Trajectory-based algorithm selection with warm-starting

Jankovic, A.; Vermetten, D.L.; Kostovska, A.; Nobel, J.P. de; Eftimov, T.; Doerr, C.

Trajectory-based Algorithm Selection with Warm-starting

Anja Jankovic, Diederick Vermetten, Ana Kostovska, Jacob de Nobel, Tome Eftimov and Carola Doerr

***Abstract*—Landscape-aware algorithm selection approaches have so far mostly been relying on landscape feature extraction as a preprocessing step, independent of the execution of optimization algorithms in the portfolio. This introduces a significant overhead in computational cost for many practical applications, as features are extracted and computed via sampling and evaluating the problem instance at hand, similarly to what the optimization algorithm would perform anyway within its search trajectory. As suggested in [Jankovic et al., EvoAPP 2021], trajectory-based algorithm selection circumvents the problem of costly feature extraction by computing landscape features from points that a solver sampled and evaluated during the optimization process. Features computed in this manner are used to train algorithm performance regression models, upon which a per-run algorithm selector is then built.**

In this work, we apply the trajectory-based approach onto a portfolio of five algorithms. We study the quality and accuracy of performance regression and algorithm selection models in the scenario of predicting different algorithm performances after a fixed budget of function evaluations. We rely on landscape features of the problem instance computed using one portion of the aforementioned budget of the same function evaluations. Moreover, we consider the possibility of switching between the solvers once, which requires them to be warm-started, i.e. when we switch, the second solver continues the optimization process already being initialized appropriately by making use of the information collected by the first solver. In this new context, we show promising performance of the trajectory-based per-run algorithm selection with warm-starting.

***Index Terms*—dynamic algorithm selection, exploratory landscape analysis, evolutionary computation, black-box optimization**

I. Introduction

Optimization is a central notion across a broad range of scientific disciplines and real-world applications. Finding an optimal solution for a given problem is often not a straightforward process, as problems are typically computationally hard or otherwise intractable. In many concrete use cases, knowledge about the inherent nature of the problem is very limited, which renders formal problem modeling impossible. Under these circumstances, users are required to treat such problems as a black box. *Black-box optimization* (BBO) provides techniques that are able to generate good solutions for these problems in a reasonable time. These techniques, known as BBO algorithms, operate via an iterative process of sampling and evaluating solution candidates and using the knowledge obtained in the previous iterations to guide the search towards more promising alternatives, until eventually converging to the best estimate of an optimal solution.

Due to the plethora of existing optimization problems, different BBO algorithms have been developed to this day. Various underlying operating mechanisms of these algorithms yield their complementary behavior on different problems. This large algorithmic variety poses a meta-optimization problem in achieving peak performance [1]: how does one select the most efficient algorithm for a given problem instance?

In recent years, significant research focus has been put on algorithm selection approaches that make use of the knowledge about the problem instance landscape to base the decision about which algorithm to use in that particular situation [2]–[4]. *Landscape-aware algorithm selection* generally relies on an important preprocessing step of extracting information from the problem instance landscape (independently of the optimization process). A huge challenge in this regard is the overhead computational cost induced by this preprocessing step, as further resources are spent on sampling and evaluating search points to first characterize the landscape, but are not at all considered while executing the algorithm on a problem instance. In many practical applications, users cannot afford to spend those additional function evaluations prior to optimizing, as they can be very expensive (e.g., crash tests or clinical studies). The approach suggested in [5] offers a convenient alternative perspective in which the information about the problem instance landscape is extracted via samples and their function evaluations performed anyway by the algorithm. This framework shows preliminary potential in circumventing the preprocessing step altogether, and might present a step forward in the direction of designing an efficient fully dynamic algorithm selection model. However, an important open question from [5] remains: how do we make use of the trajectory-based information of a default algorithm to predict performances of other algorithms? We tackle it with this work.

In this paper, we extend the trajectory-based landscape-aware approach to a portfolio of five widely used black-box optimization algorithms, and we consider that we can switch between the algorithms once during the optimization process.

Anja Jankovic (Email: Anja.Jankovic@lip6.fr) and Carola Doerr (Email: Carola.Doerr@lip6.fr) are with Sorbonne Université, CNRS, LIP6, Paris, France. Diederick Vermetten (Email: d.l.vermetten@liacs.leidenuniv.nl) and Jacob de Nobel (Email: j.p.de.nobel@liacs.leidenuniv.nl) are with Leiden University, The Netherlands. Ana Kostovska (Email: ana.kostovska@ijs.si) is with the Department of Knowledge Technologies, Jožef Stefan Institute, Ljubljana, Slovenia as well as with the Jožef Stefan International Postgraduate School, Ljubljana, Slovenia. Tome Eftimov (Email: tome.eftimov@ijs.si) is with the Computer Systems Department, Jožef Stefan Institute, Ljubljana, Slovenia.

Our work is supported by the Paris Ile-de-France region, by projects from the Slovenian Research Agency (research core funding No. P2-0098, P2-0103, the young researcher grant No. PR-09773 to AK, and researcher funding No. N2-0239), as well as by the EC through grant No. 952215 (TAILOR)

2022 IEEE Congress on Evolutionary Computation (CEC) | 978-1-6654-6708-7/22/$31.00 ©2022 IEEE | DOI: 10.1109/CEC55065.2022.9870222

Put differently, we start optimizing by the base algorithm (*A1*) while collecting the landscape data from the algorithm trajectory, then switch to another algorithm (*A2*) that is *warm-started*, i.e. appropriately initialized by making use out of the gathered information from *A1*. We first show that we are able to get decent predictions of the *A2* performance using *A1*'s trajectory-based landscape characterization, upon which we then build an algorithm selection model. At higher levels of performance complementarity between the algorithms (which coincide with larger total budgets for running the algorithms), the algorithm selector outperforms any of the standalone algorithms in terms of loss against the true best algorithm for each particular problem instance. We finally highlight some advantages as well as drawbacks with respect to different parts of the adopted pipeline and present some important challenges for future work.

Outline of the Paper: Throughout Sec. II, we position the work in the context of numerical black-box optimization, introduce the benchmark collection (II-A), the algorithm portfolio selected for our work (II-B), and the notion of warm-starting (II-C), recap the state-of-the-art in per-instance algorithm selection (II-D) and establish the ground for the trajectory-based approach in analyzing problem instance landscapes (II-E). We give an overview of the full experimental setup in Sec. III, focusing on each component of the trajectory-based algorithm selection framework applied to a diverse algorithm portfolio. We present and discuss the main findings of our study in Sec. IV and provide critical assessment of this approach in Sec. V. Finally, in Sec. VI, we wrap up with open questions that merit further attention.

Reproducibility and Additional Figures: To ensure that the work shown in this paper is reproducible [6], all data and code used is made available on `figshare` [7]. This includes figure generation code for figures which have not been included here because of the limited space available. For ease of viewing, these additional figures can also be viewed in this same repository. In particular, the losses of our model for different budgets can be found there.

II. Background and Selected Benchmark Environment

Black-box optimization algorithms are a wide-ranging family of adaptive sampling-based strategies. Many of these algorithms can adjust their behavior during the optimization process by taking into account the information gathered during the run. This is especially useful as typically different phases of the optimization process require different search behavior (conveniently illustrated in the trade-off between exploration and exploitation). Moreover, different algorithms employ different search routines which can be more or less beneficial depending on a given scenario. A large open question that stems from this observation is to detect which algorithm is the most suitable for a given stage of the optimization process. First steps into this direction have shown potential in switching the algorithm once during the run itself [8]. However, this potential remains largely untapped if we restrain ourselves to a family of similar algorithms [9]. Having a wider set of different solvers might help bypass this obstacle.

A. The BBOB Problem Collection

In the context of numerical black-box optimization, assessing the performance of different algorithms on different problem instances is largely facilitated by the existence of well-established *benchmark problem collections*. In this study, we rely on one such collection, the BBOB noiseless testbed from the COCO environment [10]. The BBOB collection comprises 24 functions. For each of these functions, multiple problem instances are available. We conveniently generate, access and analyze the benchmark data via the `IOHprofiler` platform [11].

B. Algorithm Portfolio

Following suggestions from [12] and related works, we opt for five algorithms that are frequently used to tackle numerical black-box problems:

- BFGS (Broyden-Fletcher-Goldfarb-Shanno [13]–[16]): a Quasi-Newton method that approximates the Jacobian or the Hessian instead of actually computing them.
- CMA-ES (Covariance Matrix Adaption - Evolution Strategy [17]): a stochastic derivative-free numerical optimization algorithm that iteratively samples the population from a multivariate normal distribution and updates the shape of the said distribution with the information gathered while running.
- DE (Differential Evolution [18]): a population-based algorithm that samples candidates purely based on numerical differences between existing population members.
- MLSL (Multi-Level Single Linkage [19], [20]): an algorithm that combines global search phases (based on clustering) with more focused, local search procedures.
- PSO (Particle Swarm Optimization [21]): simulates particles moving around the search space based on their individual velocity, determining both speed and direction, motivated by the swarm behavior of some animal species.

C. Warm-starting

To enable switching from CMA-ES to each of the five algorithms mentioned previously, we make use of some basic warm-starting strategies. The most intuitive version of warm-starting is to inherit the best-so-far solution and use this as the starting point for the second algorithm. This point can then be used as a center to initialize the new population around, as is done by CMA-ES, DE and PSO. For switching to BFGS, we can use the covariance matrix directly, by using this as the inverse of the Hessian [22]. We should note that by warm-starting the CMA-ES using the same procedure as the other population-based approaches, we lose a significant amount of information, which might lead to worse performance than expected if we had not switched. However, it does mean that the approach is easier to extend, since we can modify the starting algorithm while having minimal impact on the procedure as a whole.

D. Per-Instance Algorithm Selection

As mentioned in Sec. I, given an optimization problem, its specific instance that needs to be solved, and a set of algorithms that can be used to solve it, the so-called *per-instance algorithm selection (PIAS)* problem arises: how to determine which of those algorithms can be expected to perform best on that particular instance? In other words, one is not interested in having an algorithm recommendation for a whole problem class (such as TSP or SAT in the discrete domain), but for a specific instance of some problem. A large body of work exists in this line of research [4], [23]–[27], and they mostly rely on extracting the information about the problem instances beforehand, contrary to this study.

To assess the algorithm selector's quality, two standard baselines are used. The performance of a (hypothetical) perfect per-instance algorithm selector, also known as the *virtual best solver* (VBS) or the oracle selector, provides a lower bound on the performance of any realistically achievable algorithm selector. The VBS always selects the true best algorithm per each problem instance. On the other hand, a natural upper bound on the algorithm selector performance is provided by the *single best solver* (SBS), which is the algorithm with the best overall performance among all other algorithms in the considered portfolio. The ratio between VBS and SBS performances, also referred to as the *VBS-SBS gap*, gives an indication of the performance gains that can be obtained by per-instance algorithm selection in the best case. Consequently, the fraction of this gap closed by a certain algorithm selector provides a measure of its quality [28].

E. Trajectory-Based Exploratory Landscape Analysis

In order to represent the considered optimization problem instances in a suitable and useful way for the algorithm selection pipeline, we shall want to quantify their different characteristics via appropriate measures. This is typically done by means of *exploratory landscape analysis* (ELA) [29]. Problem instances are characterized by automatically computed ELA *features* using information extracted via sampling and evaluating the problem. A vector of numerical ELA feature values is assigned to each instance and can be then used to train a predictive model that maps it to different algorithms' performances on the said instance. The feature extraction step is commonly considered to be independent from the optimization process, and diverse sampling strategies can be employed, see [30] for a discussion. Conveniently, feature computation is done via the R package `flacco` [46], and the ELA features we consider here are suggested in [5].

However, as the nature of the knowledge needed to extract features and to optimize a problem instance is the same, the motivation arises to save computational resources from the preprocessing step by incorporating the feature extraction within the optimization process. The core idea is to utilize samples already evaluated by the algorithm to compute the landscape features (as seen locally on the algorithm's search trajectory) [5]. We adopt this perspective in this paper, extending it to a more diverse portfolio. We collect the points sampled by the base algorithm and use so-computed features to predict the performance of another (warm-started) algorithm which continues the optimization process.

III. Experimental Setup

As discussed in Sec. II-B, in this work we make use of a small portfolio of algorithms. The specific portfolio is chosen based on the observed differences in their potential performance as either the first (*A1*) or second (*A2*) part of a dynamically switching algorithm [12]. In particular, we consider the following algorithm implementations:

- CMA-ES: we use the modular CMA-ES (`modCMA`) [31], [32], which implements a wide range of variants into one modular framework with default settings and saturation as the boundary correction method.
- DE: we use the `scipy` [33] implementation.
- PSO: we implement a basic version with clipped velocity to avoid exploding trajectories.
- MLSL: we implement the version described in [34], using `scipy`'s version of BFGS for the local search procedure.
- BFGS: we adapt `scipy`'s implementation.

We then use the first five instances of each of the 24 BBOB functions mentioned in Sec. II-A for our experiments.

Our dynamic algorithm follows a two-stage process; first it starts with the modular CMA-ES for $30 \cdot D$ evaluations, rounded up to the nearest multiple of the used population size. This equates to 154 evaluations for the 5-dimensional version of the functions. After this point, the run is interrupted and the second algorithm is warm-started as described in Sec. II-C. This experiment is repeated 10 times on each of the first five instances of all 24 BBOB functions and for each of the five algorithms, resulting in a total of $6\,000$ runs. To execute this data collection, we used `IOHprofiler` [35], which enabled us to keep track of the full search history, as well as performance data and the state variables, some of which are needed to warm-start the algorithms that we switch to. As our performance measure, we take the function value reached after a fixed number of function evaluations (i.e., the fixed-budget target precision).

The study presented in [5] also experimented with the use of state variables as features for the performance predictions. Since no significant advantage was observed in [5] for these variables, we do not make use of them here in this work (apart from extracting the information that is needed to warm-start the algorithms after the switch). In order to allow for a comparison with such an approach, we have nevertheless recorded the state variables listed in [5]; they can be found in the data record made available at [7].

A. Performance Data

For each switching algorithm, we collect a total of $1\,200$ runs. Since these runs all start with the same 154 evaluations from the CMA-ES, we are mostly interested in the complementarity of their performance after this point. To visualize this, we show the evolution of the mean function value over time in Fig. 1. Here, we can see that for most

of the unimodal functions, the switch to BFGS significantly outperforms all others, as would be expected, since the BFGS has the most involved warm-starting procedure. For the more complex functions, however, this initial benefit from switching to BFGS disappears after a while, with the other algorithm catching up and steadily overtaking it. This highlights a key aspect of the prediction problem, namely the allocation of budget to the second algorithm. The optimal switching algorithm for a total budget of 350 can differ widely from one where the overall budget is $1\,050$ evaluations. Luckily, we can simulate the procedure for short budgets by cutting of the run at the required point and measuring the performance at that point, allowing us to investigate the impact of this overall budget is more detail. In particular, we consider the following set of *A2* budgets for the second part of the search: $\{100, 200, 300, 500, 700, 900\}$ function evaluations, while the *A1* trajectory budget allocated for the feature extraction is fixed at 150 function evaluations.

B. Selection of Regression Models

We note here that, as we operate within a fixed-budget setting, target precision values rapidly get smaller as we converge to the optimum. Therefore, not only do we perform a classical regression on the actual data, but we also take into account the possibility of more accurately predicting these very small target precision values via training the regression model on the log-values of the same data. We then learn a separate regression model per each combination of the algorithm, considered budget and type of the target value (actual and log). This leaves us with a total number of 60 different regression models (5 algorithms $\times$ 6 budgets $\times$ 2 targets). To learn the regression models, as classically suggested in the literature, we use the Random Forest (RF) algorithm as implemented in the Python package `scikit-learn` [37] and perform hyperparameter tuning using the grid search methodology. We tune five different RF hyperparameters: (1) *n_estimators* – the number of trees in the random forest; (2) *max_features* – the number of features used for making the best split ; (3) *max_depth* – the maximum depth of the trees; (4) *min_samples_split* – the minimum number of samples required for splitting an internal node in the tree; and (5) *criterion* – the function that measures the quality of a given split. The full list of tuned hyperparameters and their corresponding search spaces is given in Tab. I.

To evaluate the predictive performance of the regression models, we employ the *leave-one-group-out* strategy. Here, the groups are defined on the ID of the problem instance (1–5), which means we work with five different groups. We thus perform five iterations over the data, and we hold one instance out each time (all 10 runs included), train the model on the remaining data, and test on the test (hold-out) data. We use the R^2 score as an evaluation measure of predictive power of the models. Finally, to obtain the test error, we compute the average R^2 score over the five hold-out groups. The average R^2 scores for the regression models with actual target precision are given in Tab. II, while R^2 scores for the regression models with the log-target precision can be found in Tab. III.

TABLE I: RF hyperparameter names and their corresponding values considered in the grid search.

Hyperparameter	Search space
n_estimators	$[100, 500, 1000]$
max_features	[AUTO, SQRT, LOG2]
max_depth	$[4, 8, 15,$ NONE$]$
min_samples_split	$[2, 5, 10]$
criterion	[SQUARED_ERROR, ABSOLUTE_ERROR, POISSON]

TABLE II: R^2 scores for the regression models trained on the actual target precision for all considered *A2* budgets.

Algorithm	100	200	300	500	700	900
BFGS	0.0637	0.3059	0.4764	0.4854	0.4869	0.4860
CMAES	0.5030	0.1473	0.0993	0.2514	0.2353	0.1152
DE	0.1700	0.2699	0.1571	0.1322	0.0333	-0.0321
MLSL	0.2410	0.2066	0.3142	-0.1059	-0.0641	-0.0279
PSO	0.5361	0.5694	0.5884	0.3919	0.1398	-0.8812

We observe from Tab. II and Tab. III that the regression models for log-target precision generally outperform the models with the actual target precision. For this reason, in the remainder of the paper we focus exclusively on the log-trained models as a basis for our algorithm selector.

C. Evaluation of the Algorithm Selector

Once the predictions from all regression models are available, the next step is to select the best algorithm for each performed run on every problem instance. To this end, we choose the algorithm whose regression model provides the best predicted performance value for that run (i.e., we refer to this algorithm as the selected algorithm). For each run, we also identify the best algorithm based on the raw performance data (i.e., we refer to this algorithm as the best algorithm or the *virtual best solver*).

To evaluate the performance of the algorithm selector, for each run individually, we compute the difference between the target precision of the selected algorithm $F_{\mathcal{A}}$ and that of the best one $F_{\mathcal{A}^*}$ (for that particular run). More precisely, we consider the difference after taking the logarithm of the achieved target precision: $\mathcal{L}(\mathcal{A}, \mathcal{A}^*) = \log(F_{\mathcal{A}}) - \log(F_{\mathcal{A}^*})$. This gives us one performance measure per run, and we mainly investigate the distribution of these "*losses*" over all $1\,200$ runs, which we compare to that of the five algorithms.

TABLE III: R^2 scores for the regression models trained on the log-target precision for all considered *A2* budgets.

Algorithm	100	200	300	500	700	900
BFGS	0.7016	0.6691	0.7073	0.7425	0.7492	0.7570
CMAES	0.6708	0.7006	0.7695	0.8423	0.8053	0.7894
DE	0.6721	0.6549	0.6324	0.6109	0.6194	0.6669
MLSL	0.7296	0.7277	0.8722	0.8687	0.8678	0.8688
PSO	0.7205	0.7017	0.7980	0.9128	0.9137	0.8745

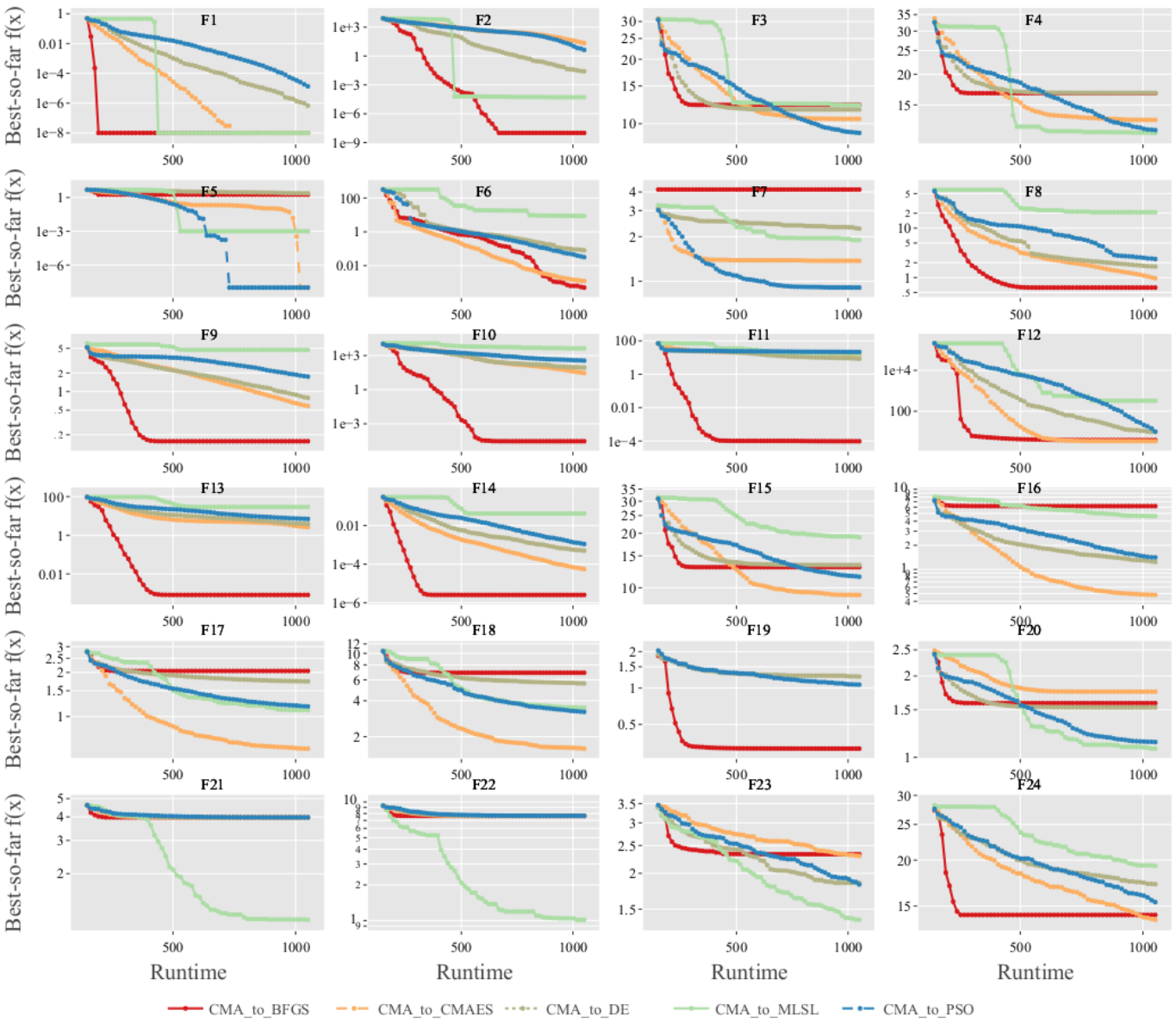


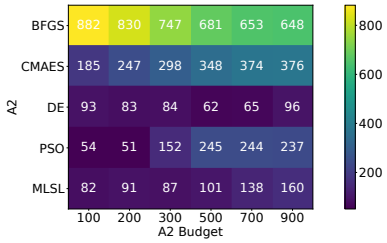
Fig. 1: Mean best-so-far function value (target precision) reached by each of the five switching algorithms on all 24 BBOB functions. Each line corresponds to 50 runs: 10 on each of the first five instances of the function. Note that the first 154 evaluations are identical for each algorithm, and are thus excluded from the figure. Figure generated using `IOHanalyzer` [36].

IV. Results and Discussion

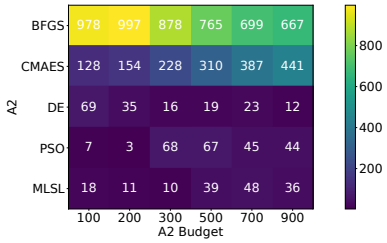
We first present the results of our trajectory-based algorithm selection approach for the full algorithm portfolio. Since BFGS clearly dominates several of the settings, we also consider what happens if we exclude it from the portfolio.

Full Portfolio. Fig. 3 shows the loss (computed as described in Sec. III-C) of the five algorithms and our trajectory-based algorithm selector, for two *A2* budgets, 100 and 900. As already visible in Fig. 1, the performance differences between the algorithms is not very pronounced for the small budget, with a vast majority of losses smaller than one order of magnitude. BFGS nevertheless clearly outperforms the other four algorithms. Our algorithm selector selects BFGS on 972 out of all 1 200 runs (see Fig. 2b). It performs slightly worse than BFGS, i.e., we do not gain in this setting from the landscape-aware selection.

For budget 900 the situation is different. Here, BFGS is still the best solver when considering the loss distribution over all 1 200 runs. However, CMA-ES and MLSL are best for 252 and 167 runs, respectively (see Fig. 2a), and our algorithm selector manages to distinguish between these runs in at least some cases. To further probe into the decision of the algorithm selector, we present a confusion matrix in Tab. IV. Our selector has chosen BFGS 667 times in total, and in 487 of these cases this choice was optimal. For 48 runs it would have been better

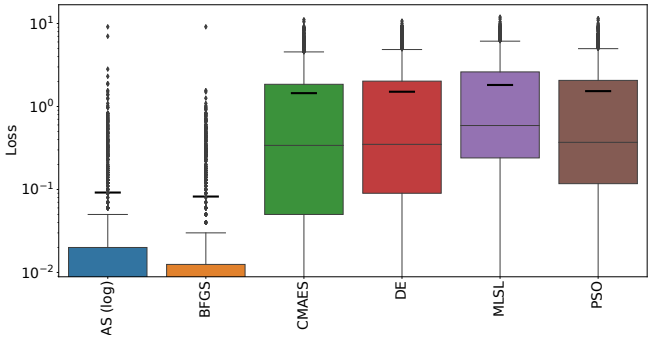


(a) How often each solver is the best to switch to.

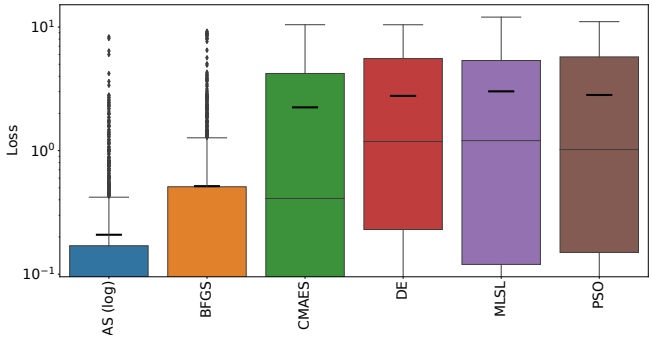


(b) How often each solver is actually selected.

Fig. 2: Heatmap showing in how many (out of 1 200) runs each algorithm is the best one to switch to (left) and is selected to switch to by the logarithmic model (right), based on the amount of budget given to this second part of the search. Results are capped at 10^{-8} target precision, which can lead to ties. The number of best algorithms per budget can therefore be larger than 1 200.



(a) Budget 100



(b) Budget 900

Fig. 3: Loss (measured as difference between the achieved target precision and that of the virtual best solver, in log-performance space) of the logarithmic algorithm selection model and each of the five individual algorithms, for different budgets of the second part of the search. The thick black bar represents the mean loss for each method.

TABLE IV: Confusion matrix for our algorithm selector for *A2* budget 900. Total number is less than 1,200 because we did not assign a confusion when more then one algorithm different from the selected one had (equal) best performance.

	Selected algorithm					
True (single) best	BFGS	CMA-ES	DE	MLSL	PSO	**Total**
BFGS	487	44	1	1	2	**535**
CMA-ES	48	252	3	4	8	**315**
DE	11	18	1	3	1	**34**
MLSL	82	66	2	33	4	**187**
PSO	33	61	3	0	20	**117**
Total (1 188)	**661**	**441**	**10**	**41**	**35**	

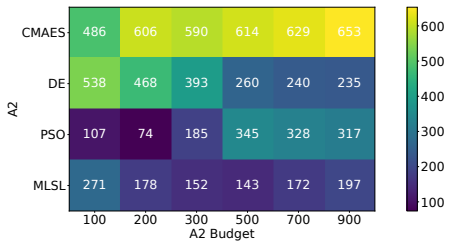
to select CMA-ES, etc.

Excluding BFGS. We have seen above that the settings are largely dominated by BFGS, which is the best among the five algorithms for 648 (*A2* budget 900) up to 882 (*A2* budget 100) out of the 1 200 runs. We therefore analyze how the results change if we exclude BFGS from our portfolio. Note that we do not need to retrain our regression models for this setup, as they were trained for each algorithm individually.

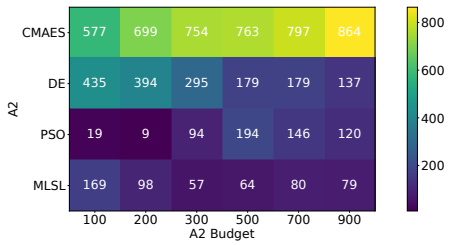
Fig. 4a summarizes how often each of the four algorithms is best (out of the same 1 200). We see that CMA-ES is now the dominating algorithm, however, to a much lesser extent as BFGS dominated the full portfolio. The algorithms selected by our algorithm selector seem to be equally balanced as the number of runs in which they are optimal. Note, though, that MLSL, DE, and PSO are selected much less often than the number of cases in which they are optimal suggests. That is, our algorithm selector often chooses CMA-ES. To evaluate the impact on the overall loss, we created again boxplots as in Fig. 3, for the same six *A2* budgets as studied in the case with BFGS. Fig. 5 shows the results for *A2* budgets 200 (left) and 900 (right). The results are very similar for all other *A2* budgets: the loss of the CMA-ES is best among all four budgets, but the selector is better both in terms of mean performance (e.g., 0.14 vs. 0.17 for *A2* budget 200 and 0.21 vs. 0.45 for *A2* budget 900) and with respect to the 75% percentile (0.13 vs. 0.16 for A2 budget 200 and 0.21 vs. 0.30 for *A2* budget 900, respectively; the median is 0 for both the CMA-ES and the selector for most cases).

V. Limitations of Our Approach

There are several limitations in our approach. First, we investigate only 10 runs per each algorithm on each problem instance and the results are computed on a per-run basis. In particular, an algorithm that happens to underperform in this

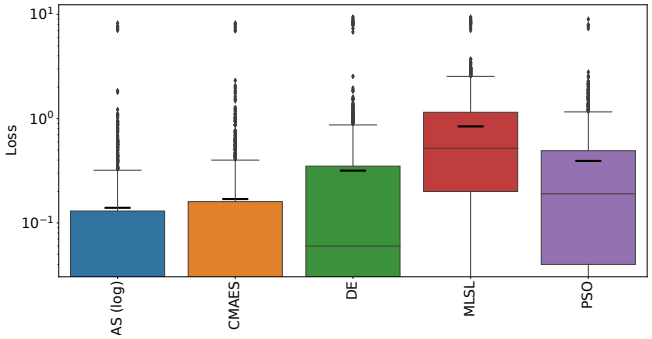


(a) How often each solver, *excluding BFGS*, is the best to switch to.

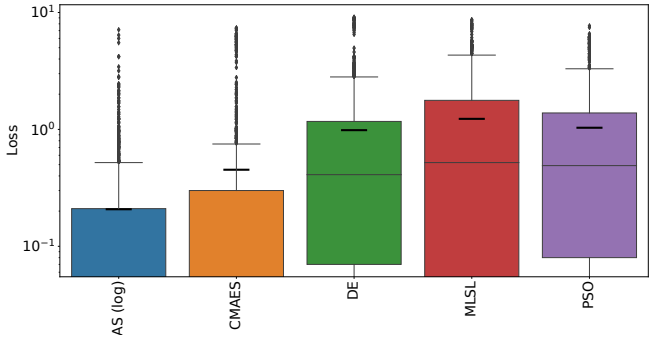


(b) How often each solver, *excluding BFGS*, is actually selected.

Fig. 4: Heatmaps showing in how many (out of 1 200) runs each algorithm (excluding BFGS) is the best one to switch to (left) and is selected to switch to by the logarithmic model (right), based on the amount of budget given to this second part of the search. Results are capped at 10^{-8} target precision, the number of best algorithms per budget can therefore be larger than 1 200.



(a) Budget 200



(b) Budget 900

Fig. 5: Loss (measured as difference between the achieved target precision and that of the virtual best solver, in log performance space) of the logarithmic algorithm selection model when excluding BFGS, and each of the 4 remaining individual algorithms, for different budgets of the second part of the search. The thick black bar represents the mean loss for each method.

particular run is assigned a large loss in our evaluation, even though its "typical" performance for the same setting may be much better than what the results of that one particular run suggests. This can of course also happen the other way around, i.e., an algorithm may appear to be much better than its "typical" performance. While we think that the overall large number of runs considered in our work helps to average out such unwanted outlier effects, a more robust experimental setup should be considered for future work.

In addition to this, we should note that the used warm-starting techniques are quite straightforward. While this is useful for dynamic algorithm selection in general, we could also extend the warm-starting procedure to better utilize the available information. This should lead to better overall performance of the switching algorithms. Furthermore, to this end, each of the considered algorithms require some level of warm-starting customization, which results in different warm-starting procedures being applied depending on the algorithm. Additional effort merits to be put towards defining a universal warm-starting procedure that can be employed independently of the algorithm's internal operating mechanism.

VI. Conclusions and Future Work

We have shown that the trajectory-based selection is able to outperform all of the individual algorithms in this portfolio, given that there is sufficient complementarity in their performance. Since our experimental pipeline makes use of a relatively small number of samples to determine the algorithm to switch to, without considering any algorithm-specific state features, it highlights the potential of the overall approach.

Going forward, we will extend our work to settings in which a proper transfer of learned regression models needs to be performed. To this end, we will consider a transfer to the benchmark collections from the CEC competitions [38]–[41] and to the (artificial and real-world) problem suits available in `nevergrad` [42]. We also plan on extending our approach towards larger algorithm portfolios. Specifically, it would be good to focus on a portfolio which contains complementary algorithms, which show varying behavior on different problem instances. In addition, more research is required to define suitable ways to warm-start the algorithms with the information gathered by the first algorithm.

As mentioned in Sec. III, we recorded several state variables of the CMA-ES, but we did not make use of them in this present study. We believe that the regression models can

strongly benefit from this information; possibly not in the naïve way applied in [5] (where only the final state variables at the time of the switch were used as features for the regression model), but by extracting information from the *evolution* of the state variables during the first part of the optimization process, before the switch. Such an approach based on time-series analysis have been suggested in the literature [43]. There, it was shown that features computed on evolution of the state variables of the CMA-ES can be used to accurately classify variants of the algorithm, and predict which of the BBOB problems was being optimized. Combining such an approach with the algorithm selection methodology presented in this work would be a promising direction of research. In addition, approaches with recurrent neural networks [44] (i.e., long short-term memory) and transformers [45] for predicting from longitudinal trajectory data should be considered to enrich the performance regression, which is a key component of our algorithm selection pipeline.

Finally, an adaptive switching policy (as opposed to switching after a fixed number of evaluations as investigated in this present paper) is another important direction towards practical applicability and adoption of our trajectory-based landscape-aware algorithm selection approach.

REFERENCES

[1] J. R. Rice, "The algorithm selection problem." *Advances in Computers*, vol. 15, pp. 65–118, 1976.

[2] P. Kerschke and H. Trautmann, "Automated algorithm selection on continuous black-box problems by combining exploratory landscape analysis and machine learning," *ECJ*, vol. 27, pp. 99–127, 2019.

[3] M. A. Muñoz, Y. Sun, M. Kirley, and S. K. Halgamuge, "Algorithm selection for black-box continuous optimization problems: A survey on methods and challenges," *Inform. Sciences*, vol. 317, pp. 224–245, 2015.

[4] P. Kerschke, H. H. Hoos, F. Neumann, and H. Trautmann, "Automated algorithm selection: Survey and perspectives," *ECJ*, vol. 27, 2019.

[5] A. Jankovic, T. Eftimov, and C. Doerr, "Towards feature-based performance regression using trajectory data," in *EvoApplications'21*. Springer, 2021, pp. 601–617.

[6] M. López-Ibáñez, J. Branke, and L. Paquete, "Reproducibility in evolutionary computation," *TELO*, vol. 1, no. 4, pp. 14:1–14:21, 2021.

[7] A. Jankovic, D. Vermetten, A. Kostovska, J. de Nobel, T. Eftimov, and C. Doerr, "Trajectory-based algorithm selection with warmstarting - reproducibility," Feb 2022. [Online]. Available: https://figshare.com/s/eaea33f0023891b4c60e

[8] D. Vermetten, S. van Rijn, T. Bäck, and C. Doerr, "Online selection of cma-es variants," in *GECCO'19*. ACM, 2019, pp. 951–959.

[9] S. van Rijn, "Modular CMA-ES framework from [31], v0.3.0," https://github.com/sjvrijn/ModEA. Available also as pypi package at https://pypi.org/project/ModEA/0.3.0/, 2018.

[10] N. Hansen, A. Auger, R. Ros, O. Mersmann, T. Tušar, and D. Brockhoff, "COCO: a platform for comparing continuous optimizers in a black-box setting," *Optimization Methods and Software*, vol. 36, pp. 1–31, 2020.

[11] C. Doerr, F. Ye, N. Horesh, H. Wang, O. M. Shir, and T. Bäck, "Benchmarking discrete optimization heuristics with iohprofiler," *Appl. Soft Comput.*, vol. 88, p. 106027, 2020.

[12] D. Vermetten, H. Wang, T. Bäck, and C. Doerr, "Towards dynamic algorithm selection for numerical black-box optimization: investigating BBOB as a use case," in *GECCO'20*. ACM, 2020, pp. 654–662.

[13] C. G. Broyden, "The convergence of a class of double-rank minimization algorithms," in *J. Inst. Math. Appl. 6*, 1970, pp. 76—-90.

[14] R. Fletcher, "A new approach to variable metric algorithms," in *Comp. J. 13*, 1970, pp. 317—-322.

[15] D. F. Goldfarb, "A family of variable-metric methods derived by variational means," in *Math. Comp. 24*, 1970, pp. 23—-26.

[16] D. Shanno, "Conditioning of quasi-newton methods for function minimization," in *Math. Comp. 24*, 1970, pp. 647—-656.

[17] N. Hansen and A. Ostermeier, "Completely derandomized self-adaptation in evolution strategies," *ECJ*, vol. 9, p. 159–195, 2001.

[18] R. Storn and K. Price, "Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces," *Journal of Global Optimization*, vol. 11, no. 4, pp. 341–359, 1997.

[19] A. H. G. Rinnooy Kan and G. T. Timmer, "Stochastic global optimization methods. part 1: Clustering methods," *Math. Program.*, 1987.

[20] A. Kan and G. Timmer, "Stochastic global optimization methods part ii: Multi level methods," *Mathematical Programming*, vol. 39, 1987.

[21] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *ICNN'95*.

[22] O. M. Shir and A. Yehudayoff, "On the covariance-hessian relation in evolution strategies," *Theor. Comput. Sci.*, vol. 801, pp. 157–174, 2020.

[23] B. Bischl, O. Mersmann, H. Trautmann, and M. Preuss, "Algorithm selection based on exploratory landscape analysis and cost-sensitive learning," in *GECCO'12*. ACM, 2012, pp. 313–320.

[24] R. Cosson, B. Derbel, A. Liefooghe, H. E. Aguirre, K. Tanaka, and Q. Zhang, "Decomposition-based multi-objective landscape features and automated algorithm selection," in *EvoCOP'21*. Springer, 2021.

[25] F. Hutter, L. Kotthoff, and J. Vanschoren, Eds., *Automated Machine Learning - Methods, Systems, Challenges*. Springer, 2019.

[26] M. Lindauer, H. H. Hoos, F. Hutter, and T. Schaub, "Autofolio: An automatically configured algorithm selector," *JAIR*, 2015.

[27] L. Xu, F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Evaluating component solver contributions to portfolio-based algorithm selectors," in *SAT'12*. Springer, 2012, pp. 228–241.

[28] M. Lindauer, J. N. van Rijn, and L. Kotthoff, "Open algorithm selection challenge 2017: Setup and scenarios," in *Open Algorithm Selection Challenge*, vol. 79. PMLR, 2017, pp. 1–7.

[29] O. Mersmann, B. Bischl, H. Trautmann, M. Preuss, C. Weihs, and G. Rudolph, "Exploratory Landscape Analysis," in *GECCO'11*.

[30] Q. Renau, C. Doerr, J. Dreo, and B. Doerr, "Exploratory landscape analysis is strongly sensitive to the sampling strategy," in *PPSN'20*.

[31] S. van Rijn, H. Wang, M. van Leeuwen, and T. Bäck, "Evolving the structure of Evolution Strategies," in *SSCI'16*.

[32] J. de Nobel, D. Vermetten, H. Wang, C. Doerr, and T. Bäck, "Tuning as a means of assessing the benefits of new ideas in interplay with existing algorithmic modules," in *GECCO'21, Companion*.

[33] P. Virtanen, et al., "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," *Nature Methods*, vol. 17, pp. 261–272, 2020.

[34] L. Pál, "Benchmarking a hybrid multi level single linkageal gorithm on the bbob noiseless testbed," in *GECCO'13*, 2013, pp. 1145–1152.

[35] C. Doerr, H. Wang, F. Ye, S. van Rijn, and T. Bäck, "IOHprofiler: A benchmarking and profiling tool for iterative optimization heuristics," *arXiv e-prints:1810.05281*, 2018.

[36] H. Wang, D. Vermetten, F. Ye, C. Doerr, and T. Bäck, "IOHanalyzer: Detailed performance analysis for iterative optimization heuristic," *ACM Trans. Evol. Learn. Optim.*, 2022, to appear.

[37] F. Pedregosa, *et al.*, "Scikit-learn: Machine learning in python," *JMLR*, vol. 12, pp. 2825–2830, 2011.

[38] J. Liang, B. Qu, P. Suganthan, and A. Hernández-Díaz, "Problem definitions and evaluation criteria for the CEC 2013 special session on real-parameter optimization," 2013.

[39] J. Liang, B. Qu, and P. Suganthan, "Problem definitions and evaluation criteria for the CEC 2014 special session and competition on single objective real-parameter numerical optimization," 2013.

[40] J. Liang, B. Qu, P. Suganthan, and Q. Chen, "Problem definitions and evaluation criteria for the CE 2015 competition on learning-based real-parameter single objective optimization," 2014.

[41] G. Wu, R. Mallipeddi, and P. Suganthan, "Problem definitions and evaluation criteria for the CEC 2017 competition and special session on constrained single objective real-parameter optimization," 2016.

[42] J. Rapin and O. Teytaud, "Nevergrad - A gradient-free optimization platform," https://GitHub.com/FacebookResearch/Nevergrad, 2018.

[43] J. de Nobel, H. Wang, and T. Bäck, "Explorative data analysis of time series based algorithm features of CMA-ES variants," in *GECCO'21*.

[44] G. Uribarri and G. B. Mindlin, "Dynamical time series embeddings in recurrent neural networks," *Chaos, Solitons & Fractals*, 2022.

[45] G. Woo, C. Liu, D. Sahoo, A. Kumar, and S. Hoi, "Etsformer: Exponential smoothing transformers for time-series forecasting," *arXiv:2202.01381*, 2022.

[46] P. Kerschke and H. Trautmann, "Comprehensive Feature-Based Landscape Analysis of Continuous and Constrained Optimization Problems Using the R-Package Flacco," *https://github.com/kerschke/flacco*, 2019.