HAL
open science

Online selection of CMA-ES variants

Diederick Vermetten, Sander van Rijn, Thomas Back, Carola Doerr

To cite this version:

Diederick Vermetten, Sander van Rijn, Thomas Back, Carola Doerr. Online selection of CMA-ES variants. The Genetic and Evolutionary Computation Conference, Jul 2019, Prague, Czech Republic. pp.951-959, 10.1145/3321707.3321803 . hal-02175767

HAL Id: hal-02175767

https://hal.sorbonne-universite.fr/hal-02175767

Submitted on 14 Jan 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Online Selection of CMA-ES Variants

Diederick Vermetten
Leiden Institute for Advanced Computer Science
Leiden, The Netherlands

Sander van Rijn
Leiden Institute for Advanced Computer Science
Leiden, The Netherlands

Thomas Bäck
Leiden Institute for Advanced Computer Science
Leiden, The Netherlands

Carola Doerr
Sorbonne University, LIP6, and CNRS
Paris, France

ABSTRACT

In the field of evolutionary computation, one of the most challenging topics is algorithm selection. Knowing which heuristics to use for which optimization problem is key to obtaining high-quality solutions. We aim to extend this research topic by taking a first step towards a selection method for adaptive CMA-ES algorithms. We build upon the theoretical work done by van Rijn *et al.* [PPSN'18], in which the potential of switching between different CMA-ES variants was quantified in the context of a modular CMA-ES framework.

We demonstrate in this work that their proposed approach is not very reliable, in that implementing the suggested adaptive configurations does not yield the predicted performance gains. We propose a revised approach, which results in a more robust fit between predicted and actual performance. The adaptive CMA-ES approach obtains performance gains on 18 out of 24 tested functions of the BBOB benchmark, with stable advantages of up to 23%. An analysis of module activation indicates which modules are most crucial for the different phases of optimizing each of the 24 benchmark problems. The module activation also suggests that additional gains are possible when including the (B)IPOP modules, which we have excluded for this present work.

CCS CONCEPTS

• **Theory of computation → Evolutionary algorithms**; *Bio-inspired optimization*; *Online algorithms*; Algorithm design techniques.

KEYWORDS

Evolution Strategies, Algorithm Configuration, Algorithm Selection

1 INTRODUCTION

The creation of optimization algorithms has long been a topic of study in Mathematics and Computer Science. As the complexity of optimization problems increased, the need for fast heuristic algorithms became clear. Techniques like evolution strategies have been around for quite some time, and much research has been done into improving them, such as the popular Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [17].

For CMA-ES, many adaptations have been developed over the last decade, all with the goal of creating a better optimization algorithm for certain types of problems. However, with all these different options to choose from, it can be difficult to know which version to use. The comparison of different CMA-ES variants consistently shows that there is no single best variant [4, 16], giving rise to the question how to best choose which of the variants to use for which kind of optimization problems. In addition, all CMA-ES variants have several hyper-parameters, whose values can have considerable impact on their performance. This means that besides the *algorithm selection problem* we also face an *algorithm configuration problem*. These two problems are highly interlinked. The choice of algorithm and corresponding parameter configuration for a given problem at hand thereby forms itself a new optimization problem, which has gained increasing attention in the last decades. With the increased accuracy of performance prediction models, state-of-the-art research in evolutionary computation focuses on *automating* the decision of which algorithm and which parameter configuration to select. Among the most widely used prediction models are Gaussian processes [6, 7], but also techniques from the machine learning community, such as random forests [19, 22]. Surveys on algorithm selection are available in [18, 21, 23].

While algorithm selection generally considers *static* choices of algorithms, we aim to extend existing work to allow for *adaptive* changes of the algorithm configuration. Note here that *on-the-fly* algorithm selection and configurations are also studied under the umbrella term *hyper-heuristics* [10, 11]. To date, however, research on hyper-heuristics mostly focuses on discrete optimization and much less on continuous optimization. In particular, we are not aware of any previous approaches to address a dynamic selection of CMA-ES module variants.

We base our work on the modular CMA-ES framework introduced in [27]. Within this framework, we can create many variants of CMA-ES by turning on or off certain modules, e.g. elitism, orthogonal sampling, and different weighting schemes (cf. Table 1). Each module combination gives a different *configuration* of the modular CMA-ES.

Using this modular CMA-ES framework, a theoretical analysis of the potential benefits of adaptive CMA-ES configurations has been provided in [26]. In this work, we will extend their study to all 24 BBOB functions [14], quantifying the actual performance gains, identify a need for a more reliable prediction, and propose an alternative method, which shows a much better fit between predicted performances and actual achieved running times.

We focus on the case in which the CMA-ES variant can be changed only once. Our main results show that even such single switches allow for performance gains on 18 out of the 24 studied BBOB functions. In 11 cases the relative gain against the best static configurations are larger than 5%, 8 functions show a relative gain $\geq$ 10% and the largest observed improvement (for F21) is 51%. However, the 51% is achieved for one particular configuration swap only, and when averaging the 10 best adaptive configuration pairs, the relative gain over the best static variant vanishes. We nevertheless see other functions with stable relative gains. The largest stable gains of around 20% are obtained for F5 and F6.

All numbers reported above are with respect to the default parameter-choices. That is, we focus in this work exclusively on the algorithms selection aspect, thereby leaving the question of configuring the default hyper-parameters for future work. Additional performance gains can be expected from such parameter tuning, as, for example, the results presented in [8] indicate.

Investigating the best-performing modules per each stage of the optimization process, we identify which configurations are most efficient for which parts of the optimization process.

Structure of the paper: Section 2 summarizes the modular framework, in which we build our adaptive CMA-ES. We introduce our implementation of the approach used in [26] in Section 3, followed by a discussion of our additional techniques for configuration selection. Section 4 shows that the original approach is not very stable. In Section 5 we discuss the results from our experiment, with a focus on the configurations and switching points at which the modules are changed. We conclude our contribution in Section 6 by looking at possible directions for future work.

2 THE MODULAR CMA-ES FRAMEWORK

The adaptive CMA-ES configurations we use were implemented in the modular CMA-ES framework introduced in [27], which is freely available at [25].

This framework implements 11 different modules. Of these 11 modules, 9 are binary and 2 are ternary, allowing for a combined total of 4,608 different configurations. The full list of available modules is shown in Table 1.

To clarify the terminology, note that we follow the example given in [26] and consider each combination of the 11 modules a "configuration". As mentioned in the introduction, we do not, in this work, change or optimize the hyper-parameters of the CMA-ES variants (such as the population size, the speed of the step size adaptation, etc.). We use the default values for these parameters instead.

For our benchmarking, we use the BBOB-suite of benchmark functions [15]. We used all noiseless, five-dimensional functions. Each BBOB "function" is a set of functions (commonly referred to as "instances") with similar fitness landscapes. In accordance with most comparative studies within the BBOB framework, we study the first five instances. For each of these instances, we run every configuration five times. This results in 2,649,600 total runs. The budget for these runs are set at $10^4 \cdot D$.

While this gives us data for a total of 4,608 different configurations, we ended up only using only one third of the configuration space in our adaptive experiments. This is due to the fact that switching between configurations with a different setting for increasing population ((B)IPOP) would possibly create an information deficit, in that we do not know what population size we should have at the moment of switching. To eliminate this uncertainty, we decided not to consider (B)IPOP in our adaptive experiments, i.e., we dropped two options out of three. This leaves us with 1,536 configurations to consider.

#	Module name	0	1	2
1	Active Update [20]	off	on	-
2	Elitism	(μ, λ)	$(\mu+\lambda)$	-
3	Mirrored Sampling [9]	off	on	-
4	Orthogonal Sampling [29]	off	on	-
5	Sequential Selection [9]	off	on	-
6	Threshold Convergence [24]	off	on	-
7	TPA [12]	off	on	-
8	Pairwise Selection [1]	off	on	-
9	Recombination Weights [3]	$\log(\mu+\frac{1}{2})-\frac{\log(i)}{\sum_j w_j}$	$\frac{1}{\mu}$	-
10	Quasi-Gaussian Sampling	off	Sobol	Halton
11	Increasing Population [2, 13]	off	IPOP	BIPOP

Table 1: Overview of the CMA-ES modules available in the used framework. The entries in row 9 specify the formula for calculating each weight w_i.

3 ADAPTIVE CONFIGURATIONS

Our main research question in this work is the following. For each of the 24 BBOB functions f we aim to estimate from the detailed runtime data of the 1,536 static configurations the triples (C_1, C_2, τ) that exhibit the best performance for optimizing f. Each triple is interpreted as follows. The first configuration, C_1, is run until hitting target value ("splitpoint") τ. Immediately after the first iteration in which a solution of fitness $\leq \tau$ is sampled, we switch to configuration C_2 and run this configuration until the target 10^{-8} is reached or the budget exhausted.

The long term vision of our research is to extend our study to a landscape-aware configuration. To this end, we will combine our data with explanatory landscape analysis, and to build an automated tool to select CMA-ES configurations *on the fly*. Put differently, we aim at extending the *per instance algorithm configuration* approach analyzed in [8] to the modular CMA-ES and towards a non-static selection. Rather than absolute performance gains, our main interests is therefore in identifying the best performing configurations per each function and each phase of the optimization process.

3.1 Performance Measures

The hitting time for a certain target is defined as follows:

Definition 3.1 (Hitting Time). The *hitting time* of a target ϕ, a configuration c and a run i on a function f (written as $HT(f, c, \phi, i)$) is defined as the first function evaluation during run i of c in which a function value with a difference of at most ϕ to the global optimum is observed.

In this paper, we use two different performance measures. The first is the average hitting time (AHT), which is based on the previously defined hitting time as follows:

Definition 3.2 (AHT). The average hitting time of a configuration c for a target ϕ on function f is defined as:

$$AHT(f,c,\phi) = \begin{cases} \infty & \text{IF } \exists i : HT(f,c,\phi,i) = \infty \\ \frac{\sum_i HT(f,c,\phi,i)}{\sum_i 1} & \text{OTHERWISE} \end{cases}$$

This method places an infinite penalty on non-finished runs. This can be beneficial when dealing with relatively few configurations to avoid selecting those which are unreliable. However, when dealing with more difficult functions and more runs, we want to deal with these non-finished runs differently. This can be done using the expected running time (ERT), which is defined as follows:

Definition 3.3 (ERT). The expected running time (ERT) of a configuration c and target ϕ on a function f is calculated as follows:

$$ERT(f,c,\phi) = \frac{\sum_i (F(f,c,\phi,i) \cdot b + HT(f,c,\phi,i))}{\sum_i (1 - F(f,c,\phi,i))},$$

where b is the *budget* of each run (i.e., the maximum number of evaluations) and $F(f,c,\phi,i) = 1$ if run i on function f failed to reach the target ϕ, while $F(f,c,\phi,i) = 0$ otherwise.

Note that if each run of a configuration c on function f found a solution x with $f(x) - f_{opt} \leq \phi$, then $ERT(f,c,\phi) = AHT(f,c,\phi)$.

3.2 Selecting Adaptive Configurations

We gather the AHTs for targets $\Phi = \{10^{2-(0.2 \cdot i)} \mid i \in \{0 \ldots 50\}\}$. Based on these AHT values, the adaptive configurations suggested in [26] are chosen as follows:

- For each configuration c, each of the 24 BBOB functions f, and each of the 51 target values ϕ, we calculate the AHT over all 25 runs (5 runs for each of the first five instances).
- From this data, we determine the best target value ϕ_{min} for which there exists at least one configuration whose 25 runs all reached this target.
- For every target value $\phi \in \Phi$ satisfying $\phi > \phi_{\text{min}}$ we calculate the best configuration before this target, i.e., we select the configuration c for which $AHT(c,\phi)$ is minimized. We denote this configuration C_1. We then compute the best configuration c from this target until ϕ_{min}, which we denote as C_2, i.e., C_2 is the configuration for which $AHT(f,c,\phi_{\text{min}}) - AHT(f,c,\phi_{\text{min}})$ is minimized. In [26], the theoretical performance (TH for 'theoretical hitting time') is then calculated as $TH(f,C_1,C_2,\phi) = AHT(f,C_1,\phi) - AHT(f,C_2,\phi) + AHT(f,C_2,\phi_{\text{min}})$.
- From this data we compute the target value τ for which the overall performance $TH(f,C_1,C_2,\phi)$ is minimized. This gives us the adaptive configuration (C_1,C_2,τ). We refer to τ as the 'splitpoint' of the adaptive configuration.

We illustrate this approach in Fig. 1. In this figure we can clearly see that following C_1 until the splitpoint and then switching to C_2 gives an adaptive configuration which reaches the target with less evaluations than the best static configuration.

3.3 Mitigating Uncertainty in the Selection

Initial experiments showed a poor match between the predicted and achieved performance of the adaptive configurations as selected by the previously described procedure. A relatively large variance among the performance profiles of the original runs was identified as cause for this. To select good adaptive configurations despite this variance in the original data, we use the following two techniques:

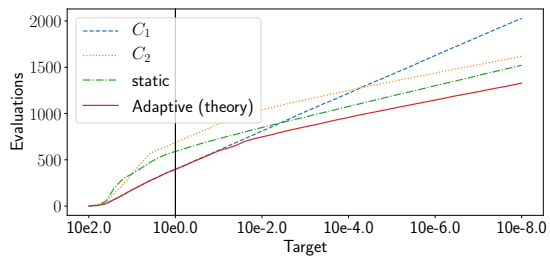

Figure 1: Example of a theoretical adaptive configuration performing better than a static configuration (on F5).

Sliding window: Instead of considering only the performance at a single target and choosing that as the splitpoint to use, we consider a sliding window around the target, and take the average of the performances at all these targets. This should deal with variance by smoothing out the hitting times. Using this method and the AHT, the value we assign to a splitpoint and configuration pair is defined as follows:

Definition 3.4 (Sliding window value). For a function f, adaptive configuration (C_1,C_2,ϕ_i) and using AHT as a performance measure, we define the sliding window value SWV as follows:

$$SWV(f,C_1,C_2,\phi_i) = \begin{cases} \infty & \text{IF } i < w \text{ OR } (50-i) < w \\ \sum_{j=i-w}^{i+w} HT(f,\phi_j,C_1,C_2) & \text{OTHERWISE} \end{cases}$$

Here, w determines the radius of the window, in our case we use $w = 2$ for a window of size $2w + 1 = 5$.

Worst Case: Instead of considering the mean performance, we take a version of the worst-case performance. To still deal with the different instances, we take the worst-case performance for every instance and average these to get a performance value. This should reduce the impact of the variance, since even when we assume to get bad runs, this configuration performs the best out of all possible adaptive configurations. This way, the variance might improve the performance of the adaptive configuration.

We run our tests with 3 different settings: sliding window using means, using worstcase, and the original method.

3.4 Two-Stage Configuration Selection

In addition to these two techniques, we introduce an entirely new procedure to more robustly select which adaptive configurations to run. This is based on the finding that the static configurations are not quite stable enough to use as a baseline. The first step in this process consists of selecting some static configurations for which we should gather more data. The configurations we will consider are made up of two parts. The first part consists of the 50 best performing static configurations.[1] We then extend this set by looking at the configurations which have been selected to be a part of the 50 theoretically best adaptive configurations. Since this

[1]The best static configurations are determined by their AHT at the final reached target. If fewer than 50 configurations reached this target for a function, we extend these configurations by the ones that have the lowest AHT for the previous target. We repeat this process until we have selected 50 configurations.

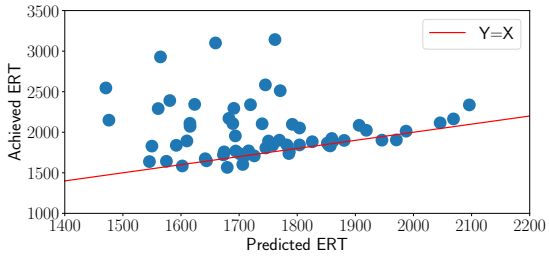

Figure 2: ERT of the selected static configurations for F10. The predicted values are based on the original 5×5 runs, while the achieved ones are on 5×50 runs.

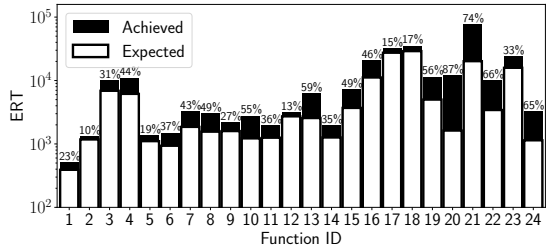

Figure 3: Comparison of the expected ERTs (logscale) for the adaptive configurations against the achieved values. Percentage differences are shown as well.

might not be a diverse set of configurations, as one configuration might be chosen as C_1 50 times, we decide to limit the amount of times a certain configuration can be selected as C_1 and as C_2 to three times each ('limited selection method'). This should give us a more diverse set of configurations which might contribute to good adaptive configurations. We then rerun these configurations using 50 runs on each of the 5 instances, for a total of 250 runs each.

In total, this set of configurations will contain anywhere from 50 to 150 configurations to rerun. We can then compare the ERT of the reruns to the original data. This is done for F10 in Fig. 2, from which we can see that this fit is indeed not very good.

Based on the new, more robust data set, we can now determine the best adaptive configurations to run. This is done using the means and a sliding window method. We still use a sliding window, but instead of taking the worst-case runs for every instance, we revert back to using the AHT. This is done because we have a lot more data available, and the mean should be more stable. We still use the same limited selection method as described previously.

4 INITIAL EXPERIMENTS

As previously mentioned, our first aim was to extend the work presented in [26]. We recall that in [26] only 4 selected functions have been considered. We extend this to all 24 BBOB-functions. We then run the adaptive configurations suggested by this approach, and observe that the fit between the theoretical performance prediction and the actual running times is not very stable.

Using the approach mentioned in the Section 3.2, we can calculate the theoretical improvements when using adaptive configurations. These results are not shown here, but are available in [28]. Note that our results differ slightly from those found in the original paper [26]. The reason for this is twofold: First, the configuration space we consider is more restricted since we do not use the (B)IPOP module. Second, we added another restriction on which configurations to use in the adaptive configurations. We require both C_1 and C_2 to reach the target $\phi_{\min}$. This is done to avoid situations where a C_1 is selected which very quickly converges to a local optimum and gets stuck there. In practice, switching to C_2 would then still result in being stuck in this local optimum, which would not be a benefit to finding the global optimum.

Overall, we found expected improvements to range from 0 to 15%. The 0% improvements are caused by the fact that for some functions, only one or two configurations manage to reach the selected target value. So in these cases, we have $C_1 = C_2$, leading to the same performance for the adaptive and static configuration. This is present in 4 out of the 24 functions, while all others do have some level of expected improvement.

When running the adaptive configurations,[2] we decided on running them on the same 5 instances as the original data, but instead of the 5 runs per instance which we used to gather the data for the static configurations, we run them 50 times per instance. This should give us more robust results to work with.

The resulting ERTs are compared to the ones we expected in Fig. 3. This comparison shows a clear mismatch between the actually achieved and the expected ERT values. This indicates that the original approach suggested in [26] and summarized in Section 3.2 is not very reliable.

5 RESULTS

We next show how the ideas presented in Sections 3.3 and 3.4 yield more robust results.

In a first step, we ran the adaptive configurations determined by the additional methods described in Section 3.3. The results from running these adaptive configurations showed that these ideas alone are not sufficient to obtain reliable performance predictions. More detailed information about these experiments can be found in [28]. We summarize the main findings.

In total, we manage to see improvements (relative to the rerun static configuration) on 10, 8 and 13 functions for the original, sliding window with AHT, and sliding window using worst-case methods, respectively. That is, the sliding window using the worst-case approach performs slightly better than the other two methods we tested. We expect this to be caused by the fact that this method is less prone to major decreases in performance as a result of variance.

Even though we do see improvements for several functions, all methods have significant outliers, both in positive and negative sense. This seems to be caused by the fact that the ERT of the static configurations is not robust. As shown in Fig. 2, the ERT on the 5×50 runs can deviate significantly from the original ERT calculated based on 5×5 runs. And since the static configurations do not have robust ERTs, the adaptive configurations which are based on these values are not necessarily reliable. This lead us to implementing the two-stage approach, as described in Section 3.4.

[2]Implementation detail: For all our experiments, when switching between different configurations, we reset all static parameters that depend on the configuration. The dynamic parameters such as population, covariance matrix, etc. are kept from the first configurations.

Variant	Representation	Short name
CMA-ES	00000000000	CM_0
Active CMA-ES	10000000000	CM_1
Elitist CMA-ES	01000000000	CM_2
Mirrored-pairwise CMA-ES	00100001000	CM_3
IPOP-CMA-ES	00000000001	CM_4
Active IPOP-CMA-ES	10000000001	CM_5
Elitist Active IPOP-CMA-ES	11000000001	CM_6
BIPOP-CMA-ES	00000000002	CM_7
Active BIPOP-CMA-ES	10000000002	CM_8
Elitist Active BIPOP-CMA-ES	11000000002	CM_9

Table 2: Common CMA-ES Variants. A selection of ten common CMA-ES variants is listed here, taken from [27].

5.1 Comparison of different methods

We next compare the resulting ERT using three different methods: the original method from Section 3.2, the worst-case method using sliding window from Section 3.3 and the two-stage method from Section 3.4. Since the two-stage approach selects and evaluates 50 configurations instead of just one, we show both the ERT of the configuration with the best predicted performance, as well as the one which performed best out of all 50 selected configurations. This is shown in Fig. 4. From this figure, we can see that for most functions, an improvement relative to the static configuration is achieved for at least some configuration selected in the two-stage method. However, this does not always correspond to the configuration which was predicted to perform best. In the remainder of this paper, we focus on the two-stage approach to see how these performances can be achieved.

5.2 Performance Comparison

The results of the two-stage method are shown in more detail in Fig. 6 for F10. From this figure, we can see that the fit between theory and practice is quite good, and many of the adaptive configurations manage to outperform the best static configuration by around 10%. Some outliers are present, but the general trend is positive. In this figure, we also note the ERT of the best "common" CMA-ES variant. The set of these configurations is taken from [27] and shown in Table 2.

Table 3 summarizes the results of the two-stage method for all 24 benchmark functions. The first notable result from this table is the fact that the best "common" static configuration can outperform the general best static. This is caused by the fact that these common configurations can have (B)IPOP enabled, which is not the case for the best static. In these cases, we assume that this (B)IPOP module is important to finding the optimum, and an adaptive configuration without this module will not be able to perform very well.

Next, we consider the functions for which the best static ERT is lower than that of the common variants. For these functions, we manage to improve upon this best static configuration when using an adaptive configuration. More specifically, we can see that when the best static configuration from the entire configuration space does not have (B)IPOP enabled, we can reliably achieve an improvement when using adaptive configurations.

We also note that when the best static configuration with (B)IPOP significantly outperforms the best rerun configuration, we do not manage to get the same improvements. If we would consider the best static configurations to include those with (B)IPOP and compare the performance of the adaptive configurations to those, no improvement is made at all.

In total, we find performance gains on 18 out of 24 functions of the BBOB benchmark, with stable advantages of up to 23%.

5.3 Module Activation Plots

We will now study two functions in more detail. Plots and values for other functions are available in [28]. The functions we will analyze are F10, for which we see a decent improvement for most adaptive configurations, and F24, for which we see very negative results.

First, we look at which static configurations have been selected, and how they are used within the adaptive configurations. To do this, we introduce what we call *combined module activation plots*. These plots consist of two parts, corresponding to C_1 and C_2 respectively. In each of these subplots, every line indicates a configuration. The lowest line corresponds with the theoretically best adaptive configuration, increasing from there.

In Fig. 5 and 7 we see these combined module activation plots for the selected adaptive configurations for F10 and F24 respectively. These figures clearly show that for F10 there is a pattern present among the adaptive configurations: the modules TPA and threshold start activated and in almost all cases get turned off after the split-point. Such patterns are not present in the adaptive configurations for F24. This seems to indicate that for F24 the splits are mostly chosen because of small variances between the different configurations, instead of actual inherent properties of the configurations to perform well at certain points of the search.

For the other BBOB-functions, we gathered the same data. In Fig. 8 we plot a heatmap of the module activity in the 50 best adaptive configurations for all functions we considered. For every function and all binary modules, we indicate in how many (out of 50) configurations they are selected, while differentiating between C_1 (left) and C_2 (right). From this figure, we can see that for some functions, there are clear differences between the two parts. These functions correspond to those in which the adaptive configurations improved upon the static ones. For the other functions, the differences between the two parts are not as pronounced.

This conclusion is supported by Fig. 10, where we show the relation between module difference and the fraction of adaptive configurations which do improve upon the best static configuration. The module difference is calculated as follows:

$$M = \max_{j=1}^{9} \frac{\sum_{i=1}^{50} |C_1(i,j) - C_2(i,j)|}{50}$$

Here, we have $C_1(i,j)$ indicating the value of module j of the configuration chosen as C_1 in adaptive configuration number i, and the same holds for $C_2(i,j)$. We only use the binary modules in this calculation, so the sampler is ignored. Based on this figure, we can see that there seems to be a correlation between the module difference percentage and the potential of the adaptive configurations. This supports our claim that the functions where few modules are consistently changed do not exploit any inherent properties of the functions, and instead rely on small differences in search behaviour to predict improvements. While for the functions which do see larger module differences between the first and second part, the

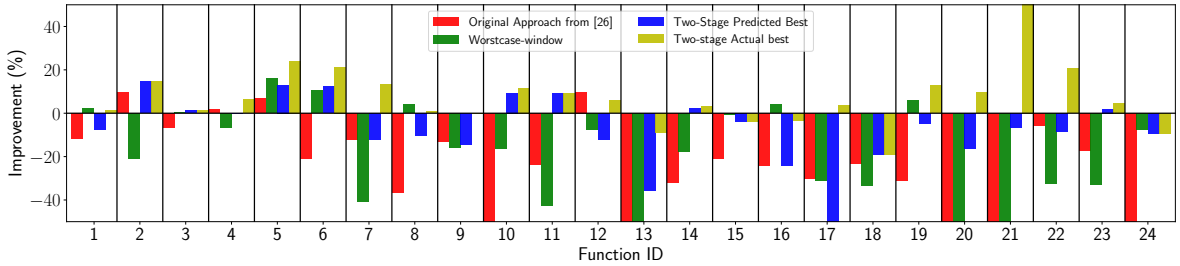

Figure 4: Relative improvement in ERTs of the adaptive configurations over the best static configuration.

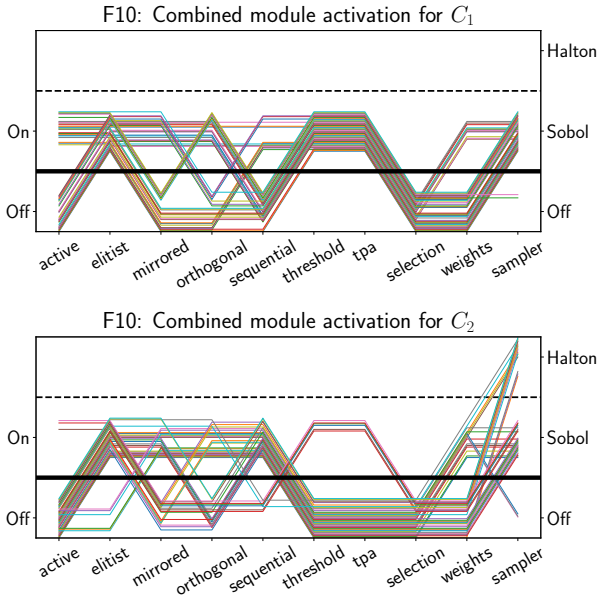

Figure 5: Combined module activation plot for the 50 best adaptive configurations for F10.

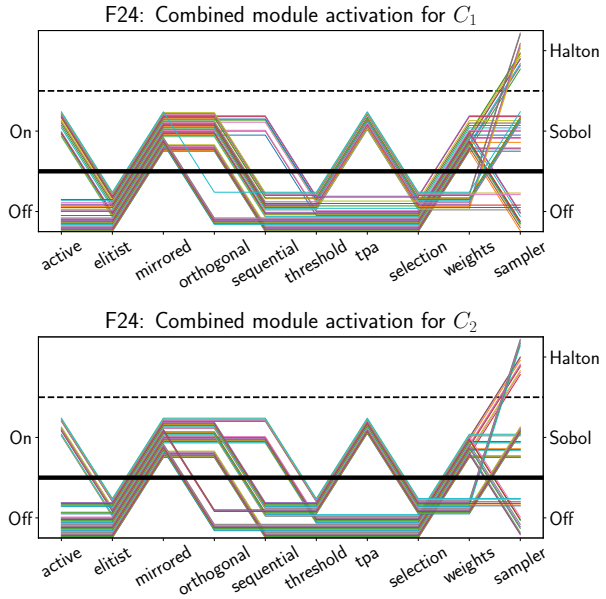

Figure 7: Combined module activation plot for the 50 best adaptive configurations for F24.

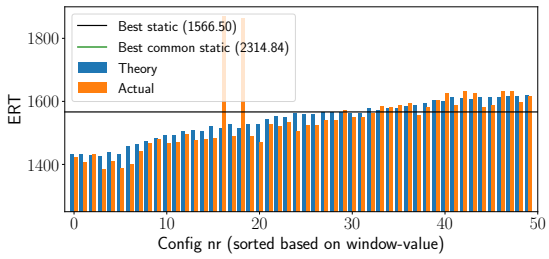

Figure 6: F10: ERT of adaptive configurations compared to the best static and "common" static configurations.

predicted improvements are more likely to be based on inherent properties of the function.

5.4 Summary of Results

From our experiment, we found large differences in the potential of our approach between functions. For some functions, such as F10, our approach seems quite stable, resulting in improvements of over 10% for several adaptive configurations, as can be seen in Fig. 6. However, this is not representative of all functions. We can see from Table 3, in the column "Amount of impr (% of configs)", that there are many functions for which few if any adaptive configurations manage to outperform the static configurations.

The total improvements gained over the static configurations can be seen in Fig. 9. In this figure, we can see the relative differences between the best static and best adaptive configurations. We can see that, for the functions where the non-IPOP common static configuration performs similarly to the general best common static ones, the adaptive configuration manages to outperform the static one. This is in line with our previous comments, indicating that the lack of (B)IPOP modules in our configuration space is a large part of the reason why some functions do not see the amount of improvement we would have hoped for.

To verify that the negative improvements are indeed not caused by our approach itself, we we compare the best expected and achieved ERT of the adaptive configurations for each function. This is shown in Fig. 11. We can see that the fit between achieved and predicted ERT is quite good, with some negative outliers for the functions with relatively high predicted ERT. This seems to confirm that our methodology is solid, and that the reason we are not seeing improvement for some functions is most likely caused by the limited configuration space and evaluation budgets.

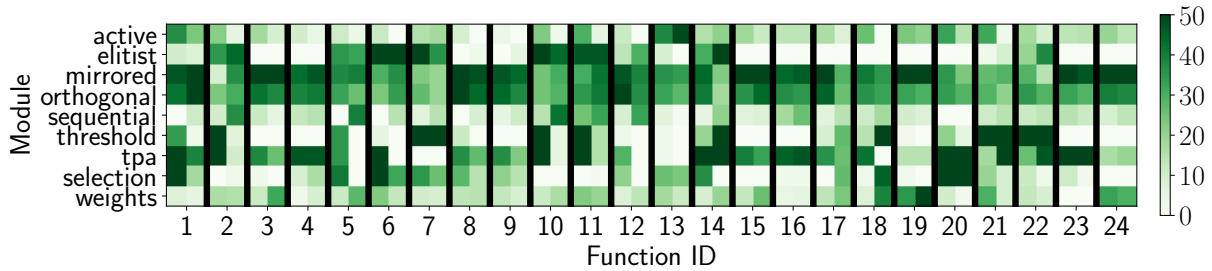

Figure 8: Heatmap of the module activations in the 50 best adaptive configuration for all our functions. For each function, activations for C_1 are on the left, while those for C_2 are on the right. The black lines separate the functions from each other.

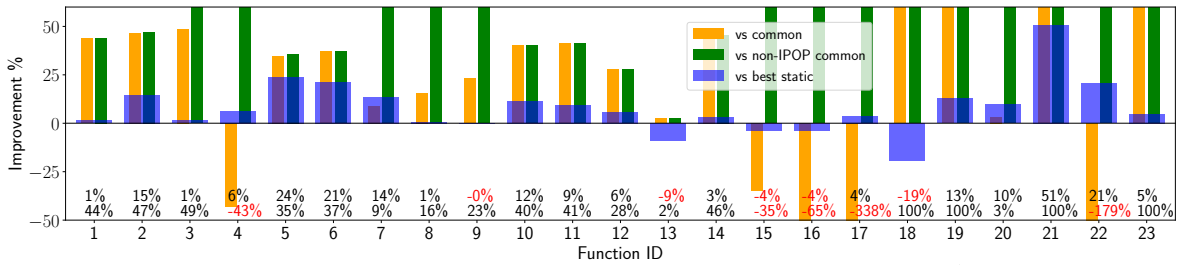

Figure 9: Comparison of the best achieved split ERT relative to the ERTs of the common statics (with and without IPOP; 5×5 runs) and the best non-IPOP static on 5×50 runs. Improvements are cut of at 60% and -50%, respectively. The precise values of the improvements are shown above the x-axis for the improvement relative to the best static (top) and relative to the best common (bottom) configuration. The precise ERT values can be found in Table 3

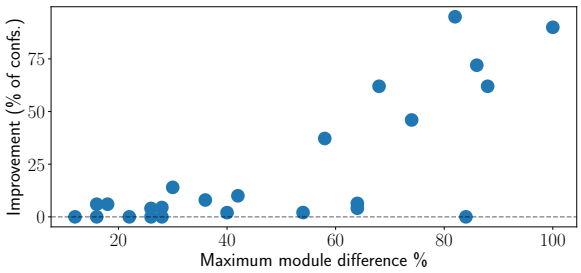

Figure 10: Relation between the maximum module difference % and the fraction of configurations which manage to improve relative to the best static configuration.

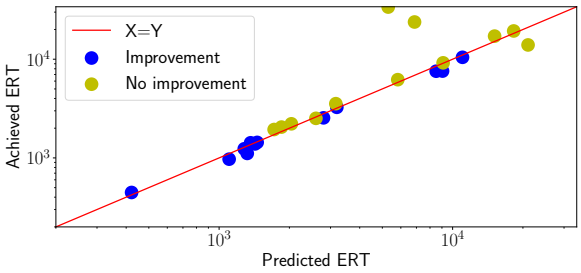

Figure 11: Relation between the best predicted and best achieved ERT for adaptive configurations. Blue dots represent functions for which the average of the 10 best achieved ERT is lower than the best static ERT, while yellow dots indicate the other functions.

6 CONCLUSION AND FUTURE WORK

In this work, we have shown the potential for configuration adaptation in CMA-ES to outperform static configurations. We have determined that for more difficult functions (B)IPOP might be required to be able to achieve improvements, but for the easier functions this module is not required.

Since this work breaks the black-box assumption of optimization by including knowledge of the value of the global optimum, which we need to determine when the target τ is hit to switch configurations, it is not directly applicable to real-world problems.

A next step would be to implement some form of landscape analysis and base the splitpoint and configurations on features of the function. We would expect the improvement on real-world problems to be larger than those we achieved on the BBOB-suite of functions, since we expect that the transition in the local fitness landscape as seen by the algorithms is more drastic than in the "sterile" BBOB functions. A cross-validation of our approach on problems found in practice forms therefore another step that we plan to work on in the near future.

Another approach would be to extend the framework we used with more modules, and allow the (B)IPOP module to be used in adaptive configurations. This would greatly increase the configuration space, and might necessitate a different approach to determining which configurations to consider.

Finally, we plan on extending our work by tuning the hyperparameters of the most successful CMA-ES configurations.

ACKNOWLEDGMENTS

This work was supported by the Paris Ile-de-France Region. Our experiments were run on the Leiden DSLAB and DAS-4 [5] servers.

Fid	Target	Common config	Common ERT	Common config (non IPOP)	non IPOP ERT	Best static (B)IPOP	Best static ERT	Best rerun config	Best rerun ERT	Best adaptive ERT	Relative improvement	Predicted adaptive ERT	Avg 10 splits ERT	Fraction of impr. configs (%)
1	10e-8.0	CM_6	795	CM_2	799	0	412	00110011010	453	**446**	1.4%	*422*	453	6%
2	10e-8.0	CM_4	2,313	CM_2	2,338	0	1,348	10010110010	1,448	**1,236**	14.6%	1,282	1,283	72%
3	10e0.4	CM_9	17,871	/	∞	2	2,752	00110000110	9,316	**9,189**	1.4%	9,104	9,771	2%
4	10e0.8	CM_7	7,301	/	∞	2	2,837	00110010010	11,162	**10,456**	6.3%	*11,024*	11,034	6%
5	10e-8.0	CM_4	1,700	CM_3	1,731	0	1,268	00111000010	1,461	**1,110**	24.0%	*1,320*	1,128	95%
6	10e-8.0	CM_2	1,540	CM_2	1,540	0	1,106	01110001010	1,234	**971**	21.3%	*1,104*	1,008	90%
7	10e-8.0	CM_5	2,768	/	∞	1	1,471	01000101010	2,912	**2,517**	13.5%	*2,598*	3,002	4%
8	10e-8.0	CM_4	2,623	/	∞	2	1,765	00110000010	2,233	**2,216**	0.8%	*2,039*	2,239	4%
9	10e-8.0	CM_4	2,524	/	∞	1	1,726	00110000020	1,938	1,940	-0.1%	*1,720*	2,095	0%
10	10e-8.0	CM_2	2,315	CM_2	2,315	2	1,437	01000110010	1,566	**1,386**	11.5%	1,425	1,423	62%
11	10e-8.0	CM_0	2,444	CM_0	2,444	0	1,399	11110110010	1,586	**1,436**	9.4%	1,456	1,475	62%
12	10e-8.0	CM_2	4,509	CM_2	4,509	0	3,019	00110000010	3,463	**3,256**	6.0%	*3,192*	3,312	37%
13	10e-8.0	CM_1	3,632	CM_1	3,632	2	2,798	10110001000	3,253	3,544	-8.9%	*3,161*	3,736	0%
14	10e-8.0	CM_0	2,619	CM_0	2,619	1	1,329	01001110010	1,471	**1,423**	3.3%	*1,364*	1,434	46%
15	10e0.4	CM_4	4,598	/	∞	1	1,847	00110000110	5,980	6,204	-3.7%	5,822	6,883	0%
16	10e-2.0	CM_8	10,379	/	∞	1	3,151	00110010010	16,534	17,140	-3.7%	*15,127*	18,753	0%
17	10e-4.4	CM_7	5,437	/	∞	2	2,209	00110010010	24,738	**23,824**	3.7%	*6,870*	28,991	2%
18	10e-4.0	/	∞	/	∞	1	6,850	00110101110	28,312	33,754	-19.2%	5,298	38,432	0%
19	10e-0.6	/	∞	/	∞	1	3,994	00100000120	8,666	**7,559**	12.8%	*8,503*	8,378	14%
20	10e0.2	CM_6	2,628	/	∞	2	769	00010011010	2,817	**2,547**	9.6%	*2,800*	2,785	8%
21	10e-0.6	/	∞	/	∞	2	5,889	00001110010	28,386	**13,949**	50.9%	*21,040*	30,338	4%
22	10e0.0	CM_7	2,714	/	∞	2	653	00000110010	9,546	**7,581**	20.6%	*9,054*	9,085	10%
23	10e-0.8	/	∞	/	∞	2	5,721	00110010010	20,296	**19,328**	4.8%	*18,263*	21,386	6%
24	10e1.0	CM_8	2,953	/	∞	2	1,187	00110000110	1,877	2,053	-9.4%	*1,850*	2,358	0%

Table 3: Overview of the results from the experiment described in Section 3.4. The "Common config" and "Common config (non IPOP)" are selected from the configurations in Table 2. The column "Best static (B)IPOP" and its ERT ("Best static ERT") are selected from the full configuration space using the 5×5 run data. The remaining ERTs are from the 5×50 runs, as described in Section 3.4. The columns "Best adaptive" and "Predicted adaptive ERT" do not necessarily relate to the same configuration. When this is not the case, it is indicated by the italic values in the predicted column. The column "Relative improvement" shows the relative improvement in ERT of the best adaptive configuration vs the best rerun static configuration. The column "Avg 10 splits ERT" is calculated as the average of the 10 best achieved ERTs of the adaptive configurations. The column "Fraction of impr. configs (%)" indicates how many of the adaptive configurations performed better than the "Best rerun" configuration.

REFERENCES

[1] Anne Auger, Dimo Brockhoff, and Nikolaus Hansen. 2011. Mirrored Sampling in Evolution Strategies with Weighted Recombination. In *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation (GECCO '11)*. ACM, New York, NY, USA, 861–868. https://doi.org/10.1145/2001576.2001694

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

[3] Anne Auger, Mohamed Jebalia, and Olivier Teytaud. 2005. Algorithms (X, sigma, eta): Quasi-random Mutations for Evolution Strategies. In *Artificial Evolution, 7th International Conference, Evolution Artificielle, EA 2005, Lille, France, October 26-28, 2005, Revised Selected Papers (Lecture Notes in Computer Science)*, El-Ghazali Talbi, Pierre Liardet, Pierre Collet, Evelyne Lutton, and Marc Schoenauer (Eds.), Vol. 3871. Springer, 296–307. https://doi.org/10.1007/11740698_26

[4] Thomas Bäck, Christophe Foussette, and Peter Krause. 2013. *Contemporary Evolution Strategies*. Springer.

[5] Henri Bal, Dick Epema, Cees de Laat, Rob van Nieuwpoort, John Romein, Frank Seinstra, Cees Snoek, and Harry Wijshoff. 2016. A medium-scale distributed system for computer science research: Infrastructure for the long term. *Computer* 5 (2016), 54–63.

[6] Thomas Bartz-Beielstein, Marco Chiarandini, Luís Paquete, and Mike Preuss. 2010. *Experimental methods for the analysis of optimization algorithms*. Springer.

[7] Thomas Bartz-Beielstein, Christian Lasarczyk, and Mike Preuss. 2010. The sequential parameter optimization toolbox. In *Experimental methods for the analysis of optimization algorithms*. Springer, 337–362.

[8] Nacim Belkhir, Johann Dréo, Pierre Savéant, and Marc Schoenauer. 2017. Per instance algorithm configuration of CMA-ES with limited budget. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO'17)*. ACM, 681–688. https://doi.org/10.1145/3071178.3071343

[9] Dimo Brockhoff, Anne Auger, Nikolaus Hansen, Dirk V. Arnold, and Tim Hohm. 2010. Mirrored Sampling and Sequential Selection for Evolution Strategies. In *Parallel Problem Solving from Nature, PPSN XI*. Springer, Berlin, Heidelberg, 11–21. https://doi.org/10.1007/978-3-642-15844-5_2

[10] Edmund K. Burke, Michel Gendreau, Matthew R. Hyde, Graham Kendall, Gabriela Ochoa, Ender Özcan, and Rong Qu. 2013. Hyper-heuristics: a survey of the state of the art. *JORS* 64, 12 (2013), 1695–1724. https://doi.org/10.1057/jors.2013.71

[11] Edmund K Burke, Barry McCollum, Amnon Meisels, Sanja Petrovic, and Rong Qu. 2007. A graph-based hyper-heuristic for educational timetabling problems. *European Journal of Operational Research* 176, 1 (2007), 177–192.

[12] Nikolaus Hansen. 2008. CMA-ES with Two-Point Step-Size Adaptation. *arXiv:0805.0231 [cs]* (May 2008). http://arxiv.org/abs/0805.0231 arXiv: 0805.0231.

[13] Nikolaus Hansen. 2009. 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)*. ACM, New York, NY, USA, 2389–2396. https://doi.org/10.1145/1570256.1570333

[14] Nikolaus Hansen, Anne Auger, Dimo Brockhoff, Dejan TuÅąar, and Tea TuÅąar. 2016. COCO: Performance Assessment. *arXiv:1605.03560 [cs]* (May 2016). http://arxiv.org/abs/1605.03560 arXiv: 1605.03560.

[15] Nikolaus Hansen, Anne Auger, Steffen Finck, and Raymond Ros. 2009. *Real-Parameter Black-Box Optimization Benchmarking 2009: Experimental Setup*. report. INRIA. https://hal.inria.fr/inria-00362649/document

[16] Nikolaus Hansen, Anne Auger, Raymond Ros, Steffen Finck, and Petr Pošík. 2010. Comparing results of 31 algorithms from the black-box optimization benchmarking BBOB-2009. In *Proceedings of the 12th annual conference companion on Genetic and evolutionary computation*. ACM, 1689–1696.

[17] Nikolaus Hansen and Andreas Ostermeier. 2001. Completely Derandomized Self-Adaptation in Evolution Strategies. *Evolutionary Computation* 9, 2 (2001), 159–195. https://doi.org/10.1162/106365601750190398

[18] Holger H. Hoos, Frank Neumann, and Heike Trautmann. 2016. Automated Algorithm Selection and Configuration (Dagstuhl Seminar 16412). *Dagstuhl Reports* 6, 10 (2016), 33–74. https://doi.org/10.4230/DagRep.6.10.33

[19] Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. 2011. Sequential model-based optimization for general algorithm configuration. In *International Conference on Learning and Intelligent Optimization*. Springer, 507–523.

[20] G. A. Jastrebski and D. V. Arnold. 2006. Improving Evolution Strategies through Active Covariance Matrix Adaptation. In *2006 IEEE International Conference on Evolutionary Computation*. 2814–2821. https://doi.org/10.1109/CEC.2006.1688662

[21] Pascal Kerschke, Holger H. Hoos, Frank Neumann, and Heike Trautmann. 2018. Automated Algorithm Selection: Survey and Perspectives. *CoRR* abs/1811.11597 (2018). arXiv:1811.11597 http://arxiv.org/abs/1811.11597

[22] Pascal Kerschke and Heike Trautmann. 2017. Automated Algorithm Selection on Continuous Black-Box Problems By Combining Exploratory Landscape Analysis and Machine Learning. *arXiv preprint arXiv:1711.08921* (2017).

[23] Mario A Muñoz, Yuan Sun, Michael Kirley, and Saman K Halgamuge. 2015. Algorithm selection for black-box continuous optimization problems: a survey on methods and challenges. *Information Sciences* 317 (2015), 224–245.

[24] A. Piad-Morffis, S. Estévez-Velarde, A. Bolufé-Röhler, J. Montgomery, and S. Chen. 2015. Evolution strategies with thresheld convergence. In *2015 IEEE Congress on Evolutionary Computation (CEC)*. 2097–2104. https://doi.org/10.1109/CEC.2015.7257143

[25] Sander van Rijn. 2018. Modular CMA-ES framework from [27], v0.3.0. https://github.com/sjvrijn/ModEA. Available also as pypi package at https://pypi.org/project/ModEA/0.3.0/.

[26] Sander Van Rijn, Carola Doerr, and Thomas Bäck. 2018. Towards an Adaptive CMA-ES Configurator. In *Proc. of 15th International Conference on Parallel Problem Solving from Nature (PPSN'18) (Lecture Notes in Computer Science)*, Vol. 11101. Springer, 54–65. https://doi.org/10.1007/978-3-319-99253-2_5

[27] Sander van Rijn, Hao Wang, Matthijs van Leeuwen, and Thomas Bäck. 2016. Evolving the structure of Evolution Strategies. In *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*. 1–8. https://doi.org/10.1109/SSCI.2016.7850138

[28] Diederick Vermetten, Sander van Rijn, Thomas Bäck, and Carola Doerr. 2019. Online Selection of CMA-ES Variants. *CoRR* abs/1904.07801 (2019). arXiv:1904.07801 http://arxiv.org/abs/1904.07801 A GitHub repository containing more data and experiments from this project is available at https://github.com/Dvermetten/Online_CMA-ES_Selection.

[29] Hao Wang, Michael Emmerich, and Thomas Bäck. 2014. Mirrored Orthogonal Sampling with Pairwise Selection in Evolution Strategies. In *Proceedings of the 29th Annual ACM Symposium on Applied Computing (SAC '14)*. ACM, New York, NY, USA, 154–156. https://doi.org/10.1145/2554850.2555089