Per-run Algorithm Selection with Warm-starting using Trajectory-based Features

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

Per-run Algorithm Selection with Warm-starting using Trajectory-based Features

Ana Kostovska[1,2] , Anja Jankovic[3] , Diederick Vermetten[4] ,

Jacob de Nobel[4], Hao Wang[4] , Tome Eftimov[1] , Carola Doerr[3]

[1] Jožef Stefan Institute, Ljubljana, Slovenia
[2] Jožef Stefan International Postgraduate School, Ljubljana, Slovenia
[3] LIP6, Sorbonne Université, CNRS, Paris, France
[4] LIACS, Leiden University, Leiden, The Netherlands

Abstract. Per-instance algorithm selection seeks to recommend, for a given problem instance and a given performance criterion, one or several suitable algorithms that are expected to perform well for the particular setting. The selection is classically done offline, using openly available information about the problem instance or features that are extracted from the instance during a dedicated feature extraction step. This ignores valuable information that the algorithms accumulate during the optimization process.

In this work, we propose an alternative, online algorithm selection scheme which we coin as "per-run" algorithm selection. In our approach, we start the optimization with a default algorithm, and, after a certain number of iterations, extract instance features from the observed trajectory of this initial optimizer to determine whether to switch to another optimizer. We test this approach using the CMA-ES as the default solver, and a portfolio of six different optimizers as potential algorithms to switch to. In contrast to other recent work on online per-run algorithm selection, we warm-start the second optimizer using information accumulated during the first optimization phase. We show that our approach outperforms static per-instance algorithm selection. We also compare two different feature extraction principles, based on exploratory landscape analysis and time series analysis of the internal state variables of the CMA-ES, respectively. We show that a combination of both feature sets provides the most accurate recommendations for our test cases, taken from the BBOB function suite from the COCO platform and the YABBOB suite from the Nevergrad platform.

Keywords: Algorithm Selection · Black-Box Optimization · Exploratory Landscape Analysis · Evolutionary Computation

1 Introduction

It is widely known that optimization problems are present in many areas of science and technology. A particular subset of these problems are the black-box

problems, for which a wide range of optimization algorithms has been developed. However, it is not always clear which algorithm is the most suitable one for a particular problem. Selecting which algorithm to use comes with its own cost and challenges, so the choice of an appropriate algorithm poses a meta-optimization problem that has itself become an increasingly important area of study.

Moreover, a user needs to be able to select different algorithms for different *instances* of the same problem, which is a scenario that very well reflects real-world conditions. This **per-instance algorithm selection** most often relies on being able to compute a set of features which capture the relevant properties of the problem instance at hand. A popular approach is the *landscape-aware* algorithm selection, where the problem features' definition stems from the field of *exploratory landscape analysis* (ELA) [25]. In this approach, an initial set of points is sampled and evaluated on the problem instance to identify its global properties. However, this induces a significant overhead cost to the algorithm selection procedure, since the initial sample of points used to extract knowledge from the problem instance is usually not directly used by the chosen algorithm in the subsequent optimization process.

Previous research into landscape-aware algorithm selection suggests that, as opposed to creating a separate set of samples to compute ELA features in a dedicated preprocessing step, one could use the samples observed by some initial optimization algorithm. This way, the algorithm selection changes from being a purely offline procedure into being one which considers whether or not to switch between different algorithms during the search procedure. This is an important step towards dynamic (online) algorithm selection, in which the selector is able to track and adapt the choice of the algorithm throughout the optimization process in an intelligent manner.

In this paper, we coin the term **per-run algorithm selection** to refer to the case where we make use of information gained by running an initial optimization algorithm ($A1$) during a single run to determine which algorithm should be selected for the remainder of the search. This second algorithm ($A2$) can then be warm-started, i.e., initialized appropriately using the knowledge of the first one. The pipeline of the approach is shown in Fig. 1.

Following promising results from [15], in this work we apply our *trajectory-based algorithm selection* approach to a broader set of algorithms and problems. To extract relevant information about the problem instances, we rely on ELA features computed using samples and evaluations observed by the initial algorithm's search trajectory, i.e., *local* landscape features. Intuitively, we consider the problem instance as perceived from the algorithm's viewpoint.

In addition, we make use of an alternative aspect that seems to capture critical information during the search procedure – the algorithm's internal state, quantified through a set of state variables at every iteration of the initial algorithm. To this end, we choose to track their evolution during the search by computing their corresponding *time-series* features.

Using the aforementioned values to characterize problem instances, we build algorithm selection models based on the prediction of the fixed-budget perfor-

mance of the second solver on those instances, for different budgets of function evaluations. We train and test our algorithm selectors on the well-known BBOB problem collection of the COCO platform [11], and extend the testing on the YABBOB collection of the Nevergrad platform [28]. We show that our approach leads to promising results with respect to the selection accuracy and we also point out interesting observations about the particularities of the approach.

State of the Art. Given an optimization problem, a specific instance of that problem which needs to be solved, and a set of algorithms which can be used to solve it, the so-called per-instance algorithm selection problem arises. How does one 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 [2, 5, 13, 21, 23, 36]. All of these deal predominantly with offline AS. An effort towards online AS has been recently proposed [24], where the switching rules between algorithms were defined based on non-convex ratio features extracted during the optimization process. However, this particular study is not based on using supervised machine learning techniques to define the switching rule, which is the key difference presented in our approach.

Paper Outline. In Section 2, we introduce the problem collections and the algorithm portfolio, and give details about the raw data generation for our experiments. The full experimental pipeline is more closely presented in Section 3. We discuss the main results on two benchmark collections in Sections 4 and 5, respectively. Finally, Section 6 gives several possible avenues for future work.

Data and Code Availability. Our source code, raw data, intermediate artefacts and analysis scripts have been made available on our Zenodo repository [17]. In this paper, we highlight only selected results for reasons of space.

2   Data Collection

Problem Instance Portfolio. To implement and verify our proposed approach, we make use of a set of black-box, single-objective, noiseless problems. The data set is the BBOB suite from the COCO platform [11], which is a very common benchmark set within numerical optimization community. This suite consists of a total of 24 functions, and each of these functions can be changed by applying pre-defined transformations to both its domain and objective space, resulting in a set of different instances of each of these problems that share the same global characteristics [10].

Another considered benchmark set is the YABBOB suite from the Nevergrad platform [28], that contains 21 black-box functions, out of which we keep 17 for this paper. By definition, YABBOB problems do not allow for generating different instances.

Algorithm Portfolio. As our algorithm portfolio, we consider the one used in [18,32]. This gives us a set of 5 black-box optimization algorithms: MLSL [19, 31], BFGS [3,8,9,33], PSO [20], DE [34] and CMA-ES [12]. Since for the CMA-ES
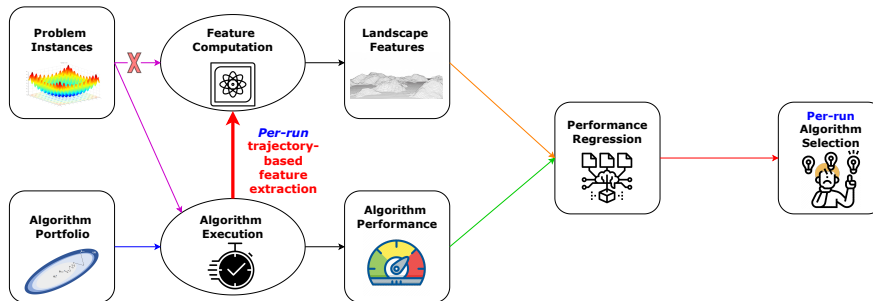
Fig. 1: Per-run algorithm selection pipeline. The overhead cost of computing ELA features per problem instance is circumvented via collecting information about the instance during the default optimization algorithm run.

we consider two versions from the modular CMA-ES framework [29] (elitist and non-elitist), this gives us a total portfolio of 6 algorithm variants. The implementation of the algorithms used can be found in more detail in our repository [17].

Warm-starting. To ensure we can switch from our initial algorithm (A1) to any of the others (A2), we make use of a basic warm-starting approach specific to each algorithm. For the two versions of modular CMA-ES, we do not need to explicitly warm-start, since we can just continue the run with the same internal parameters and turn on elitist selection if required. The detailed warm-start mechanisms are discussed in [18], and the implementations are available in our repository [17].

Performance Data. For our experiments, we consider a number of data collection settings, based on the combinations of dimensionality of the problem, where we use both 5- and 10-dimensional versions of the benchmark functions, and budget for A1, where we use $30 \cdot D$ budget for the initial algorithm. This is then repeated for all functions of both the BBOB and the YABBOB suite. For BBOB, we collect 100 runs on each of the first 10 instances, resulting in 1 000 runs per function. For YABBOB (only used for testing), we collect 50 runs on each function (as there are no instances in Nevergrad).

We show the performance of the six algorithms in our portfolio in the 5-dimensional case in Figure 2. Since the A1 budget is $30 \cdot D = 150$, the initial part of the search is the same for all algorithms until this point. In the figure, we can see that, for some functions, clear differences in performance between the algorithm appear very quickly, while for other functions the difference only becomes apparent after some more evaluations are used. This difference leads us to perform our experiments with three budgets for the A2 algorithm, namely $20 \cdot D$, $70 \cdot D$ and $170 \cdot D$. To highlight the differences between the algorithms for each of these scenarios, we can show in what fraction of runs each algorithm performs best. This is visualized in Figure 3. From this figure, we can see that while some algorithm are clearly more impactful than others, the differences between them are still significant. This indicates that there would be a significant difference between

a virtual best solver which selects the best algorithm for each run and a single best solver which uses only one algorithm for every run.
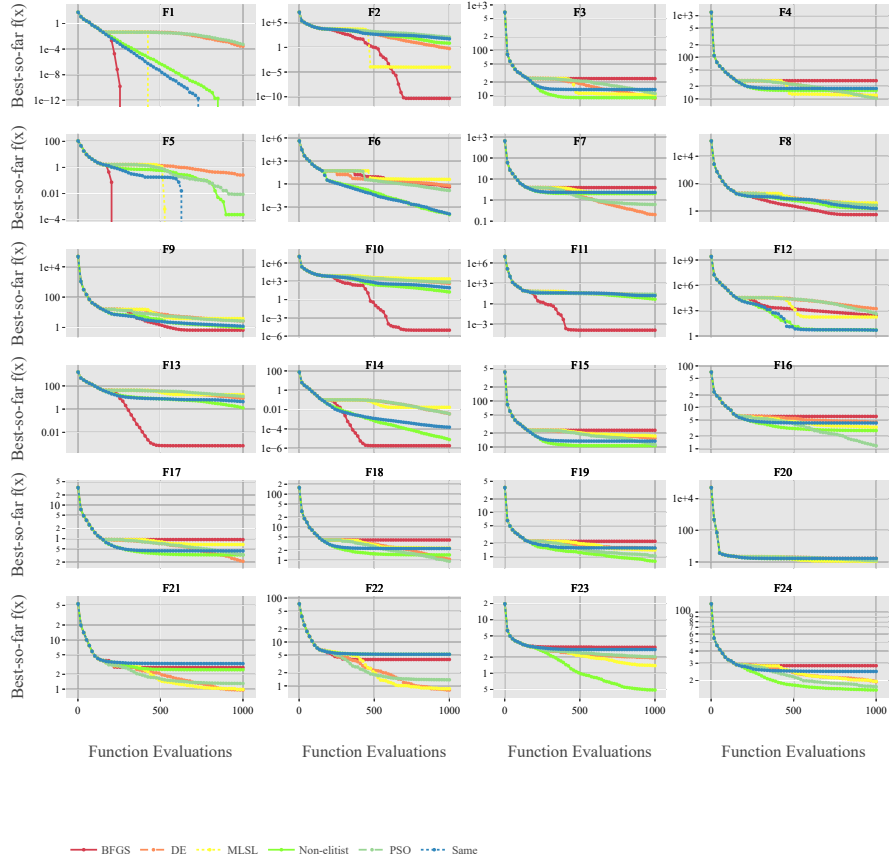
Fig. 2: Mean best-so-far function value (precision to global optimum) for each of the six algorithms in the portfolio. For computational reasons, each line is calculated based on a subset of 10 runs on each of the 10 instances used, for a total of 100 runs. Note that the first 150 evaluations for each algorithm are identical, since this is the budget used for A1. Figure generated using IOHanalyzer [35].

3 Experimental Setup

Adaptive Exploratory Landscape Analysis. As previously discussed, the *per-run* trajectory-based algorithm selection method consists of extracting ELA features from the search trajectory samples during a single run of the initial solver. A vector of numerical ELA feature values is assigned to each run on the

Fig. 3: Matrix showing for each scenario (with respect to the dimensionality and A2 budget) in what proportion of runs each algorithm reaches the best function value. Note that these value per scenario can add to more than 1 because of ties.

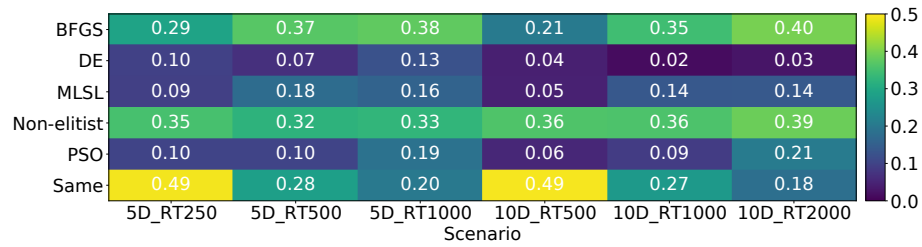problem instance, and can be then used to train a predictive model that maps it to different algorithms' performances on the said run. To this end, we use the ELA computation library named FLACCO [22].

Among over 300 different features (grouped in feature sets) available in FLACCO, we only consider features that do not require additional function evaluations for their computation, also referred to as *cheap features* [1]. They are computed using the fixed initial sample, while *expensive features*, in contrast, need additional sampling during the run, an overhead that makes them more inaccessible for practical use. For the purpose of this work, as suggested in preliminary studies [15, 18], we use 38 cheap features most commonly used in the literature, namely those from *y-Distribution*, *Levelset*, *Meta-Model*, *Dispersion*, *Information Content* and *Nearest-Better Clustering* feature sets.

We perform this per-run feature extraction using the initial $A1 = 30 \cdot D$ budget of samples and their evaluations per each run of each of the first 10 instances of each of the 24 BBOB problems, as well aas 17 YABBOB problems (that have no instances) in $5D$ and $10D$.

Time-Series Features. In addition to ELA features computed during the optimization process, we consider an alternative – *time-series* features of the internal states of the CMA-ES algorithm. Since the internal variables of an algorithm are adapted during the optimization, they could potentially contain useful information about the current state of the optimization. Specifically, we consider the following internal variables: the step size σ, the eigenvalues of covariance matrix $\vec{v}$, the evolution path $\vec{p}_c$ and its conjugate $\vec{p}_\sigma$, the Mahalanobis distances from each search point to the center of the sampling distribution $\vec{\gamma}$, and the log-likelihood of the sampling model $\mathcal{L}\left(\vec{m}, \sigma^2, \mathbf{C}\right)$. We consider these dynamic strategy parameters of the CMA-ES as a multivariate real-valued time series, for which at every iteration of the algorithm, we compute one data point of the time series as follows: $\forall t \in [L]$:

$$\vec{\psi}_t := \left(\sigma, \mathcal{L}(\vec{m}, \sigma^2, \mathbf{C}), ||\vec{v}||, ||\vec{p}_\sigma||, ||\vec{p}_c||, ||\vec{\gamma}||, \text{ave}(\vec{v}), \text{ave}(\vec{p}_\sigma), \text{ave}(\vec{p}_c), \text{ave}(\vec{\gamma})\right)^\top,$$

where L represents the number of iterations these data points were sampled, which equals the A1 budget divided by the population size of the CMA-ES. In order to store information invariant to the problem dimension, we compute the

component-wise average ave($\cdot$) and norm $||\vec{x}|| = \sqrt{\vec{x}^\top \vec{v}}$ of each vector variable. Given a set of m feature functions $\{\phi_i\}_{i=1}^m$ from TSFRESH (where $\phi_i \colon \mathbb{R}^L \to \mathbb{R}$), we apply each feature function over each variable in the collected time series. Examples of such feature functions are autocorrelation, energy and continuous wavelet transform coefficients. In this paper, we take this entire time series (of length L) as the feature window. We employ all 74 feature functions from the TS-FRESH library [4], to compute a total of $9\,444$ time-series features per run. After the feature generation, we perform a feature selection method using a Random Forests classifier trained to predict the function ID, for computing the feature importance. We then select only the features whose importance is larger than 2×10^{-3}. This selection procedure yields 129 features, among which features computed on the Mahalanobis distance and the step-size σ are dominant. More details on this approach can be found in [26].

Regression Models. To predict the algorithm performance after the A2 budget, we use as performance metric the target precision reached by the algorithm in the fixed-budget context (i.e., after some fixed number of function evaluations). We create a mapping between the input feature data, which can be one of the following: (1) the trajectory-based representation with 38 ELA features per run (ELA-based AS), (2) the trajectory-based representation with 129 time-series (TS) features per run (TS-based AS), or (3) a combination of both (ELA+TS-based AS)

and the target precision of different algorithm runs. We then train supervised machine learning (ML) regression models that are able to predict target precision for different algorithms on each of the trajectories involved in the training data. Following some strong insights from [14] and subsequent studies, we aim at predicting the logarithm ($log10$) of the target precision, in order to capture the order of magnitude of the distance from the optimum. In our case, since we are dealing with an algorithm portfolio, we have trained a separate single target regression (STR) model for each algorithm involved in our portfolio. As an ML regression method, we opt for using a random forest (RF), as previous studies have shown that they can provide promising results for automated algorithm performance prediction [16]. To this end, we use the RF implementation from the Python package SCIKIT-LEARN [27].

Evaluation Scenarios. To find the best RF hyperparameters and to evaluate the performance of the algorithm selectors, we have investigated two evaluation scenarios:

(1) Leave-instance out validation: in this scenario, 70% of the instances from each of the 24 BBOB problems are randomly selected for training and 30% are selected for testing. Put differently, all 100 runs for the selected instance will either appear in the train or the test set. We thus end up with $16\,800$ trajectories used for training and $7\,200$ trajectories for testing.

(2) Leave-run out validation: in this scenario, 70% of the runs from each BBOB problem instance are randomly selected for training and 30% are selected for testing. Again, we end up with $16\,800$ trajectories used for training and $7\,200$ trajectories for testing.

Table 1: RF hyperparameter names and their corresponding values considered in the grid search.

Hyperparameter	Search space
n_estimators	$[100, 300]$
max_features	$[\text{AUTO}, \text{SQRT}, \text{LOG2}]$
max_depth	$[3, 5, 15, \text{NONE}]$
min_samples_split	$[2, 5, 10]$

We repeat each evaluation scenario five independent times, in order to analyze the robustness of the results. Each time, the training data set was used to find the best RF hyperparameters, while the test set was used only for evaluation of the algorithm selector.

Hyperparameter Tuning for the Regression Models. The best hyperparameters are selected for each RF model via grid search for a combination of an algorithm and a fixed A2 budget. The training set for finding the best RF hyperparameters for each combination of algorithm and budget is the same. Four different RF hyperparameters are selected for tuning: (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, and (4) *min_samples_split*: the minimum number of samples required for splitting an internal node in the tree. The search spaces of the hyperparameters for each RF model utilized in our study are presented in Table 1.

Per-run Algorithm Selection. In real-world dynamic AS applications, we rely on the information obtained within the current run of the initial solver on a particular problem instance to make our decision to switch to a better suited algorithm. A randomized component of black-box algorithms comes into play here, as one algorithm's performance can vastly differ from one run to another on the very same problem instance.

We estimate the quality of our algorithm selectors by comparing them to standard baselines, the virtual best solver (VBS) and the single best solver (SBS). As we make a clear distinction between per-run and per-instance perspective, in order to compare we need to suitably aggregate the results. Our baseline is the *per-run VBS*, which is the selector that always chooses the real best algorithm for a particular run on a certain problem (i.e., function) instance. We then define VBS_{iid} and VBS_{fid} as virtual best solvers on instance and problem levels, i.e., selectors that always pick the real best algorithm for a certain instance (across all runs) or a certain problem (across all instances). Last, we define the SBS as the algorithm that is most often the best one across all runs.

For each of these methods, we can define their performance relative to the *per-run VBS* by considering their performance ratio, which is defined on each run as taking the function value achieved by the VBS and dividing it by the value reach by the considered selector. As such, the performance ratio for the *per-run VBS* is 1 by definition, and in $[0, 1]$ for each other algorithm selector.
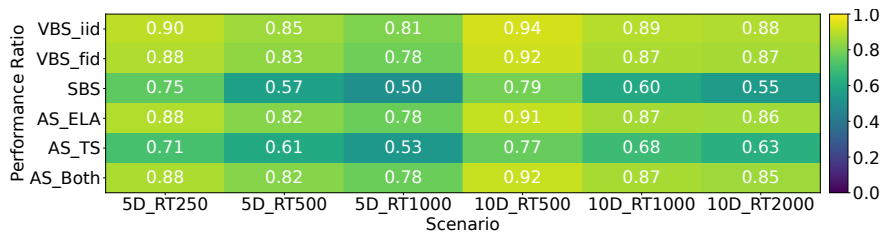
Fig. 4: Heatmap showing for each scenario the average performance ratio relative to the per-run virtual best solver of different versions of VBS, SBS and algorithm selectors (based on the per-instance folds). Scenario names show the problem dimensionality and the total used budget.

To measure the performance ratio for the algorithm selectors themselves, we calculate this performance ratio on every run in the test-set of each of the 5 folds, and average these values. We point out here that the performance of different AS models are not statistically compared, since the obtained performance values from the folds are not independent [6].

4 Evaluation Results: COCO

For our first set of experiments, we train our algorithm selectors on BBOB functions using the evaluation method described in Section 3. Since we consider 2 dimensionalities of problems and 3 different A2 budgets, we have a total of 6 scenarios for each of the 3 algorithm selectors (ELA-, TS-, and ELA+TS-based). In Figure 4, we show the performance ratios of these selectors, as well as the performance ratios of the previously described VBS and SBS baselines. Note that for this figure, we make use of the *instance-run* folds, but results are almost identical for the *per-run* case.

Based on Figure 4, we can see that the ELA-based algorithm selector performs almost as well as the per-function VBS, which itself shows only minor performance differences to the per-instance VBS. We also notice that as the total evaluation budget increases, the performance of every selector deteriorates. This seems to indicate that as the total budget becomes larger, there are more cases where runs on the same instance have different optimal switches.

To study the performance of the algorithm selectors in more detail, we can consider the performance ratios for each function separately, as is visualized in Figure 5. From this figure, we can see that for the functions where there is a clearly optimal A2, all algorithm selectors are able to achieve near-optimal performance. However, for the cases where the optimal A2 is more variable, the discrepancy between the ELA and TS-based algorithm selectors increases.
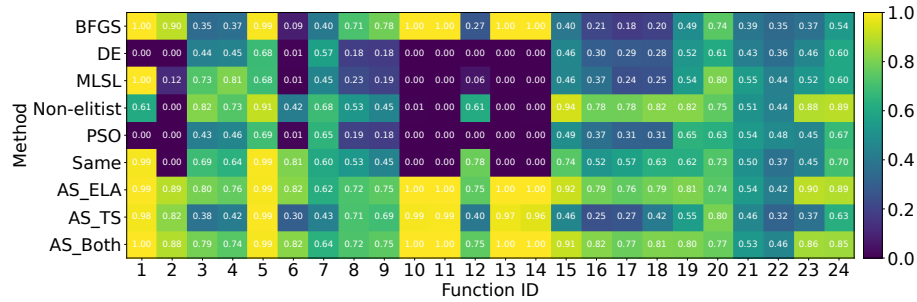
Fig. 5: Heatmap showing for each 5-dimensional BBOB function the mean performance ratio at 500 total evaluations relative to the per-run virtual best solver, as well as the average performance ratio of each of the 3 algorithm selectors.

5   Evaluation Results: Nevergrad

We now study how a model trained on BBOB problem trajectories can be used to predict the performances on trajectories not included in the training. We do so by considering the YABBOB suite from the Nevergrad platform. While there is some overlap between these two problem collections, introducing another sufficiently different validation/test suite allows us to verify the stability of our algorithm selection models. We recall that for the performance data of the same algorithm portfolio on YABBOB functions, we have target precisions for 850 runs, 50 runs per 17 problems, in all considered A2 budgets.

Training on COCO, testing on Nevergrad. This experiment has resulted in somewhat poorer performance of the algorithm selection models on an inherently different batch of problems. The comparison of the similarity between BBOB and YABBOB problems presented below nicely shows how the YABBOB problems are structurally more similar to one another than to the BBOB ones. To investigate performance flaws of our approach when testing on Nevergrad, we compare, for each YABBOB problem, how often a particular algorithm is selected by the algorithm selection model trained on the BBOB data with how often that algorithm was actually the best one. This comparison is exhibited in Figure 6. We observe that MLSL in particular is not selected often enough in the case of a large A2 budget, as well as a somewhat strong preference of the selector towards BFGS. An explanation for these results may be the (dis)similarities between the benchmarks. Only for some YABBOB functions in the second half of the set we might have similarities in the trajectories already seen from the second half of the BBOB data, but this is anecdotal as the overall tendency is that there are few parallels between BBOB and YABBOB.

Analyzing the complementarity between the COCO and Nevergrad suites. We illustrate the intra-similarity among the YABBOB test trajectories from the Nevergrad suite, which are not part of the training data set. This is shown via correlation between the YABBOB trajectories (test data) and the BBOB trajectories (train data). For this purpose, we first find the subspace
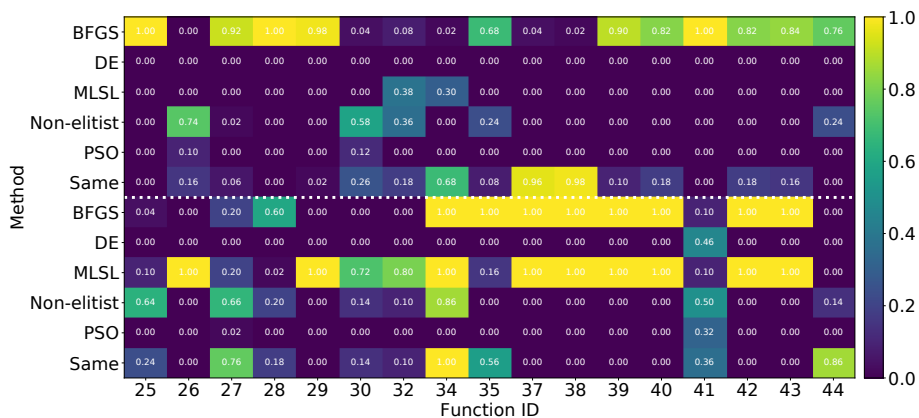
Fig. 6: Heatmap showing for each 5-dimensional YABBOB/Nevergrad function the fraction of times each algorithm was optimal to switch to when considering a total budget of 500 evaluations (bottom) and how often each of these algorithm was selected by the algorithm selector trained on BBOB/COCO (top). Note that the columns of the bottom part can sum to more than 1 in case of ties.

that is covered by the training trajectories, where we then project the testing trajectories. To find the subspace that is covered by training data, we apply singular value decomposition (SVD), following an approach presented in [7]. For the training and testing data, we summarize the trajectories on a problem level using the median values for each ELA feature by using all trajectory instances that belong to the same problem. Next, we map the BBOB trajectories to a linear vector space they cover (found by the SVD decomposition), where the trajectories are represented in different uncorrelated dimensions of the data. We then project each of the YABBOB trajectories to the linear subspace that is covered by the 24 BBOB problems, which allows us to find their correlation.

The Pearson correlation values between the trajectories obtained for $5D$ and $10D$ problem instances are showcased in Figure 7. We opt for the Pearson correlation coefficient since the trajectories are projected in a linear subspace. The trajectories from 1 to 24 correspond to the BBOB suite, and the trajectories starting from 25 to 44 correspond to the YABBOB suite. It is important to recall here that the YABBOB problems F31, F33, F36 and F45 were omitted from further analysis due to missing values. This figure shows that the BBOB trajectories are not correlated (the white square portion of the lower left part of the heatmap), which confirms high diversity in the training trajectory portfolio. However, there are lower positive and negative correlations between BBOB and YABBOB trajectories, which indicate that the properties of the YABBOB trajectories are not captured in the training data. This might be a possible explanation for the poor performance for the algorithm selection models which is trained on the BBOB trajectories, but only tested on the YABBOB trajectories.
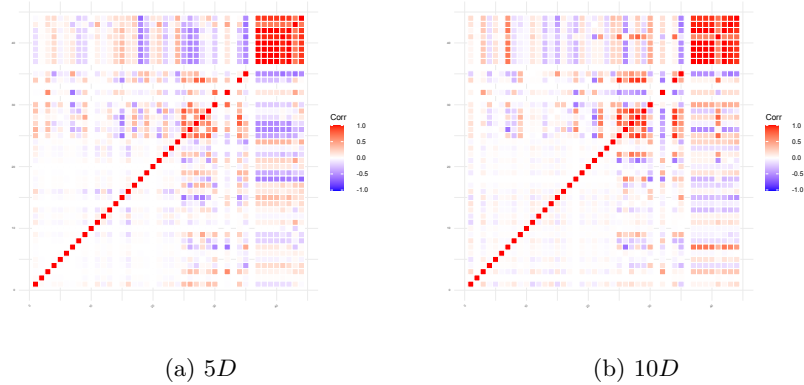
(a) $5D$ (b) $10D$

Fig. 7: Pearson correlation between BBOB (lower left portion) and YABBOB (bright red upper right corner) trajectories for $5D$ and $10D$.

6   Conclusions and Future Work

We have shown the feasibility of building an algorithm selector based on a very limited amount of samples from an initial optimization algorithm. Results within the BBOB benchmark suite show performance comparable to the per-function virtual best solver when using a selector based on ELA features. While these results did not directly transfer to other benchmark suites, this seems largely caused by the relatively low similarity between the collections.

Since this work is based on warm-starting the algorithms using the information of the initial search trajectory, further improvement in warm-starting would be highly beneficial to the overall performance of this feature-based selection mechanism. In addition, identifying exactly what features contribute to the decisions being made can show us what properties might be important to the performance of the switching algorithm, which in turn can support the development of better warm-starting mechanisms.

While the time-series based approach did not perform as well as the one based on ELA, it still poses an interesting avenue for future research. In particular, it would be worthwhile to consider the combined model in more detail, and aim to identify the level of complementarity between landscape and algorithm state features, which would help gain insight into the complex interplay between problems and algorithms.

References

1. Belkhir, N., Dréo, J., Savéant, P., Schoenauer, M.: Per instance algorithm configuration of CMA-ES with limited budget. In: Proc. of Genetic and Evolutionary Computation (GECCO'17). pp. 681–688. ACM (2017). `https://doi.org/10.1145/3071178.3071343`, `https://doi.org/10.1145/3071178.3071343`
2. Bischl, B., Mersmann, O., Trautmann, H., Preuss, M.: Algorithm selection based on exploratory landscape analysis and cost-sensitive learning. In: Proc. of Genetic and Evolutionary Computation Conference, GECCO'12. pp. 313–320. ACM (2012). `https://doi.org/10.1145/2330163.2330209`
3. Broyden, C.G.: The convergence of a class of double-rank minimization algorithms. In: J. Inst. Math. Appl. 6. pp. 76—-90 (1970)
4. Christ, M., Braun, N., Neuffer, J., Kempa-Liehr, A.W.: TSFRESH package for time-series feature engineering, `https://tsfresh.readthedocs.io/en/latest/text/list_of_features.html`
5. Cosson, R., Derbel, B., Liefooghe, A., Aguirre, H.E., Tanaka, K., Zhang, Q.: Decomposition-based multi-objective landscape features and automated algorithm selection. In: Proc. of Evolutionary Computation in Combinatorial Optimization (EvoCOP'21). LNCS, vol. 12692, pp. 34–50. Springer (2021). `https://doi.org/10.1007/978-3-030-72904-2_3`
6. Demšar, J.: Statistical comparisons of classifiers over multiple data sets. The Journal of Machine learning research **7**, 1–30 (2006)
7. Eftimov, T., Popovski, G., Renau, Q., Korošec, P., Doerr, C.: Linear matrix factorization embeddings for single-objective optimization landscapes. In: Proc. of IEEE Symposium Series on Computational Intelligence (SSCI'20). pp. 775–782. IEEE (2020). `https://doi.org/10.1109/SSCI47803.2020.9308180`
8. Fletcher, R.: A new approach to variable metric algorithms. In: Comp. J. 13. pp. 317—-322 (1970)
9. Goldfarb, D.F.: A family of variable-metric methods derived by variational means. In: Math. Comp. 24. pp. 23—-26 (1970)
10. Hansen, N., Auger, A., Finck, S., Ros, R.: Real-Parameter Black-Box Optimization Benchmarking: Experimental Setup. RR-7215, INRIA (2010)
11. Hansen, N., Auger, A., Ros, R., Mersmann, O., Tušar, T., Brockhoff, D.: COCO: a platform for comparing continuous optimizers in a black-box setting. Optimization Methods and Software **36**, 1–31 (2020). `https://doi.org/10.1080/10556788.2020.1808977`
12. Hansen, N., Ostermeier, A.: Completely derandomized self-adaptation in evolution strategies. Evol. Comput. **9**(2), 159–195 (Jun 2001). `https://doi.org/10.1162/106365601750190398`, `https://doi.org/10.1162/106365601750190398`
13. Hutter, F., Kotthoff, L., Vanschoren, J. (eds.): Automated Machine Learning - Methods, Systems, Challenges. The Springer Series on Challenges in Machine Learning, Springer (2019). `https://doi.org/10.1007/978-3-030-05318-5`
14. Jankovic, A., Doerr, C.: Landscape-aware fixed-budget performance regression and algorithm selection for modular CMA-ES variants. In: Proc. of Genetic and Evolutionary Computation Conference (GECCO'20). pp. 841–849. ACM (2020). `https://doi.org/10.1145/3377930.3390183`
15. Jankovic, A., Eftimov, T., Doerr, C.: Towards feature-based performance regression using trajectory data. In: Proc. of Applications of Evolutionary Computation (EvoApplications'21). LNCS, vol. 12694, pp. 601–617. Springer (2021). `https://doi.org/10.1007/978-3-030-72699-7_38`

16. Jankovic, A., Popovski, G., Eftimov, T., Doerr, C.: The impact of hyper-parameter tuning for landscape-aware performance regression and algorithm selection. In: Proc. of Genetic and Evolutionary Computation Conference (GECCO'21). pp. 687–696. ACM (2021). `https://doi.org/10.1145/3449639.3459406`
17. Jankovic, A., Kostovska, A., Vermetten, D., de Nobel, J., Wang, H., Eftimov, T., Doerr, C.: Per-Run Algorithm Selection with Warm-starting using Trajectory-based Features – Data (Apr 2022). `https://doi.org/10.5281/zenodo.6458266`
18. Jankovic, A., Vermetten, D., Kostovska, A., de Nobel, J., Eftimov, T., Doerr, C.: Trajectory-based algorithm selection with warm-starting (2022). `https://doi.org/10.48550/arxiv.2204.06397`
19. Kan, A., Timmer, G.: Stochastic global optimization methods part II: Multi level methods. Mathematical Programming **39**, 57–78 (09 1987). `https://doi.org/10.1007/BF02592071`
20. Kennedy, J., Eberhart, R.: Particle swarm optimization. In: Proc. of ICNN'95 - International Conference on Neural Networks. vol. 4, pp. 1942–1948 (1995). `https://doi.org/10.1109/ICNN.1995.488968`
21. Kerschke, P., Hoos, H.H., Neumann, F., Trautmann, H.: Automated algorithm selection: Survey and perspectives. Evolutionary Computation **27**(1), 3–45 (2019). `https://doi.org/10.1162/evco_a_00242`
22. Kerschke, P., Trautmann, H.: The R-package FLACCO for exploratory landscape analysis with applications to multi-objective optimization problems. In: CEC. pp. 5262–5269. IEEE (2016), `https://doi.org/10.1109/CEC.2016.7748359`
23. Lindauer, M., Hoos, H.H., Hutter, F., Schaub, T.: Autofolio: An automatically configured algorithm selector. Journal of Artificial Intelligence Research **53**, 745–778 (2015). `https://doi.org/10.1613/jair.4726`
24. Meidani, K., Mirjalili, S., Farimani, A.B.: Online metaheuristic algorithm selection. Expert Systems with Applications p. 117058 (2022)
25. Mersmann, O., Bischl, B., Trautmann, H., Preuss, M., Weihs, C., Rudolph, G.: Exploratory Landscape Analysis. In: Proc. of Genetic and Evolutionary Computation Conference (GECCO'21). pp. 829–836. ACM (2011). `https://doi.org/10.1145/2001576.2001690`
26. Nobel, J., Wang, H., Bäck, T.: Explorative data analysis of time series based algorithm features of CMA-ES variants. pp. 510–518 (06 2021). `https://doi.org/10.1145/3449639.3459399`
27. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., et al.: Scikit-learn: Machine learning in Python. JMLR **12**, 2825–2830 (2011)
28. Rapin, J., Teytaud, O.: Nevergrad - A gradient-free optimization platform. `https://GitHub.com/FacebookResearch/Nevergrad` (2018)
29. van Rijn, S.: Modular CMA-ES framework from [30], v0.3.0. `https://github.com/sjvrijn/ModEA`. Available also as PyPi package at `https://pypi.org/project/ModEA/0.3.0/` (2018)
30. van Rijn, S., Wang, H., van Leeuwen, M., Bäck, T.: Evolving the structure of Evolution Strategies. In: Proc. of IEEE Symposium Series on Computational Intelligence (SSCI'16). pp. 1–8. IEEE (2016). `https://doi.org/10.1109/SSCI.2016.7850138`
31. Rinnooy Kan, A.H.G., Timmer, G.T.: Stochastic global optimization methods. part 1: Clustering methods. Math. Program. **39**(1), 27–56 (1987)
32. Schröder, D., Vermetten, D., Wang, H., Doerr, C., Bäck, T.: Chaining of numerical black-box algorithms: Warm-starting and switching points (2022). `https://doi.org/10.48550/arxiv.2204.06539`

33. Shanno, D.: Conditioning of quasi-newton methods for function minimization. In: Math. Comp. 24. pp. 647—-656 (1970)
34. Storn, R., Price, K.: Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. Journal of Global Optimization **11**(4), 341–359 (1997). `https://doi.org/10.1023/A:1008202821328`
35. Wang, H., Vermetten, D., Ye, F., Doerr, C., Bäck, T.: Iohanalyzer: Detailed performance analysis for iterative optimization heuristic. ACM Trans. Evol. Learn. Optim. (2022). `https://doi.org/10.1145/3510426`, `https://doi.org/10.1145/3510426`, to appear. IOHanalyzer is available at CRAN, on GitHub, and as web-based GUI, see `https://iohprofiler.github.io/IOHanalyzer/` for links
36. Xu, L., Hutter, F., Hoos, H.H., Leyton-Brown, K.: Evaluating component solver contributions to portfolio-based algorithm selectors. In: Proc. of Theory and Applications of Satisfiability Testing (SAT'12). Lecture Notes in Computer Science, vol. 7317, pp. 228–241. Springer (2012). `https://doi.org/10.1007/978-3-642-31612-8_18`, `https://doi.org/10.1007/978-3-642-31612-8_18`