. Hansen, N.: 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 2009, pp. 2389–2396. ACM, New York (2009). https://doi.org/10.1145/1570256.1570333, http://doi.acm.org/10.1145/1570256.1570333

8. Hansen, N., Auger, A., Brockhoff, D., Tuar, D., Tuar, T.: COCO: Performance Assessment. arXiv:1605.03560 [cs], May 2016
9. Hansen, N., Auger, A., Finck, S., Ros, R.: Real-parameter black-box optimization benchmarking 2009: experimental setup. Report, INRIA (2009). https://hal.inria.fr/inria-00362649/document
10. Jastrebski, G.A., Arnold, D.V.: Improving evolution strategies through active covariance matrix adaptation. In: 2006 IEEE International Conference on Evolutionary Computation, pp. 2814–2821 (2006). https://doi.org/10.1109/CEC.2006.1688662
11. Piad-Morffis, A., Estvez-Velarde, S., Boluf-Rhler, A., Montgomery, J., Chen, S.: Evolution strategies with thresheld convergence. In: 2015 IEEE Congress on Evolutionary Computation (CEC), pp. 2097–2104, May 2015. https://doi.org/10.1109/CEC.2015.7257143
12. van Rijn, S., Wang, H., van Leeuwen, M., Bäck, T.: Evolving the structure of evolution strategies. In: 2016 IEEE Symposium Series on Computational Intelligence (SSCI), pp. 1–8, December 2016. https://doi.org/10.1109/SSCI.2016.7850138
13. van Rijn, S., Wang, H., van Stein, B., Bäck, T.: Algorithm configuration data mining for CMA evolution strategies. In: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2017, pp. 737–744. ACM, New York (2017). https://doi.org/10.1145/3071178.3071205, http://doi.acm.org/10.1145/3071178.3071205
14. Wang, H., Emmerich, M., Bäck, T.: Mirrored orthogonal sampling with pairwise selection in evolution strategies. In: Proceedings of the 29th Annual ACM Symposium on Applied Computing, SAC 2014, pp. 154–156. ACM, New York (2014). https://doi.org/10.1145/2554850.2555089, http://doi.acm.org/10.1145/2554850.2555089