



Universiteit
Leiden
The Netherlands

Comparing algorithm selection approaches on black-box optimization problems

Kostovska, A.; Jankovic, A.; Vermetten, D.L.; Dzeroski, S.; Eftimov, T.; Doerr, C.

Citation

Kostovska, A., Jankovic, A., Vermetten, D. L., Dzeroski, S., Eftimov, T., & Doerr, C. (2023). Comparing algorithm selection approaches on black-box optimization problems. *Gecco '23 Companion*, 495-498. doi:10.1145/3583133.3590697

Version: Publisher's Version

License: [Licensed under Article 25fa Copyright Act/Law \(Amendment Taverne\)](#)

Downloaded from: <https://hdl.handle.net/1887/3718555>

Note: To cite this publication please use the final published version (if applicable).

Comparing Algorithm Selection Approaches on Black-Box Optimization Problems

Ana Kostovska
Jožef Stefan Institute
Ljubljana, Slovenia

Anja Jankovic
Sorbonne Université, LIP6
Paris, France

Diederick Vermetten
Leiden University, LIACS
Leiden, The Netherlands

Sašo Džeroski
Jožef Stefan Institute
Ljubljana, Slovenia

Tome Eftimov
Jožef Stefan Institute
Ljubljana, Slovenia

Carola Doerr
Sorbonne Université, CNRS, LIP6
Paris, France

ABSTRACT

Performance complementarity of solvers available to tackle black-box optimization problems gives rise to the important task of algorithm selection (AS). Automated AS approaches can help replace tedious and labor-intensive manual selection, and have already shown promising performance in various optimization domains. Automated AS relies on machine learning (ML) techniques to recommend the best algorithm given the information about the problem instance. Unfortunately, there are no clear guidelines for choosing the most appropriate one from a variety of ML techniques. Tree-based models such as Random Forest or XGBoost have consistently demonstrated outstanding performance for automated AS. Transformers and other tabular deep learning models have also been increasingly applied in this context.

We investigate in this work the impact of the choice of the ML technique on AS performance. We compare four ML models on the task of predicting the best solver for the BBOB problems for 7 different runtime budgets in 2 dimensions. While our results confirm that a per-instance AS has indeed impressive potential, we also show that the particular choice of the ML technique is of much minor importance.

CCS CONCEPTS

• **Computing methodologies** → **Machine learning; Supervised learning**; • **Theory of computation** → **Design and analysis of algorithms**.

ACM Reference Format:

Ana Kostovska, Anja Jankovic, Diederick Vermetten, Sašo Džeroski, Tome Eftimov, and Carola Doerr. 2023. Comparing Algorithm Selection Approaches on Black-Box Optimization Problems. In *Genetic and Evolutionary Computation Conference Companion (GECCO '23 Companion)*, July 15–19, 2023, Lisbon, Portugal. ACM, New York, NY, USA, 4 pages. <https://doi.org/10.1145/3583133.3590697>

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).

GECCO '23 Companion, July 15–19, 2023, Lisbon, Portugal

© 2023 Copyright held by the owner/author(s).

ACM ISBN 979-8-4007-0120-7/23/07.

<https://doi.org/10.1145/3583133.3590697>

1 INTRODUCTION

Black-box problems are challenging to optimize as we do not have direct access to information about problem instances at hand, but instead learn about them by sampling and evaluating solution candidates. Black-box optimization algorithms use precisely this information in order to iteratively steer the search towards the optimum. Many algorithms have been developed for the purpose of solving black-box problems, and they exhibit different performances depending on their structural and behavioral properties. Choosing the best-suited algorithm for a certain problem instance is known as per-instance algorithm selection (PIAS), and it has led to impressive performance gains when algorithms available to choose from exhibit complementary performances [6]. Automated PIAS relies on ML techniques to predict which algorithm to recommend based on the characteristics of the problem instance at hand. There are many open challenges in adapting PIAS to black-box settings. One of them concerns the characterization of black-box problem instances without having any prior information about them. A classical technique to approximate problem instances by computing statistics from a set of solution candidates and their evaluations is known as *Exploratory Landscape Analysis* (ELA) [9]. So-obtained (*landscape*) *features* are then used to discriminate between instances.

Another prominent open question concerns the choice of the ML technique; it is not evident which approaches are best-suited for applications in the context of black-box optimization, and whether there is a big difference between the techniques at all. Given an ELA-based instance representation as input, *feature-based* ML approaches are commonly employed to learn the mapping between them and algorithm performances or rankings for each of those instances as output. Classical supervised approaches for such prediction tasks used in the literature are *regression* and *classification*, as well as a variant of the latter that is *pairwise classification*. Regression models learn to predict a real-valued metric (e.g., target precision or expected runtime), while classification consists of predicting a label of the category in which the input belongs (i.e., the name of the best algorithm). Pairwise classification allows for selecting the overall “winner” solver, following a tournament-like structure by converting a multi-class problem into a series of two-class problems. Among different models used for either regression or (pairwise-) classification tasks, classical *tree-based* models, such as Random Forests and XGBoost, have demonstrated tremendous success in algorithm performance prediction in the optimization domain. Recent advances in ML and deep learning (DL) have shown that certain neural network architectures, such as Transformers, could also be

successfully applied to tabular data prediction tasks [2, 3]. Notable examples include TabPFN [5], and FT-Transformer [2]. A large variety of available techniques yields a meta-optimization problem of choosing the best-performing AS approach or ML model for applications in black-box optimization.

Our contribution. In this paper, we assess the relevance of selecting the underlying ML technique for a given AS task on a single-objective black-box problem set with a portfolio of inherently different black-box solvers. Specifically, we compare the predictive power of three AS approaches (regression, classification, and pairwise classification), orthogonal to four ML models: two tree-based (Random Forest and XGBoost) in all three AS modalities, and two DL-based models (TabPFN for classification and FT-Transformer for regression) on the same landscape-aware AS task, for seven different solver runtime budgets and in two dimensionalities. We show that the choice of the ML technique has only a minor impact on the AS performance, as well as on the reliability and robustness of the results, as long as it is a method that demonstrates good performance on tabular data in general. At the same time, we confirm that PIAS truly is a core ingredient for boosting the performance of black-box solvers, irrelevant of the underlying ML technique; any algorithm selection approach outperforms any static choice of the solver across all settings. We suggest that research efforts should be instead directed towards (1) the identification of (and possibly involving the design of) suitable training sets and (2) the reduction of the overhead cost for feature extraction. The choice of the ML technique, in contrast, seems to be much less relevant for the design of efficient PIAS approaches in black-box optimization.

Availability of data and code. Following best practices towards replicability and reproducibility, full project data, code and detailed description of the experimental setup, as well as figures for all settings, are available at [8].

2 EXPERIMENTAL SETUP

2.1 Data Collection

Problem instance portfolio. For the problem instance portfolio, we make use of 24 single-objective, noiseless black-box optimization problems from the BBOB benchmark suite of the COCO environment [4], whose objectives are to be minimized. In our study, we consider the first 10 instances of the 24 BBOB functions, in dimensions 5 and 20. This results in two separate problem instance portfolios of size 240, one for each dimension.

Problem landscape data. For representing the problem landscape data, we make use of the so-called "cheap" ELA features implemented in the R package `flacco`. We have considered a total of 69 different ELA features, which have been calculated using the Latin hypercube sampling strategy with 1 000 sample size on a total of 100 independent repetitions. To represent each problem instance, we calculate the median value for each feature over the 100 repetitions.

Algorithm portfolio. For the algorithm portfolio, we use a diverse set of 11 algorithms. For details on the selected algorithms, we refer the reader to [8]. To collect performance data for our algorithm portfolio, we use the IOHexperimenter framework [1]. For each problem instance, we perform 50 independent repetitions, for a budget of 10 000 function evaluations for each run. The complementarity of algorithms is essential for reaching peak AS performance.

To this end, we have determined the proportion of problem instances where an algorithm is the optimal choice for each runtime, both for the 5D and 20D problem instance portfolios. Only the algorithms that exhibit superiority over 5% of the problem instances are included in the final algorithm portfolio. The final algorithm portfolios for different runtimes and problem dimensionalities are made available in [8].

2.2 Algorithm Performance Prediction

AS approaches. To predict the performance of algorithms, we consider three ML-based AS approaches: regression, classification, and pairwise classification. In all three cases, an ELA feature vector is used to describe each problem instance. In the case of regression, we aim to predict the target precision that each algorithm in the portfolio will achieve on the problem instances, for a fixed runtime and problem dimensionality. We cap the target precision at 10^{-8} and perform a $\log_{10}$ transformation of the target variable. For classification, to the ELA vector of each problem instance, we assign a label of the best performing algorithm from our portfolio. This translates to the task of multi-class classification, where the target variable is the algorithm name/label we aim to predict. Note that it can happen that multiple algorithms achieve the same target precision on a given problem instance (usually when they reach a target precision of 0, i.e., the optimum). In such cases, we randomly choose one of them as the best. Finally, in pairwise classification, we transform the multi-class classification problem into a set of binary classification problems, where we predict which algorithm performs better out of the two on a given problem instance. For an algorithm portfolio of size K , in the case of the classification, we have one multi-class classification problem with K classes. In the case of pairwise classification, however, $K(K - 1)/2$ binary classification problems need to be solved to cover all possible pairs of algorithms. We count the number of times an algorithm was predicted to be the best among all pairs, and form our final prediction by selecting the algorithm with the most "wins".

ML models. For each of the AS approaches, we train different ML models to assess whether the performance of the algorithm selector depends on the underlying model. We use Random Forest and XGBoost for all three AS approaches, as well as TabPFN for (pairwise-) classification and FT-Transformer models for regression.

Hyperparameter tuning and model evaluation. In order to evaluate the performance of our ML models, we employ a nested leave-one-group-out cross-validation (CV) technique, where groups are defined based on the ID of each benchmark problem instance. The setup consists of two stages of CV, with an outer loop splitting the data into training and testing sets and an inner loop determining the optimal hyperparameters for the model. We use the grid search strategy for hyperparameter tuning and determine the best hyperparameters based on the average performance on the holdout folds in the inner CV. The R^2 score and $F1$ score are utilized as performance metrics for regression and (pairwise-)classification, respectively. The hyperparameters selected for tuning and their corresponding search spaces are available at [8]. After identifying optimal hyperparameters, the final model is trained on all the training data and evaluated using the test set from the outer loop.

2.3 Per-instance Algorithm Selection

The AS quality is evaluated by comparing it to two standard baselines: (1) the *virtual best solver* (VBS), which represents the performance of a theoretical perfect algorithm selector that always chooses the true best algorithm for each problem instance, and (2) the *single best solver* (SBS), which is the algorithm with the best average performance among all algorithms in the portfolio. The “VBS-SBS gap” represents the difference between the VBS and SBS performances and gives an indication of the potential gains from PIAS. To evaluate the performance of PIAS, we compute the difference (or *loss*) between the target precision of the selected algorithm $F_{\mathcal{A}}$ and the target precision of the per-instance VBS $F_{\mathcal{A}^*}$. We do this after taking the logarithm of the achieved target precision: $\mathcal{L}(\mathcal{A}, \mathcal{A}^*) = \log_{10}(F_{\mathcal{A}}) - \log_{10}(F_{\mathcal{A}^*})$. We calculate the losses for each instance separately and investigate their distribution.

3 RESULTS

Following the experimental protocol described in Section 2, we first affirm that AS leads to impressive performance gains compared to any standalone solver. In order to assess the performance in a principled way, we test all chosen AS approaches/ML models on 10 instances of each of the 24 BBOB problems, for different algorithm runtime values and in dimensions 5 and 10. We compare the performance of the AS with that of different solvers in the portfolio, aggregated over all BBOB problems. We see that AS undoubtedly outperforms all solvers by a large margin, irrespective of the ML model or task. While we do not show the specific results of the AS comparison with individual solvers here due to space constraints, we make them available at [8].

3.1 Comparing Algorithm Selection Approaches

After establishing the relevance of AS itself, we compare the performance of different AS approaches (or underlying ML tasks), namely regression, classification, and pairwise classification. Contrasting the losses obtained by different AS approaches, and we do not observe any, drastic differences among them. However, regression-based AS seems to have a slightly worse performance than classification and pairwise classification. This holds true across all different ML models, all seven different runtimes, and in both dimensions [8].

To intuitively quantify the high performance (and thus reliability) of all considered approaches, we record the number of times that each solver from the portfolio is the VBS and the number of times each solver is selected by the AS based on regression, classification, and pairwise classification. We observe that all AS approaches very tightly follow the overall distribution of VBSs. There are no significant differences across different runtimes for any of the ML tasks. This is consistent for other ML models as well. The results and visualizations that support these conclusions are available at [8].

3.2 Comparing Machine Learning Models

We further investigate how different ML models compare for a fixed AS approach, across all runtimes and dimensions. To achieve this, we contrast the losses obtained by different ML models, as shown in Figure 1 for a runtime of 2500 in 5D. Similarly, as in Section 3.1, we conclude that the regression-based AS approach results in slightly larger losses compared to classification and pairwise classification,

but the difference is not substantial enough to draw definite conclusions. Among the regression models, the RF model outperforms XGBoost. This can be attributed to the ease of tuning RF compared to XGBoost. Implementing a different hyperparameter tuning strategy, such as Bayesian Optimization, would likely further improve the performance of XGBoost and bring it closer to that of RF.

The FT-Transformer regression model shows the largest loss compared to the other two regression-based models, which is in line with our expectations. As highlighted in [2], tree-based models outperform deep learning models, such as the FT-Transformer, on tabular problems. This may be partly due to the fact that DL models usually need more training data than tree-based models. In this study, with only 240 data examples, the FT-Transformer might not have sufficient training data to generalize effectively. Although the performance of the FT-Transformer could potentially be improved with a different tuning strategy, we still anticipate that the tree-based models will outperform the FT-Transformer in our AS task. On the other hand, when it comes to classification and pairwise classification, there are no significant differences between the ML models. This remains consistent across all runtimes and dimensions, as evidenced by the results available in [8].

Finally, we summarize the results of the AS performance across different ML models, AS approaches, runtimes, and dimensions in Figure 2. Here, instead of focusing on the loss as a quality indicator of the AS performance, we measure the percentage of problem instances where the VBS was selected as optimal by the AS. Our previous observations still hold true, i.e., regression-based AS performs worse than the other two approaches, especially in the case of FT-Transformer models (which is now clearly visible in Figure 2), while classification and pairwise classification exhibit comparable performance with a slight edge for classification. Notably, TabPFN performs similarly to tree-based methods, or even slightly better, which has also been observed in [5]. We point out that, for small runtimes (100 and 250) in 20D, exceptionally good performance of (almost) all techniques might be due to the small size of algorithm portfolios for respective runtimes. With less algorithms in the portfolio, the AS problem gets easier to tackle, which is evident in Figure 2 from the larger percentage of correctly selected VBS.

4 DISCUSSION AND CONCLUSIONS

We have confirmed in this work that PIAS bears considerable potential for numerical black-box optimization, and that ELA-based approaches are a sensible candidate to achieve impressive performance gains. We have also shown that the choice of the ML model (given that it had already been shown to perform well on tabular data in general) and the AS approach (regression, classification, pairwise classification) is of minor importance. However, this does not suffice to benefit from PIAS in practice, as real-world use cases remain very costly to both optimize and extract information from. We thus face two key challenges in applying these methods to the black-box optimization domain: (1) the cost for feature extraction and (2) warm-starting the chosen solvers.

Since our primary objective in this work has been the comparison of the different ML techniques, we have ignored challenge (1) by assuming that ELA features are available at no cost. This is of course not true in practice, where the solvers need to balance the trade-off

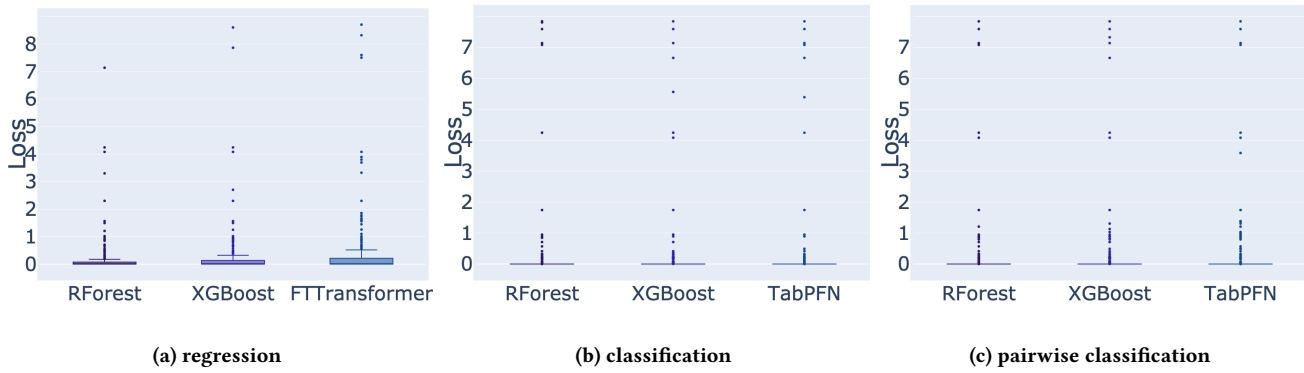


Figure 1: The loss of the AS across the ML model types for the three AS approaches (i.e., R = regression, C = classification, and PC = pairwise classification), aggregated over BBOB problems in 5D for runtime 2500.

	5D-100	5D-250	5D-500	5D-1000	5D-2500	5D-5000	20D-10000	20D-100	20D-250	20D-500	20D-1000	20D-2500	20D-5000	20D-10000
RForest-R	68.8	63.7	61.7	66.2	68.8	75.8	75.4	85	85	82.5	68.3	65.8	69.2	75
RForest-C	72.9	70.8	76.2	79.2	80.4	85	80	90	91.7	87.9	77.9	77.5	81.2	81.7
RForest-PC	73.8	64.6	69.2	75.4	80	78.8	80.4	90	85	88.3	75.8	75.4	76.7	79.2
XGBoost-R	62.9	60.4	49.2	60.4	62.9	69.2	75.4	79.6	83.8	61.3	50	57.1	59.6	69.6
XGBoost-C	70.8	71.2	76.2	81.7	81.7	80.8	79.6	91.7	91.2	87.1	78.3	76.7	80	81.2
XGBoost-PC	69.2	64.2	68.8	76.2	78.3	76.7	79.6	91.7	83.3	88.3	74.6	74.2	78.8	76.2
TabPFN-C	74.6	73.8	74.2	80.4	82.5	80.8	80.8	92.1	92.9	86.7	77.1	74.2	81.7	79.6
TabPFN-PC	72.5	64.2	70.8	75	75.4	83.3	80.8	92.1	83.3	86.2	74.6	72.5	76.7	78.3
FTTTransformer-R	48.8	36.7	42.1	53.3	58.8	59.6	57.5	71.2	69.2	53.3	46.2	38.8	44.6	53.8

Figure 2: A heatmap depicting the percentage of problem instances where the VBS is selected as optimal by the AS, for the various problem dimensions, runtimes, ML methods, and AS approaches.

between sampling for ELA computation and the budget that remains for the actual optimization phase. The first attempts aimed at integrating the ELA computation into the optimization process ([7] and references therein) encourage us to believe that ELA features can be valuable even when extracted from search trajectories rather than from a dedicated sampling phase that precedes the optimization, but also demonstrate much room for improvement. Regarding the warm-starting aspect, item (2) in our list above, we observe a surprising ignorance in the community, and this in works on per-instance algorithm selection, per-instance algorithm configuration, as well as in studies deliberately focusing on switching from one solver to another. We believe that neglecting the warm-starting aspect is wasteful. First proposals for warm-starting techniques have been made in [10] and [11]. The latter was considered in the *per-run* approach suggested in [7], but the small number of algorithms for which dedicated warm-starting techniques have been designed suggests that this can only be seen as the first starting point. We suspect both open questions (1) and (2) are of utmost

importance for the wider adoption of PIAS and per-instance algorithm configuration techniques in practice, both in academic and industrial use cases.

Acknowledgments: The authors acknowledge the support of the Slovenian Research Agency through program grant No. P2-0103 and P2-0098, project grants N2-0239 and J2-4460, a young researcher grant (PR-09773) to AK, and a bilateral project between Slovenia and France grant No. BI-FR/23-24-PROTEUS-001 (PR-12040), as well as the EC through grant No. 952215 (TAILOR). Our work is also supported by ANR-22-ERCS-0003-01 project VARIATION.

REFERENCES

- [1] Jacob de Nobel, Furong Ye, Diederick Vermetten, Hao Wang, Carola Doerr, and Thomas Bäck. 2021. IOHexperimenter: Benchmarking Platform for Iterative Optimization Heuristics. *CoRR* abs/2111.04077 (2021). arXiv:2111.04077 <https://arxiv.org/abs/2111.04077>
- [2] Yury Gorishniy, Ivan Rubachev, Valentin Khrukov, and Artem Babenko. 2021. Revisiting deep learning models for tabular data. In *Advances in Neural Information Processing Systems*, Vol. 34. 18932–18943. https://openreview.net/forum?id=i_Q1yrOegLY
- [3] Léo Grinsztajn, Edouard Oyallon, and Gaël Varoquaux. 2022. Why do tree-based models still outperform deep learning on typical tabular data?. In *NeurIPS 2022 Datasets and Benchmarks Track (Advances in Neural Information Processing)*. New Orleans, United States. <https://hal.science/hal-03723551>
- [4] Nikolaus Hansen, Anne Auger, Raymond Ros, Olaf Mersmann, Tea Tušar, and Dimo Brockhoff. 2020. COCO: A platform for comparing continuous optimizers in a black-box setting. *Optimization Methods and Software* 36 (2020), 114–144. <https://doi.org/10.1080/10556788.2020.1808977>
- [5] Noah Hollmann, Samuel Müller, Katharina Eggenberger, and Frank Hutter. 2022. TabPFN: A Transformer That Solves Small Tabular Classification Problems in a Second. <https://doi.org/10.48550/ARXIV.2207.01848>
- [6] Pascal Kerschke, Holger H. Hoos, Frank Neumann, and Heike Trautmann. 2019. Automated Algorithm Selection: Survey and Perspectives. *Evolutionary Computation* 27, 1 (2019), 3–45. https://doi.org/10.1162/evco_a_00242
- [7] Ana Kostovska, Anja Jankovic, Diederick Vermetten, Jacob de Nobel, Hao Wang, Tome Eftimov, and Carola Doerr. 2022. Per-run Algorithm Selection with Warm-starting using Trajectory-based Features. In *Parallel Problem Solving from Nature (PPSN) (LNCS, Vol. 13398)*. Springer, 46–60. https://doi.org/10.1007/978-3-031-14714-2_4 Free version available at <https://arxiv.org/abs/2204.09483>.
- [8] Ana Kostovska, Anja Jankovic, Diederick Vermetten, Sašo Džeroski, Tome Eftimov, and Carola Doerr. 2023. Comparing Algorithm Selection Approaches: data and code. <https://doi.org/10.5281/zenodo.7807853>
- [9] Olaf Mersmann, Bernd Bischl, Heike Trautmann, Mike Preuss, Claus Weihs, and Günter Rudolph. 2011. Exploratory landscape analysis. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO'11)*. ACM, 829–836. <https://doi.org/10.1145/2001576.2001690>
- [10] Hossein Mohammadi, Rodolphe Le Riche, and Eric Touboul. 2015. Making EGO and CMA-ES Complementary for Global Optimization. In *Proc. of Learning and Intelligent Optimization (LION) (LNCS, Vol. 8994)*. Springer, 287–292. https://doi.org/10.1007/978-3-319-19084-6_29
- [11] Dominik Schröder, Diederick Vermetten, Hao Wang, Carola Doerr, and Thomas Bäck. 2022. Chaining of Numerical Black-box Algorithms: Warm-Starting and Switching Points. *CoRR* abs/2204.06539 (2022). <https://doi.org/10.48550/arXiv.2204.06539>