An Efficient Contesting Procedure for AutoML Optimization

Duc Anh Nguyen, Anna Kononova, Stefan Menzel, Bernhard Sendhoff, Thomas Bäck

2022

Preprint:

This is an accepted article published in IEEE Access. The final authenticated version is available online at: https://doi.org/10.1109/ACCESS.2022.3192036

An Efficient Contesting Procedure for AutoML Optimization

DUC ANH NGUYEN[1], ANNA V. KONONOVA[1], STEFAN MENZEL[2], BERNHARD SENDHOFF[2], and THOMAS BÄCK[1]

[1]Leiden Institute of Advanced Computer Science (LIACS), Leiden University, The Netherlands
(e-mail: {d.a.nguyen,a.kononova,t.h.w.baeck}@liacs.leidenuniv.nl)
[2]Honda Research Institute Europe (HRI-EU), Offenbach/Main, Germany
(e-mail: {stefan.menzel,bernhard.sendhoff}@honda-ri.de)

Corresponding author: Duc Anh Nguyen (e-mail: d.a.nguyen@liacs.leidenuniv.nl).

The article has been accepted for publication in IEEE Access.

The work can also be accessed here: https://doi.org/10.1109/ACCESS.2022.3192036

IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier xx.xxxx/ACCESS.xxx.DOI

An Efficient Contesting Procedure for AutoML Optimization

DUC ANH NGUYEN[1], ANNA V. KONONOVA[1], STEFAN MENZEL[2], BERNHARD SENDHOFF[2], and THOMAS BÄCK[1]

[1]Leiden Institute of Advanced Computer Science (LIACS), Leiden University, The Netherlands
(e-mail: {d.a.nguyen,a.kononova,t.h.w.baeck}@liacs.leidenuniv.nl)
[2]Honda Research Institute Europe (HRI-EU), Offenbach/Main, Germany
(e-mail: {stefan.menzel,bernhard.sendhoff}@honda-ri.de)

Corresponding author: Duc Anh Nguyen (e-mail: d.a.nguyen@liacs.leidenuniv.nl).

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement number 766186 (ECOLE).

ABSTRACT Automated Machine Learning (AutoML) frameworks are designed to select the optimal combination of operators and hyperparameters. Classical AutoML-based Bayesian Optimization (BO) approaches often integrate all operator search spaces into a single search space. However, a disadvantage of this history-based strategy is that it can be less robust when initialized randomly than optimizing each operator algorithm combination independently. To overcome this issue, a novel contesting procedure algorithm, **D**ivide **A**nd **C**onquer **Opt**imization (DACOpt), is proposed to make AutoML more robust. DACOpt partitions the AutoML search space into a reasonable number of sub-spaces based on algorithm similarity and budget constraints. Furthermore, throughout the optimization process, DACOpt allocates resources to each sub-space to ensure that (1) all areas of the search space are covered and (2) more resources are assigned to the most promising sub-space. Two extensive sets of experiments on 117 benchmark datasets demonstrate that DACOpt achieves significantly better results in 36% of AutoML benchmark datasets: 5% when to compared to TPOT, 8% - to AutoSklearn, 15% - to H20 and 18% - to ATM.

INDEX TERMS Automated Machine learning (AutoML) Optimization, Machine Learning, Divide and Conquer, Bayesian Optimization, Hyperparameter Optimization, Classification

I. INTRODUCTION

Machine learning (ML) has achieved significant advances in a variety of real-world applications [1]–[3]. To achieve this goal, a practitioner needs to choose a well-performing sequence of algorithms, a.k.a, the ML pipeline, for the given problem and tune its hyperparameters to maximize the performance. The algorithms utilized in the ML pipeline are closely connected, with operators in each step having a direct impact on the next step(s). For example, the data pre-processing step aims to produce a new dataset, i.e., balanced, reduced-dimensions, etc., which may affect the subsequent operator's performance, such as e.g. the learning model. As a result, deciding which ML pipeline to use is a difficult task that requires the experience of machine learning professionals. In order to efficiently solve ML problems without (or with minimal) human effort, Automated Machine Learning (AutoML) is a promising paradigm for automatically generating the optimal machine learning pipeline, i.e., the best sequence of algorithms with their hyperparameters optimally adjusted. Technically, a typical AutoML framework consists of three fundamental components [4]: a search space, an optimizer, and an objective function. The search space describes the feasible search domain. The optimizer is used to discover the best combination of algorithms over operators and their optimized hyperparameter that maximise the objective function's performance. Finally, the objective function is a child program that evaluates the settings of the ML pipeline, resulting in a real-valued performance measure, e.g., accuracy, precision, recall rate. In other words, AutoML aims at automatically identifying an efficient ML pipeline setting from the feasible search space.

In recent years, the AutoML community has increasingly applied the Bayesian Optimization (BO) [5]–[7] approach due to its high efficiency [8]–[13]. Therefore, in this study, we targeted improving BO-based approaches for solving the AutoML optimization problem. The AutoML optimization

(AO) problem is typically considered as a single optimization problem in the BO-based method by merging the optimization space for all algorithms of all operators – this approach is typically refer to as *integrated approach* [14]. The Combined Algorithm Selection and Hyperparameter optimization (CASH) approach [8] is a commonly used technique, where the AO problem is treated as a Hyperparameter Optimization (HPO) problem. However, HPO was initially developed to optimize hyperparameters of a single algorithm, where the considered search space is typically smaller, lower-dimensional, and less (even non)-structured than the AutoML search space. Hence, the HPO-based approach is not ideal for handling the AO problem. In order to alleviate the above limitation, we formulate the AO problem as a ML pipeline optimization problem, which is proposed by [13]. This can be seen as a generalization of the CASH approach, where the parameter classes for operator's algorithms, and hyperparameters in an algorithm were clearly identified.

An alternative to the *integrated approach*, [15] proposes the so-called CASH-oriented Multi-Armed Bandits (MAB) approach to solve the model selection and hyperparameter optimization problem for the classification problem (i.e., selecting a classification algorithm and tuning the hyperparameter, simultaneously), by applying HPO on each classifier separately. However, this is might not possible to apply to AutoML scenarios since the number of combinations of algorithms over operators can be up to thousands. Fortunately, [13], [16]–[18] has pointed out that operator algorithms can potentially be grouped and that different groups of algorithms perform better on different kinds of problems, e.g., a group of linear classification algorithms performs best on linear classification tasks.

This study therefore intend to further improve BO performance for the AO problems by applying the Divide and Conquer (DAC) strategy: the AutoML search space is divided into multiple sub-spaces based on their similarity[1], and each sub-space is solved by a separate BO process (candidate) independently. The budget is then allocated to each candidate depending on their performance using a novel competing mechanism. As a result, the most promising candidates have a larger tuning budget than the least prospective candidates. Therefore, the worst candidates will be 'fired' as soon as ample evidence against them has been gathered, saving computation time and resources for future assessments in those search areas.

Notably, since our approach handles the BO[2] processes independently, allows multiple optimization processes executed simultaneously without affecting the performance. In other words, our technique will achieve the same numerical results in both parallel and sequential settings, with the exception of different execution times.

Our contributions: We summarize our main contributions are the following:

- We propose a novel contesting procedure, namely DACOpt, to solve the AutoML optimization problem efficiently, which is complementary to the existing BO approaches.
- DACOpt efficiently allocates resources to each sub-space to ensure that (1) all areas of the search space are covered and (2) more resources are assigned to the most promising sub-space. Additionally, we provide a theoretical guarantee that our approach fixes the existing gap between serial and parallel BO execution.
- Two independent empirical studies on a range of AutoML optimization problems with 2 and 6 operators on the total of 117 benchmark datasets demonstrate the superiority of the proposed approaches.

Reproducibility and Open Science: The implementation of the proposed methods is published in a git-repository[3] and PyPi-repository[4]. The experiment scripts for the reproducibility of the reported results are provided in a git-repository[5].

The remainder of this paper is organized as follows. Section II presents the relevant background knowledge on AutoML optimization problem, Bayesian optimization, Divide and Conquer techniques and early-stop strategies. Our contributions are highlighted in Section III, whereas Section IV lays out the experimental setup. Experimental results are discussed in Section V. Finally, the paper is summed up and further work is outlined in Section VI.

II. BACKGROUND

In this section, we first review the formalization of AutoML optimization problem (Section II-A), the Bayesian optimization approaches (Section II-B), the relevant techniques to the proposed contesting procedure (Section II-C), and early-stop strategies (Section II-D).

A. AUTOML OPTIMIZATION PROBLEM

Given a ML problem with a dataset $\mathcal{D}$, let $\mathcal{M} = (\mathcal{O}_1, \ldots, \mathcal{O}_z)$ denote the sequence of operators. Each $\mathcal{O}_i$ has a set of available ML algorithms: $\mathcal{O}_{i\in\{1,\ldots,z-1\}} = \{\varnothing, \mathcal{A}_i^1, \ldots, \mathcal{A}_i^{n_i}\}$ for all operators except the last and $\mathcal{O}_z = \{\mathcal{A}_z^1, \ldots, \mathcal{A}_z^{n_z}\}$ for the last operator which defines the learning algorithm. Further, let $\Lambda = \{\{\Lambda_1^1, \ldots, \Lambda_1^{n_1}\}, \ldots, \{\Lambda_z^1, \ldots \Lambda_z^{n_z}\}\}$ be a set of hyperparameter spaces of all algorithms for all operators. The AutoML optimization problem is then to find the best ML pipeline setting $p_{(\mathcal{A}_{1,\lambda},\ldots,\mathcal{A}_{z,\lambda})^*}$ that maximizes the chosen accuracy metric f (e.g., the accuracy, precision or recall rate):

$$p_{(\mathcal{A}_{1,\lambda},\ldots,\mathcal{A}_{z,\lambda})^*} = \arg\max_{l,\lambda} f(p_{(\mathcal{A}_{1,\lambda},\ldots,\mathcal{A}_{z,\lambda})^l}, \mathcal{D}) \quad (1)$$

where $(\mathcal{A}_1, \ldots, \mathcal{A}_z)^l \in \bigtimes_{i=1}^{z} \mathcal{O}_i$ are all possible choices of algorithms for all pipeline operators, $\lambda = \{(\lambda_1, \ldots, \lambda_z) | \lambda_1 \in \Lambda_1^{j_1}, \ldots, \lambda_z \in \Lambda_z^{j_z}\}$ are algorithms' hyperparameters and $f\left(p_{(\mathcal{A}_{1,\lambda},\ldots,\mathcal{A}_{z,\lambda})^l}, \mathcal{D}\right)$ is performance of the sequence operators and their corresponding hyperparameter choices when

[1] see grouping approach proposed in [13]

[2] BO (a.k.a., Sequential model-based optimization) was originally intended as a sequential approach [5], [19]

[3] The library is published at https://github.com/ECOLE-ITN/DACOpt.

[4] https://pypi.org/project/DACOpt

[5] https://github.com/ECOLE-ITN/DACOpt/tree/main/Experiments

trained on training set $\mathcal{D}_t$ and evaluated on validation set $\mathcal{D}_v$, where $\mathcal{D} = \{\mathcal{D}_t, \mathcal{D}_v\}$.

B. BAYESIAN OPTIMIZATION

The Bayesian optimization approach is widely used in many AutoML studies, for example: [8], [9], [11]. In a nutshell, BO uses a surrogate model $\mathcal{P}(f|\mathcal{H})$ to capture the settings that have been examined thus far, $\mathcal{H} = \{(p_i, \Delta_i)_{i=1}^t\}$, where p denotes the pipeline setting under evaluation and Δ denotes the corresponding measured performance. $\mathcal{P}$ is then used to predict the performance of a numerous set of unseen pipeline configurations and choose the subsequent configuration which maximizes the specified acquisition function.

Expected improvement (EI) [20] is a common acquisition function. It balances the trade-off between exploration and exploitation via the expectation of the improvement function over the best found value $\Delta^*_{(t)}$ at time step t as $I(p) = \max\{0, \hat{f}(p) - \Delta^*_{(t-1)}\}$, where $\Delta^*_{(t)} = \max(\Delta^0, \ldots, \Delta^{t-1})$ and $\hat{f}(p)$ denotes the predicted performance of the setting p via surrogate model $\mathcal{P}$. The EI is then defined as:

$$\mathbb{E}[I(p)] = \int_0^\infty I(p)\, d\mathcal{P} \tag{2}$$

Hence, the next setting is selected by maximizing the EI:

$$p_{new} = \arg\max_{p \in \mathcal{M}} \mathbb{E}[I(p)] \tag{3}$$

where $\mathcal{M}$ denotes the AutoML search space (see Section II-A).

Past works [8], [9], [12], [19] have noted that the tree-based surrogate models, e.g., Tree-structured Parzen Estimator (TPE) [6] and Random Forest [5] have been successfully used for the class of AO problems. Thus, in this study, we also adopt TPE.

TPE uses a kernel density estimator [21] to model the likelihood. Based on a predefined threshold γ, the set of evaluated settings $\mathcal{H}$ is split into two sets: γ[6] of well-performing settings $l(p)$ and the remaining settings $g(p)$ labelled as badly-performing. In this way, maximizing the ratio $\frac{l(p)}{g(p)}$ is equivalent to maximizing the EI in Equation 2, as shown theoretically in [6]. In other words, the next setting p_{new} is chosen by maximizing the above-mentioned ratio.

C. CONTESTING PROCEDURE FOR AUTOML OPTIMIZATION

AutoML aims at solving automatically and efficiently an arbitrary ML problem. However, the No Free Lunch (NFL) theorem [22] prescribes that there is no universal optimal algorithm for all problems. As a result, in order to maximize performance on an arbitrary problem, AutoML must incorporate as many algorithm choices as possible. Additionally, every algorithm has a set of hyperparameters. Consequently, AutoML optimization typically is a high-dimensional mixed-variables (continuous, discrete, nominal) optimization problem. In order to handle such a challenge by a BO approach, three facts are considered: (1) BO performs better for low-dimensional problems [23], (2) AO problems have low effective dimensionality [24], [25], (3) the complexity of AO problem not only comes from its dimensionality, but also the number of possible combinations of algorithms within the ML pipeline [13].

Divide and Conquer (DAC) [26] is a well-known strategy for handling large problems via decomposing the target problem into c small-scale and low-dimensional sub-problems. Consider an AO problem $p^* = \arg\max_{p \in \mathcal{M}} f(p)$, for applying DAC, the approach has to first decompose the AutoML search space into c sub-spaces, and then solve each sub-space by an optimizer. Assuming that we can split the AutoML search space $\mathcal{M}$ into c small-spaces $\{\mathcal{M}_1, \ldots, \mathcal{M}_c\}$, the DAC approach can be formulated as:

$$p^* = (\arg\max_{p \in \mathcal{M}_1} f(p), \ldots, \arg\max_{p \in \mathcal{M}_c} f(p)) = (p_1^*, \ldots, p_c^*) \tag{4}$$

where p_i^* is the global optimum of sub-space $\mathcal{M}_i$ and p^* is the global optimum of the original search space $\mathcal{M}$.

The existing DAC studies typically treat elements of the input search space as the same level and decomposed by complementing [27]. That is, variables corresponding to sub-space $\mathcal{M}_i$ can change freely while the remaining $|\mathcal{M} - \mathcal{M}_i|$ dimensions are set to some fixed values. However, such approaches cannot be used for the AutoML search space where dimensions are hierarchical and, thus, dependent. Therefore, there are two challenges for adapting DAC to solve AutoML problems: (1) how to divide the AutoML space $\mathcal{M}$ onto a set of c sub-spaces efficiently; (2) how to optimize resources during the 'conquer' phase since some sub-spaces' performance might be significantly worse than others. To answer the above questions, we propose (1) a splitting approach based on the combination of groups of operator algorithms [13], (2) adopting efficient early-stop strategies that are based on the theoretical guarantees (see our discussion in Section II-D) for optimizing resources for the 'conquer' phase.

Furthermore, since the number of algorithms (and therefore, the number of sets of their parameters) is smaller in the DAC-formulation of the AutoML problem in Equation 4 compared to the original formulation in Equation 1, the proposed contesting procedure also concurrs the assumption [24], [25] that AO problem has low effective dimensionality.

D. EARLY-STOP STRATEGIES

In order to prevent over-fitting, the k-fold cross-validation is usually added to Equation 1. For readability, let p denote $p_{(\mathcal{A}_{1,\lambda},\ldots,\mathcal{A}_{z,\lambda})}$. The robust AutoML problem is then formulated as:

$$p^* = \arg\max_{p \in \mathcal{M}} \frac{1}{k} \sum_{j=1}^{k} f\left(p, \mathcal{D}_t^j, \mathcal{D}_v^j\right) \tag{5}$$

where $f\left(p, \mathcal{D}_t^j, \mathcal{D}_v^j\right)$ is performance of the pipeline setting p when trained and evaluated on the j^{th} cross-validation data

[6] $\gamma = 25\%$, can be changed by user

fold $\mathcal{D}_t^j$ and $\mathcal{D}_v^j$, correspondingly. As a consequence of using cross-validation, every function evaluation becomes k times more expensive. Early stop strategy, e.g., [19], [28]–[32] allows limiting this issue, since it avoids wasting time and resources on evaluating worse settings over all k folds.

The important concept is to stop investigating a setting as soon as sufficient information indicates that it is ineffective. A setting will only be examined on a few folds in this manner; an iterative elimination function will analyze its performance on the evaluated folds in order to compare it to other evaluated settings and determine how many folds should be utilized for the considered setting.

The elimination function in racing procedure approaches [29], [30] is based on a statistical test procedure, i.e., Friedman test [33], meanwhile bandit-based approaches [31], [32] compares the setting performance directly to the best-known setting. In a number of case studies, both strategies performed well [14], [30], [32], where the task of proposing new settings was commonly handled by a Random search [25]. Unfortunately, the inconsistencies in how settings are assessed may provide additional noise for BO, making it less reliable in suggesting subsequent settings. This means that such approaches should not be used directly and should be adopted only at the level of search sub-spaces, via the termination of unpromising sub-spaces (see Section III-A).

III. PROPOSED APPROACH

We now discuss our proposed contesting procedure for AutoML optimization problems based on the **D**ivide **A**nd **C**onquer strategy, which we call DACOpt.

A. ALGORITHM DESCRIPTION

We reformulate the AutoML optimization problem in Equations 1, 4 and 5 into the following:

$$p^* = \arg\max_{p \in \mathcal{M}} (p_1^*, \ldots, p_c^*) \tag{6}$$

$$\text{s.t.} \quad p_i^* = \arg\max_{p_i \in \mathcal{M}_i} \frac{1}{k} \sum_{j=1}^{k} f\left(p_i, \mathcal{D}_t^j, \mathcal{D}_v^j\right) \tag{7}$$

where $\mathcal{M}_i = (\mathcal{O}_1^{(i)}, \ldots, \mathcal{O}_z^{(i)})$ denotes the i^{th} sub-space $\forall i \in \{1, \ldots, c\}$, $\mathcal{O}_{l \in \{1,\ldots,z\}}^{(i)} = \{\mathcal{A}_l^1, \ldots, \mathcal{A}_l^{n_l}\}$ denotes a set of algorithms of the l^{th} operator $\forall |\mathcal{O}_l^{(i)}| \leq |\mathcal{O}_l|$, and a set of the corresponding hyperparameters of $\mathcal{O}_l^{(i)}$: $\Lambda_{l \in \{1,\ldots,z\}}^{(i)} = \{\Lambda_l^1, \ldots, \Lambda_l^{n_l}\}$ and $f\left(p_i, \mathcal{D}_t^{(j)}, \mathcal{D}_v^{(j)}\right)$ denotes performance of the setting, similar to Equation 5.

The overall structure of the contesting procedure proposed here is summarized in Figure 1. The process begins with a *Splitter* function to be applied on the input AutoML search space $\mathcal{M}$ to produce c possible sub-spaces. Here, we extends the work of [13] with with improvements (a detailed discussion on this function is given in Appendix A). Then, c BO processes are initialized (in the following discussion, the BO processes shall be called candidates). The whole contest

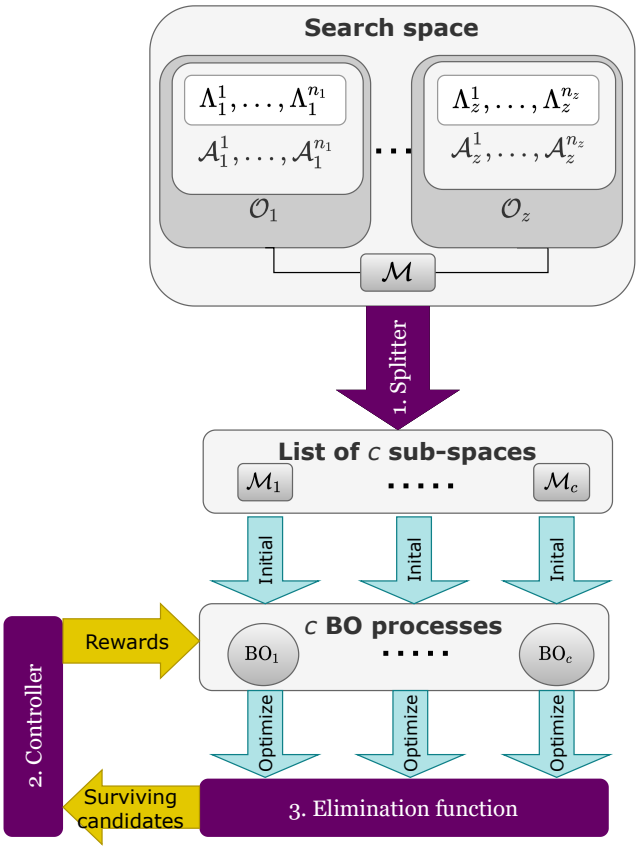

FIGURE 1: The workflow of the contesting procedure

is controlled by the *Controller* function, which allocates budgets to each candidate per contest round based on the feedback from the *Elimination function* that decides which candidates will survive into the next round based on their performances so far. As mentioned in Section II-D, we adopt two possible settings for the early-stop functionality. Therefore, two versions of DACOpt are provided, which differ mainly w.r.t. the elimination criteria:

1) Elimination criteria based on the highest performances

As discussed in Section II-B, the acquisition function maximizes the *best-found value* $\Delta_{(t)}^*$ up to time step t. Due to the fact that the goal of AutoML optimization is to find the setting that achieves the highest performance on the target ML problem, we consider the highest performance as a suitable comparison criteria. Furthermore, the way of computing the budget, step size, and the number of rounds follow the approach in [31], [32] with minor adjustments: input parameters of our procedure include the maximum number of sub-spaces to be split c, total optimizing budget B, and the ratio of candidates discarded in each round η[7]. The number of rounds in the contest is then calculated as: $R_{max} = \lceil log_\eta(c) \rceil$. Each round has the same budget $B_r = \frac{B}{R_{max}}$. That is, each of m surviving candidates at the round can have a budget of $b = \lfloor \frac{B_r}{m} \rfloor$. At the end of the round, the *Elimination* function

[7] $\eta = 3$, can be changed by user

keeps $\lceil \frac{m}{\eta} \rceil$ candidates for the following round. Therefore, the surviving candidate has η times the budget from the previous round.

Our approach, elaborated in Algorithm 1. We require the maximum sub-spaces to be split K and the ratio of candidates discarded in each round, η, as input parameters. This approach consists of the following steps:

- Initialize: Split the original search space into c ($c \leq K$) sub-spaces (line 3). Next, initialize c corresponding Bayesian optimization candidates (lines 5-7). Next, the number of contest rounds is calculated, $R_{max} = \lceil log_{\eta}(c) \rceil$ (line 8-9).
- Parameter for each round: Based on the number of surviving candidates from the previous round, the number of candidates c_r for the current round r is computed in line 18. The elimination function discards candidates labelled as performing badly and returns a set of c_r good candidates (line 20). Here, we simply select the top c_r candidates based on their best-found values. A reasonable budget for each candidate is computed based on the remaining budget, remaining rounds, and the number of surviving candidates (lines 23-25). All the above steps (lines 18-25) are repeated every round, except the first round. The *first round*, all candidates survive and are given a budget of $b = b_{init}$ (line 12-16).
- Finally, using value b obtained in the previous step, all surviving candidates continue their optimize processes (line 27-33).

2) Elimination criteria based on a statistical procedure

As mentioned in Section II-D, our second option adopts the approach of racing procedures to determine well and badly performing candidates. This approach also requires a maximum number of sub-spaces c and a level of significance α. Since the effectiveness of BO is mostly seen in the later phases of optimization when it learns to produce better settings, we only count the best-found value of the initial sampling step to be considered for further statistical tests. Unlike the first elimination criteria method, this method does not compute the number of rounds or budget for each round since it completely depends on the statistical results; instead we use a step size λ[8] to limit budget per round. At the end of the round, a Friedman test [33] is performed to verify whether there is significant difference between the pair of candidates. If it is the case, a Holm post-hoc test [34][9] is applied to compare the highest-ranked candidate to others. Any candidate that fails the test is removed from the list of surviving candidates. This loop is repeated until the best candidate is found.

This approach also requires a maximum number of sub-spaces K and a level of significance λ. This process, summarized in Algorithm 2, consists of the following steps:

- Phase 1- Initialize: Using the same split function as Algorithm 1, to produce k ($k \leq K$) sub-spaces (line 1).

[8] $\lambda = 1$, can be changed by user

[9] Following the recommendations by [35], [36]

Algorithm 1 DACopt based on the highest performance

1: **Input:** $\mathcal{M}$: Search space, K: number of sub-spaces to be splitted, f : objective function, B: maximal number of evaluations, b_{init}: number of evaluations for initial step in each of k BO processes, η: ratio controls the proportion of candidates discarded in each round
2: **Output:** p^*: the best pipeline setting, Δ^*: the best value
3: $\{\mathcal{M}_1, \ldots, \mathcal{M}_c\}, c \leftarrow$ SPLITTER$(\mathcal{M}, K)$
▷ divide the input search space into c sub-spaces, $c \leq K$
4: $b_0 = 0$
▷ initialize the corresponding BO processes with 0 budget.
▷ **BEGINNING OF INITIAL PHASE**
5: **for** $\mathcal{M}_i \in \{\mathcal{M}_1, \ldots \mathcal{M}_c\}$ **do**
6: $\text{BO}_i, \mathcal{H}_i \leftarrow$ BAYESIAN OPTIMIZER $(\mathcal{M}_i, f, b_0)$
7: **end for**
▷ **BEGINNING OF CONTESTING PHASE**
8: $R_{max} \leftarrow \lceil log_{\eta}(c) \rceil$ ▷R_{max}: number of rounds
9: $r = 0$ ▷ r: round number
10: **repeat**
11: **if** $r = 0$ **then**
12: $c_r \leftarrow c$
13: $(\text{BO}_1, \ldots, \text{BO}_{c_r}) \leftarrow (\text{BO}_1, \ldots \text{BO}_c)$
14: ▷Note: at the first round $c_r = c$, but the order of candidates are shuffled.
15: $b \leftarrow b_{\text{init}}$
16: ▷all candidates have an equal budget b_{init} for the first round
17: **else**
18: $c_r \leftarrow \lceil \frac{c_{\text{previous}}}{\eta} \rceil$
19: ▷minimum number candidates in the current round
20: $(\text{BO}_1, \ldots, \text{BO}_{c_r}) \leftarrow \text{ELI.}\big((\text{BO}, \mathcal{H})_{i \in \{1,\ldots,c\}}, c_r\big)$
21: $\triangleright \mathcal{H} = \{(p_n, \Delta_n)_{n=1}^{\text{evaluated}}\}$
22: ▷Select goodness candidates for the current round, ordered by performance/rank
23: $B_r \leftarrow \lfloor \frac{B}{R_{max} - r} \rfloor$
24: ▷B_r:total budget for the current round
25: $b \leftarrow \lfloor \frac{B_r}{c_r} \rfloor$ ▷ budget per candidate
26: **end if**
27: **for** $\text{BO}_i \in \{\text{BO}_1, \ldots, \text{BO}_{c_r}\}$ **do**
28: BO_i .ADDBUDGET(b)
▷ add budget b to the selected candidate
29: $\text{BO}_i, \mathcal{H}_i \leftarrow \text{BO}_i$ .OPTIMIZE$()$
30: ▷Continues BO_i process until the added budget b run out.
31: $B \leftarrow B - b$
32: ▷Update the remaining budgets
33: **end for**
34: $c_{\text{previous}} \leftarrow c_r$
35: $r \leftarrow r + 1$
36: **until** $r <= R_{max}$
37: **Return** $p^*, \Delta^* = \arg\max_{p,\Delta} \{\mathcal{H}\}_{i \in \{1,\ldots,c\}}$

All candidates are initialized with the minimum required budget b_{init} (line 3).

- The main operates in the contesting phase: maintain a set of surviving candidates [10]. A statistical test will be performed at each round to determine if there is any pair of candidates are significantly different (line 17). If the null hypothesis is false, a post-hoc test is being applied on each pair of candidates (line 18-19). Any candidate that fails the test is removed from the survived candidates (line 20-21). Next, a budget λ is added to each candidate in the survived set. This procedure is repeated until the total budget runs out.

Lastly, both options naturally support parallel implementation. We require the number of maximum available threads $\tau, (\tau \geq 1)$, as an extra input parameter for the parallel mode. The parallel mode will be discontinued when the best subspace is found. In both algorithms, parallel mode is applied to execute the BO processes.

B. FIXING THE GAP BETWEEN SERIAL AND PARALLEL BO

Bayesian optimization, called otherwise the Sequential model-based optimization (SMBO), is naturally sequential. However, most modern optimizer-based BO approaches include a parallelized version in addition to the original BO method. AutoML-based BO is typically parallelized by either assessing in parallel (1) cross-validation folds or (2) multiple settings, e.g., [37]–[39]. While the first approach focuses on parallelizing evaluations inside the objective function, it does not affect BO, but it is efficient when k is less than the available resources. The second approach might lead to inefficient solutions proposed by BO, in terms of the number of function evaluations. Since the objective function is expensive, we have to choose a configuration that might perform best. In the following, we discuss how parallelized BO can lead to poorer results compared to serial approaches.

Let us consider a noiseless function $f : \mathcal{M} \subset \mathbb{R}^d \rightarrow \mathbb{R}$ and it's real-valued surrogate model $\hat{f} = \{\mathcal{P}(p_i, \Delta_i)_{i=1}^t\}$ for time step t. At a new step $t+1$, a sampling approach (randomly) generates a set of solutions $\{\hat{p}_1, \ldots, \hat{p}_n\}$. Those later will be estimated by the surrogate model $\hat{f}$ and used to propose **one** setting $p_{t+1} \in \{\hat{p}_1, \ldots, \hat{p}_n\}$ by maximizing the acquisition function in Eq. 3. The set of m next settings from the time step t of the sequence approach is $\{p_{t+1}^{\prime\prime} = \arg\max_{p\in\mathcal{M}} \mathbb{E}[I(p_t)], \ldots, p_{t+m}^{\prime\prime} = \arg\max_{p\in\mathcal{M}} \mathbb{E}[I(p_{t+m-1})]\}$. In contrast, the parallel approach proposes a set of solutions $\{p_{t+1}^{||_1}, \ldots, p_{t+1}^{||_m}\} \in \arg\max_{p\in\mathcal{M}} \mathbb{E}[I(p_t)]$. Let $\overline{p} = |f(p) - \hat{f}(p)|$ denote the difference between the performance of the setting p on the true objective function f and it's surrogate $\hat{f}$. Clearly, the quality of BO in suggesting new solution(s) is highly dependent on $\hat{f}$ and the statistical property of $\hat{f}$ (i.e., uncertainty) at time t, which significantly increases as more historical data is collected. Thus, $\sum_{j=1}^m \overline{p_{t+j}^{\prime\prime}} \geq \sum_{j=1}^m \overline{p_{t+1}^{||_j}}$. Hence, the quality of m

[10]Notes for the contesting phase: Since the effectiveness of BO mainly after the initial sampling step when it learns to produce better settings. Thus, we only count the best-found value of the initial sampling step to be considered further statistical tests. We only perform the statistical test when the sample size exceeds 2.

Algorithm 2 DACOpt based on the statistical test

1: **Input:** $\mathcal{M}$: Search space, K: number of sub-search spaces, f : objective function, B: maximal number of evaluations, b_{init}: minimum evaluations per sub-search space, $\lambda = 1$: step size, $\alpha = 0.05$: level of significance
2: **Output:** p^*: the best pipeline setting, Δ^*: the best value
3: $\{\mathcal{M}_1, \ldots, \mathcal{M}_c\}, c \leftarrow$ SPLITTER$(\mathcal{M}, K)$
▷ divide the input search space into c sub-spaces, $c \leq K$
▷ **BEGINNING OF INITIAL PHASE**
4: **for** $\mathcal{M}_i \in \{\mathcal{M}_1, \ldots \mathcal{M}_c\}$ **do**
5: BO$_i, \mathcal{H}_i \leftarrow$ BAYESIAN OPTIMIZER $(\mathcal{M}_i, f, b_{\text{init}})$
6: **end for**
▷ **BEGINNING OF CONTESTING PHASE**
7: $c_r \leftarrow c$ ▷ c_r number of surviving candidates
8: **repeat**
9: ▷Performs a chosen statistical test with α to detect there is at least one pair of candidates that are significantly different
10: **if** $c_r < 3$ **then**
11: STAC $\leftarrow$ WILCOXONTEST()
12: ▷ Initial WILCOXONTEST if $c_r < 3$, this will be used in line 17
13: **else**
14: STAC $\leftarrow$ FRIEDMANTEST()
15: ▷Initial FRIEDMANTEST if $c_r \geq 3$, this will be used in line 17
16: **end if**
17: **if** $(\neg$STAC$(\{\mathcal{H}_i\}_{i=1}^{survive}, \alpha))$ & $c_r > 1$ **then**
18: $i^* = \arg\max$ RANKING$(\{\mathcal{H}_i\}_{i=1}^{survive})$
19: ▷i^*: the highest ranked among the surviving candidates based on a ranking test
20: ▷Performs a HOLM post-hoc test to detect candidates significantly worse than i^*
21: $c_r \leftarrow$ number of surviving candidates
22: **else if** $c_r = 1$ **then**
23: $\lambda \leftarrow B$
24: ▷Last round when only 1 candidate, the remaining budgets is assigned
25: **end if**
26: **for** BO$_i \in \{$BO$_i\}_{i=1}^{survive}$ **do**
27: BO$_i$ .ADDBUDGET(λ)
▷ add budget λ to the selected candidate
28: BO$_i, \mathcal{H}_i \leftarrow$ BO$_i$ .OPTIMIZE()
▷ Continues BO$_i$ process.
29: $B \leftarrow B - \lambda$
▷ Update the remaining budgets
30: **end for**
31: **until** $B \geq 0$
32: **Return** $p^*, \Delta^* = \arg\max_{p,\Delta} \{\mathcal{H}\}_{i\in\{1,\ldots,c\}}$

additional time steps in the sequential method may be more robust than those in the parallel technique. Thus, there is a discrepancy between the current serial and parallel BOs.

For the reasons above, we use sequential BO to solve each search sub-space. Fortunately, BO processes in our proposed procedure are independent (see Figure 1). Therefore, we introduce a partly-parallel approach instead of fully parallel. Instead of proposing a set of future solutions from a single search area like the fully parallel technique does, in order to ensure the best performance of BO at every iteration, DACOpt proposes a set of next setting solutions as sequential approach from multiple independent search areas: $p^{\mathcal{M}_i}_{t+1} = \arg\max_{p \in \mathcal{M}_i} \mathbb{E}[I(p_t)], \forall i \in \{1, \ldots, c\}$. Thus, $p^{\mathcal{M}_i}_{t+1}$ in either serial or parallel situations are exactly the same. For parallel computing, a parallel pool of m available workers will be repeated $\lceil \frac{c}{m} \rceil$ times to finish c processes. The last iteration of that parallel pool is partly parallel if $(c \mod m) > 0$ and fully parallel otherwise. As a result, our approach holds the same effectiveness in both cases.

The key benefits of our DACOpt approach are as follows:

- Based on the performance of the related BO process, the budget adaptively redistributes to the search area. As a result, the budget is distributed effectively.
- As a partly-parallel BO variant, the proposed approach has parallel efficiency without harming BO performance.
- BO performance and robustness can be increased since each BO process optimizes a relatively small low-dimensional search space independently.

IV. EXPERIMENTAL SETUP

In order to evaluate the robustness and general applicability of our proposed approach, we compare it to other state-of-the-art AutoML optimization approaches, reproducing the experimental setup with the total of 117 benchmark datasets on two scenarios with 2 operators (Section IV-A) and 6 operators (Section IV-B) to be optimized. In both scenarios, we compare the performance of BO-based variants with the TPE surrogate model (BO4ML and Hyperopt [6], [40], [41]) with the two proposed contesting procedures (see Table 1) against those without such procedure. Parameters setting for both experiments are summarized in Table 2.

TABLE 1: Proposed DACOPT approaches compared in this study

Name	Contesting procedure		BO variant	
	highest	statistical	BO4ML	Hyperopt
DAC-HB	✓		✓	
DAC-HH	✓			✓
DAC-SB		✓	✓	
DAC-SH		✓		✓

A. FIRST EXPERIMENTAL SETUP

The first experiment is based on a search space of resampling (21 techniques) and classification (5 algorithms) operators, for class-imbalanced problems on 44 binary datasets (see Figure 3 for the description of these datasets). The overall structure of this experimental setup is summarized in Figure 2. The process starts with data preprocessing, i.e., Label encoding, Scaling, on the input dataset. The 5-fold cross-validation is applied at this step to overcome the over-fitting problem. The outcome is then fed into the second phase optimizing the search space of 2 operators: resampling and classification operators.

TABLE 2: Parameter settings

	$\mathbf{1}^{st}$ **experiment**	$\mathbf{2}^{nd}$ **experiment**
DACOpt parameters		
- Number of candidates (K)	10	10
- Init. sample size per candidate (B_{init})	5	5
- Total budgets (B_{max})	500 (func. eval.)	1 (hour)
- BO variants used		
Hyperopt	✓	✓
BO4ML	✓	✓
Experimental parameters		
-Search space	2 operators	6 operators
Relevant papers:		
[12]	✓	–
[9]	–	✓
- Number of datasets	44	73
- Performance metric	Geometric mean (GM)	Accuracy rate (Acc)
- Train/test split	No	train: 70% test: 30%
- k-folds cross validation (for optimization)	5	4
- Final results	Average over k-folds	Accuracy rate of the unseen test set

1) Search space

This experiment used a search space of 2 operators with the total of 64 hyperparameters (see also [12]):

- The first operator is the resampling operator, aiming to resamples the imbalanced input dataset to have a balanced dataset. The resampling operator includes 21 resampling approaches; they fall into 4 major groups, such as No resampling, Over-resampling (7 algorithms), Under-resampling (11 algorithms), Combine-resampling (2 algorithms). The resampling algorithms are implemented in the Python package **imbalanced-learn**[11].
- The final operator is the classification operator, with 5 classification algorithms (i.e., Support Vector Machines (SVM), Random Forest (RF), K-Nearest Neighbors (KNN), Decision Trees (DTC), and Logistic Regression (LR)). The classification algorithms are implemented in the Python package **Scikit-learn**[12].

[11] https://github.com/scikit-learn-contrib/imbalanced-learn
[12] https://scikit-learn.org/

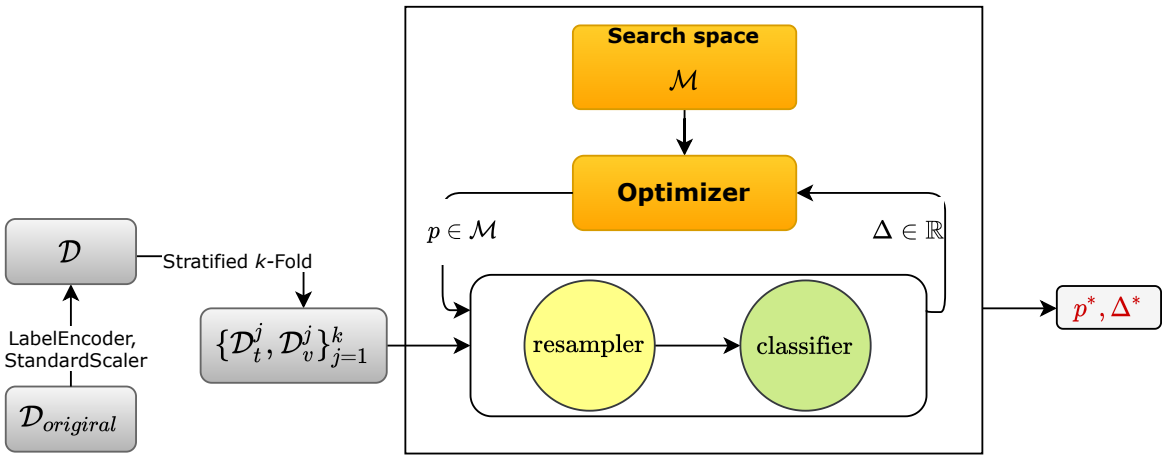

FIGURE 2: Flowchart of the experimental setup.

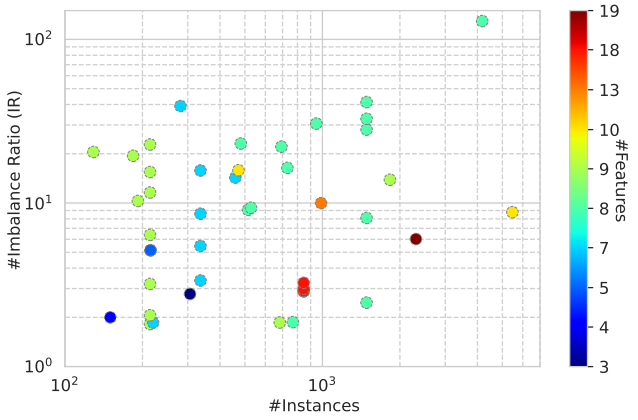

FIGURE 3: Overview of the characteristics of 44 imbalanced benchmark datasets. The scatter plot shows the #Imbalance Ratio (IR) and the number of instances (#Instances) on a logarithmic scale. The color indicates the number of features (#Features). Figure best viewed in color.

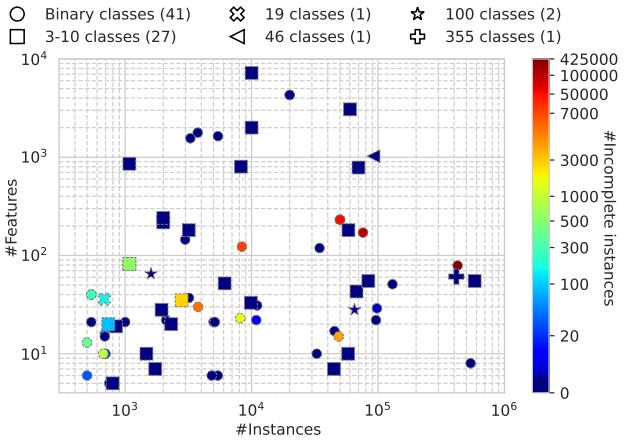

FIGURE 4: Overview of the characteristics of 73 AutoML benchmark datasets. The scatter plot shows the number of features (#Features) and the number of instances (#Instances) on a logarithmic scale. The symbols indicate the number of classes and the color indicates the number of samples that contain missing value (#Incomplete instances).

2) Parameter setting:

We used a budget of 500 function evaluations, with the original experimental setup, data pre-processed and source code provided by [13]. Number of candidates K set to 10, the initial sample size for each candidates is 5. The 5-fold cross-validation approach is used, and the averaged geometric mean values over 10 repetitions are reported.

B. SECOND EXPERIMENTAL SETUP

The second experiment is based on Auto-Sklearn [9] search space with up to 6 operators for classification problems on 73 AutoML benchmark datasets (described in Figure 4). In this experiment we compare our 4 proposed approaches to the 6 well-known AutoML frameworks, i.e., Auto-sklearn (BO and Random Search) [9], [42], TPOT [43], ATM [44],H20 [45], and Hyperopt-sklearn (HP-sklearn) [11]. All experiments ran over 10 independent repetitions with a wall-time limit of 1 hour per run on 8 CPU-cores. In this experiment, we used a search space generated by Auto-sklearn [9]. We experiment with our approaches as the same experimental setup with 73 AutoML benchmark datasets reported in [46]. The overall structure of our AutoML framework is summarized in Figure 5:

1) The process begins by downloading the corresponding dataset from OpenML [47], [48] of the OpenML #Task ID (input by user).
2) The necessary metadata is extracted from the input dataset to generate a suitable search space χ by the Auto-sklearn search space generator. Note that this search space generator is based on two aspects: the machine learning problem, i.e., binary classification, multi-class classification, multi-label classification, regression, multi-output regression, and the data representation, i.e., either dense or sparse representation. In practice,

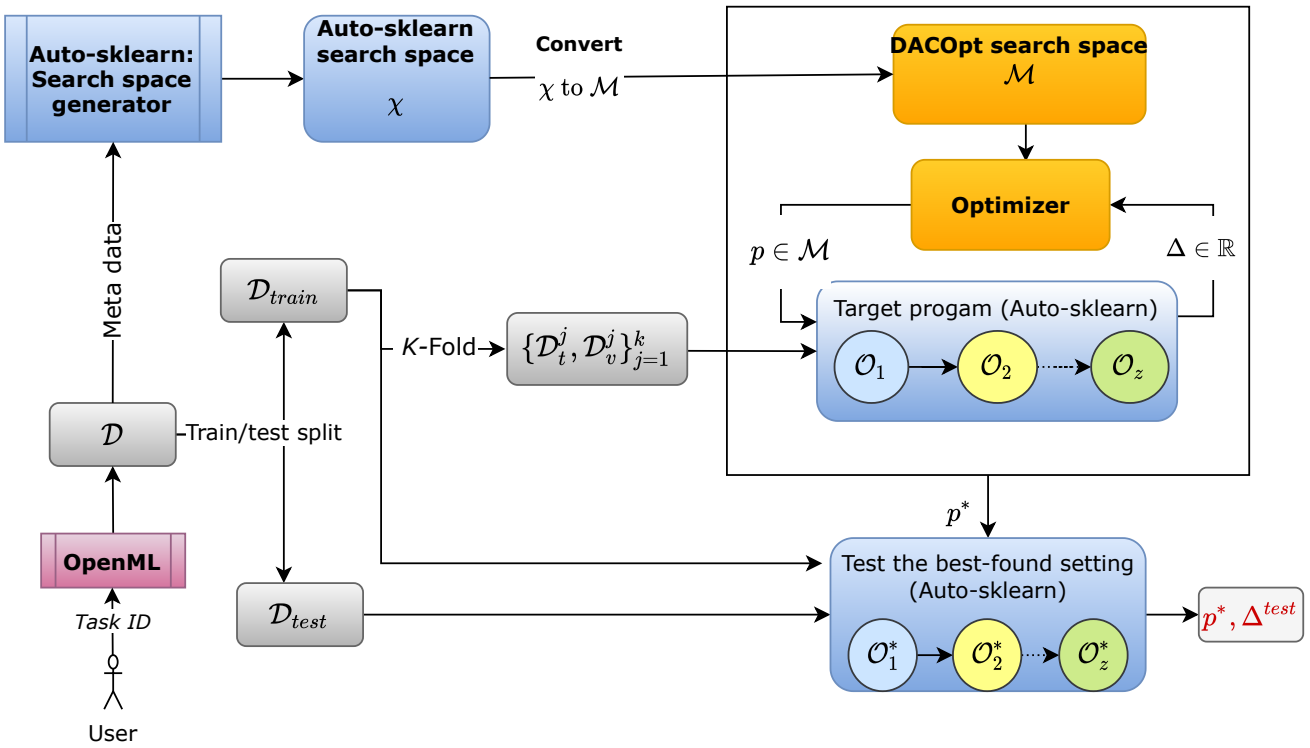

FIGURE 5: Flowchart of the second experimental setup.

the generated search space for a single ML problem is large and commonly having up to 153 hyperparameters and 6 operators, i.e., categorical encoder, numerical transformer, imputation transformer, re-scaling, feature pre-processor, and learning operator.

3) The search space χ is converted to our search space $\mathcal{M}$, which allows setting a hierarchical tree of similarity of algorithms[13]. In the meantime, the input dataset is reprocessed and split into two independent sets $\mathcal{D}_{train}$ and $\mathcal{D}_{test}$, with the original data prepossess and train/test split techniques used in [46], i.e., 30% for testing and the remaining for training. Next, the 4-fold cross-validation is applied on D_{train} to avoid the over-fitting problem. The later optimization phase takes those k-folds and search space $\mathcal{M}$. For a fair comparison, the optimization time only counts after this downloading step.
4) We optimizes the search space $\mathcal{M}$ until the wall-time reaches 1 hour and return the best-found pipeline setting p^*, consisting of a sequence of operators and their optimized hyperparameter settings.
5) Once the optimization process is done, the best-found pipeline setting p^* is used to initialize the corresponding machine learning model. It then learns $\mathcal{D}_{train}$ and predicts $\mathcal{D}_{test}$. Lastly, the test accuracy is calculated.

[13]Here we based on the hierarchy used in [49], [50] and discussed in [16]

1) Parameter setting

For a fair comparison, we use computational resources similar to [46] and [13]. For clarification, all experiments are conducted using our available computation clusters, namely *The Distributed ASCI Supercomputer 5* (DAS5) [18], each computation node (32 cores) parallelly runs 4 experiments, i.e., fixing 8 cores for one experiment. All experiments repeated 10 times with different random seeds, limited by a soft-limit of 1 hour [14] and a hard-limit of 1.25 hours [15]. The performance of a single configuration is limited to 10 minutes with 4-folds cross-validation on the training data, i.e., the evaluation of a fold is allowed to take up to 150 seconds. The evaluation of a configuration is aborted and returns a zero if any folds got an error, e.g., infeasible configuration, timeout. Finally, the average accuracy rates on test data over 10 repetitions are reported.

C. REPRODUCIBILITY:

The reproducible scripts for both experimental results are provided in a git-repository https://github.com/ECOLE-ITN/DACOpt/tree/main/Experiments.

Lastly, both experiments do not use parallel computing to ensure a fair comparison; however, we provide theoretical

[14]Soft-limit: the timeout' parameter set to optimizer.

[15]Hard-limit: The optimization process will be manually aborted after 1.25 hours for any unexpected technical reasons. In this way, the configuration which achieved the highest performance is known as the best configuration of the run.

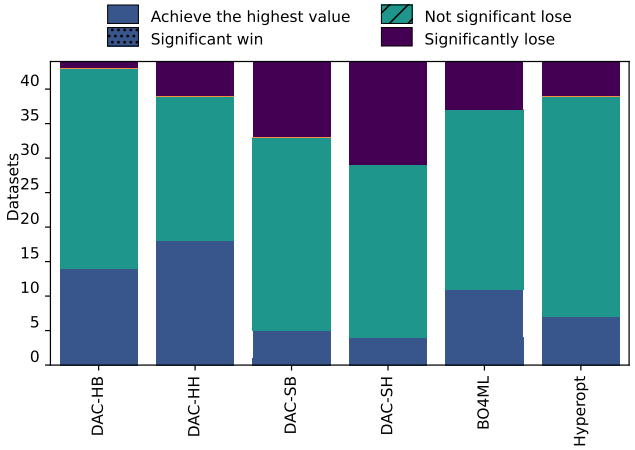

FIGURE 6: Overview of the results over 10 repetitions for the 44 binary imbalanced benchmark datasets. Figure best viewed in color.

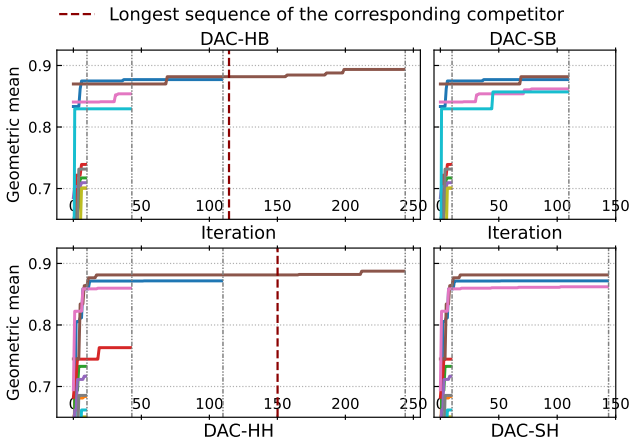

FIGURE 7: Illustration of the contesting process on dataset *abalone9-18*. This figure shows the optimization convergence plots of DAC-HB (top-left), DAC-SB (top-right), DAC-HH (bottom-left) and DAC-SH (bottom-right) approaches.

guarantees that our partially parallel execution is quicker than serial execution (see Section III-B).

V. RESULTS AND DISCUSSION

In this section, we report and discuss the results obtained from the two experimental setups introduced above. Generally speaking, we target three goals: (1) to compare the performance of our two contesting procedures in terms of *number of function evaluation* and *wall-time limit*; (2) to compare the performance of BO with and without the proposed contesting procedures; (3) to compare those against the current state-of-the-art AutoML frameworks.

For each tested case, the method that achieved the highest performance is counted as winning, provided that its performance is significantly better than all other methods, according to the Wilcoxon signed-rank test [51] with level $\alpha = 0.05$. The method that performs significantly worse than the best is counted as a loss. If there is no significant performance difference between two methods, they are considered equal. The method is counted as a well-performing if either achieved the best performance or not significantly worse than the best found method on the corresponding case.

A. FIRST EXPERIMENT RESULTS

The results of the first experiment are summarized in Figure 6 to illustrate the perfomance of our four tested approaches, i.e., DAC-HB, DAC-SB, DAC-HH, DAC-SH compared to BO4ML and Hyperopt. This figure is based on the average geometric mean over a 5-fold cross-validation over 44 imbalanced binary benchmark datasets. We make the following observations:

- Comparing two methods that use the highest performance as the elimination criteria (highest value-based contest), DAC-HH achieved the highest performance more times than DAC-HB (18 vs. 14). However, DAC-HB significantly won on more tested cases than DAC-HH. Additionally, DAC-HB loses on fewer cases than DAC-HB (1 vs. 5).
- Compared to the contesting procedures that used statistical tests as the elimination criteria (statistical-based contest), two methods, i.e., DAC-SH and DAC-SB, achieved a similar performance.
- Overall, DAC-HB performs well in most of the tested cases. More precisely, over 44 tested dataset, DAC-HB loses only on dataset *pima*, where DAC-HH is the winner.

Lastly, another point worth mentioning is that, we expected the statistical-based approaches, i.e., DAC-SB and DAC-SH, to perform better than the highest-based approaches, i.e., DAC-HB and DAC-HH. However, the experimental results are contrary to our assumption. To investigate their optimizing behavior, we plot a single run of those approaches on the dataset *abalone9-18* in Figure 7. Two plots on the left show the convergence behavior of the highest-based and statistical-based contests on the right. All approaches use a total budget of 500 function evaluations, and the search space is split into 10 sub-spaces. The dashed-grey vertical line indicates a contest round cutoff point, i.e., the end of the round where the elimination function is called. The extra dashed-red vertical line on the two left plots shows the most extended sequence of the corresponding underlying optimizer. We can observe that the examinated search space is relatively small and not significantly different. The statistical-based approach maintains more candidates throughout the contest than the highest-based approach. Consequently, the best candidate was found late with less budget than the best candidate in the competitor approach.

B. SECOND EXPERIMENTAL RESULTS

The results of the second experiment are summarized in Figure 8, based on a Wilcoxon signed-rank test with $\alpha = 0.05$. This figure is based on the accuracy of the test dataset over 10

FIGURE 8: Overview of the results over 10 repetitions for the 73 AutoML benchmark datasets. Figure best viewed in color.

repetitions to show the performance differences between two BO variants-based TPE surrogate models, namely BO4ML and Hyperopt, to compare both with and without the proposed contesting procedure, as well as with two elimination criteria, namely the highest value and a statistical procedure (see Table 1). Additionally, we also compared against the current state-of-the-art AutoML frameworks, i.e.,Auto-sklearn and Random search (Random), HPsklearn, TPOT, ATM, H20, based on the results obtained by [13], [46]. The details of our experimental results are provided in Appendix C-B.

- Firstly, comparing the three approaches that used Hyperopt as the underlying optimizer, i.e., DAC-HH, DAC-SH, and Hyperopt, we can observe that both proposed contesting procedures won on more tested cases than Hyperopt. More precisely, DAC-HH, DAC-SH, Hyperopt significantly outperform others in $8, 3, 2$ cases, respectively. However, in those tested cases of Hyperopt, it is never significantly better than both DAC-SH and DAC-HH; DAC-SH and DAC-HH are not significantly different. In contrast, DAC-HH and DAC-SH significantly outperform Hyperopt in 5 and 1 cases, correspondingly. Therefore, we can conclude that (1) both contesting procedures significantly improved the performance of BO, (2) DAC-HH won Hyperopt on more cases compared to those on DAC-SH.
- Secondly, we analyse the results of three approaches that use BO4ML as the underlying optimizer, i.e., DAC-HB, DAC-SB, and BO4ML. We observed that: (1) all three approaches performed well on $73\%, 67\%$, and 55% tested cases, respectively; (2) DAC-HB achieved the highest performances on the most of tested cases, followed by BO4ML and DAC-SB. In 11 cases where BO4ML significantly outperformed others, it was not significantly better than any of competitors in this comparison. DAC-SB was significantly better than BO4ML on 1 tested case, i.e., task 146821, but it never won DAC-HB. In comparison, DAC-HB outperformed DAC-SH and BO4ML on 3 and 7 cases, correspondingly.
- Comparing the results of 8 approaches using the search space of Auto-sklearn, i.e., our four approaches, BO4ML, Hyperopt, Auto-sklearn, and Random search. Firstly, all BO-based approaches perform better than Random search over all tested cases. Random search achieves the highest result in 1 case (#ID: 24), where all competitors perform equally (no win). Secondly, it can be seen that DAC-HB won in most tested cases, followed by BO4ML, DAC-HH, Auto-Sklearn, DAC-SH, DAC-SH, Hyperopt, and Random search, respectively.
- Additionally, when comparing all contesting variants together, it can be seen that DAC-HB won on more tested cases than others. The contesting procedure based on the highest performance, i.e., DAC-HH, DAC-HB, won on more cases than those based on statistical procedure, i.e., DAC-SH, DAC-SB. This finding could be explained by the fact that executing a statistical method adds to the overall computational cost of the procedure. As a result, the contesting technique that used statistical procedures examined fewer configurations in the same amount of time as the others.
- Finally, the proposed contesting procedures performed well on up to 73% and at least 53% over all tested cases, when to compared to Random Search - 8%, Hyperopt - 11%, AutoSklearn - 21%, TPOT - 27%, ATM - 30% and H20 - 37%.

VI. CONCLUSIONS AND FUTURE WORK

In this paper, we proposed a novel contesting procedure for the AutoML optimization problem, namely DACOpt, which is complementary to the existing BO approaches. DACOpt partitions the AutoML search space into multiple relatively small sub-spaces based on algorithm similarity and budget constraints. Next, BO approaches are employed to optimize those sub-spaces independently. The budget is then adaptively distributed to the search area based on the performance of the corresponding BO processes. The proposed contesting procedure has two different variants of elimination criteria – based on the highest performance and a statistical procedure. Additionally, we presented a partly-parallel approach to use BO to address the AutoML optimization problems with provably theoretical guarantees. Moreover, two extensive experiments on the total of 117 benchmark datasets demonstrated the superiority of our novel contesting procedures over the current state-of-the-art AutoML optimization approaches. In the future, we intend to incorporate meta-learning approaches to identify search areas that may perform well early.

.

ACKNOWLEDGMENT

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement number 766186 (ECOLE).

REFERENCES

[1] I. J. Goodfellow, Y. Bengio, and A. Courville, Deep Learning. MIT Press, 2016. http://www.deeplearningbook.org.

[2] C. Wang, T. Bäck, H. H. Hoos, M. Baratchi, S. Limmer, and M. Olhofer, "Automated machine learning for short-term electric load forecasting," in IEEE Symposium Series on Computational Intelligence (IEEE SSCI), pp. 314–321, 2019.

[3] M. Kefalas, M. Baratchi, A. Apostolidis, D. van den Herik, and T. Bäck, "Automated machine learning for remaining useful life estimation of aircraft engines," in International Conference on Prognostics and Health Management, pp. 1–9, 2021.

[4] D. Peng, X. Dong, E. Real, M. Tan, Y. Lu, G. Bender, H. Liu, A. Kraft, C. Liang, and Q. V. Le, "PyGlove: Symbolic programming for automated machine learning," in Advances in Neural Information Processing Systems (NIPS), vol. 33, pp. 96–108, 2020.

[5] F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Sequential model-based optimization for general algorithm configuration," in International Learning and Intelligent Optimization Conference, pp. 507–523, 2011.

[6] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for hyper-parameter optimization," in Advances in Neural Information Processing Systems (NIPS), vol. 24, pp. 2546–2554, 2011.

[7] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. de Freitas, "Taking the human out of the loop: A review of bayesian optimization," Proceedings of the IEEE, vol. 104, no. 1, pp. 148–175, 2016.

[8] C. Thornton, F. Hutter, H. Hoos, and K. Leyton-Brown, "Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms," in Proceedings of KDD-2013, p. 847–855, Association for Computing Machinery, 08 2012.

[9] M. Feurer, A. Klein, K. Eggensperger, J. T. Springenberg, M. Blum, and F. Hutter, "Efficient and robust automated machine learning," in Advances in Neural Information Processing Systems (NIPS), 2015.

[10] L. Kotthoff, C. Thornton, H. H. Hoos, F. Hutter, and K. Leyton-Brown, "Auto-weka 2.0: Automatic model selection and hyperparameter optimization in weka," Journal of Machine Learning Research, vol. 18, no. 25, pp. 1–5, 2017.

[11] B. Komer, J. Bergstra, and C. Eliasmith, "Hyperopt-sklearn: automatic hyperparameter configuration for scikit-learn," in Proceedings of the Python in Science Conferences (SciPy), 2014.

[12] D. A. Nguyen, J. Kong, H. Wang, S. Menzel, B. Sendhoff, A. V. Kononova, and T. Bäck, "Improved automated cash optimization with tree parzen estimators for class imbalance problems," in IEEE International Conference on Data Science and Advanced Analytics, pp. 1–9, 2021.

[13] D. A. Nguyen, A. V. Kononova, S. Menzel, B. Sendhoff, and T. Bäck, "Efficient AutoML via combinational sampling," in IEEE Symposium Series on Computational Intelligence (IEEE SSCI), 2021.

[14] D. Vermetten, H. Wang, C. Doerr, and T. Bäck, "Integrated vs. sequential approaches for selecting and tuning CMA-ES variants," in The Genetic and Evolutionary Computation Conference, 2020.

[15] J. Li, Y.and Jiang, J. Gao, Y. Shao, C. Zhang, and B. Cui, "Efficient automatic CASH via rising bandits," in Association for the Advancement of Artificial Intelligence, pp. 4763–4771, AAAI Press, 2020.

[16] M. Fernández-Delgado, E. Cernadas, S. Barro, and D. Amorim, "Do we need hundreds of classifiers to solve real world classification problems?," Journal of Machine Learning Research, vol. 15, p. 3133–3181, Jan. 2014.

[17] J. Kong, W. Kowalczyk, D. A. Nguyen, S. Menzel, and T. Bäck, "Hyper-parameter optimisation for improving classification under class imbalance," in IEEE Symposium Series on Computational Intelligence (IEEE SSCI), pp. 3072–3078, 2019.

[18] H. Bal, D. Epema, C. de Laat, R. van Nieuwpoort, J. Romein, F. Seinstra, C. Snoek, and H. Wijshoff, "A medium-scale distributed system for computer science research: Infrastructure for the long term," Computer, vol. 49, pp. 54–63, may 2016.

[19] S. Falkner, A. Klein, and F. Hutter, "BOHB: Robust and efficient hyper-parameter optimization at scale," in International Conference on Machine Learning, pp. 1437–1445, 2018.

[20] J. Močkus, "On bayesian methods for seeking the extremum," in Optimization Techniques IFIP Technical Conference Novosibirsk, pp. 400–404, Springer Berlin Heidelberg, 1975.

[21] E. Parzen, "On estimation of a probability density function and mode," The Annals of Mathematical Statistics, vol. 33, no. 3, pp. 1065 – 1076, 1962.

[22] D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization," IEEE Transactions on Evolutionary Computation, 1997.

[23] K. Eggensperger, M. Feurer, F. Hutter, J. Bergstra, J. Snoek, H. H. Hoos, and K. Leyton-brown, "Towards an empirical foundation for assessing bayesian optimization of hyperparameters," in NIPS Workshop on Bayesian Optimization in Theory and Practice, 2013.

[24] Z. Wang, M. Zoghi, F. Hutter, D. Matheson, and N. De Freitas, "Bayesian optimization in high dimensions via random embeddings," in International Joint Conference on Artificial Intelligence, 2013.

[25] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," Journal of Machine Learning Research, vol. 13, no. 10, pp. 281–305, 2012.

[26] P. Yang, K. Tang, and X. Yao, "Turning high-dimensional optimization into computationally expensive optimization," IEEE Transactions on Evolutionary Computation, vol. 22, no. 1, pp. 143–156, 2018.

[27] P. Yang, K. Tang, and X. Yao, "A parallel divide-and-conquer-based evolutionary algorithm for large-scale optimization," IEEE Access, vol. 7, pp. 163105–163118, 2019.

[28] O. Maron and A. W. Moore, "Hoeffding Races: Accelerating model selection search for classification and function approximation," in Advances in Neural Information Processing Systems (NIPS), 1993.

[29] O. Maron and A. W. Moore, "The racing algorithm: Model selection for lazy learners," Artificial Intelligence Review, vol. 11, p. 193–225, 1997.

[30] M. López-Ibáñez, J. Dubois-Lacoste, T. Stützle, and M. Birattari, "The irace package: Iterated racing for automatic algorithm configuration," Operations Research Perspectives, vol. 3, pp. 43–58, 2016.

[31] K. Jamieson and A. Talwalkar, "Non-stochastic best arm identification and hyperparameter optimization," in The International Conference on Artificial Intelligence and Statistics (AISTATS), pp. 240–248, 2016.

[32] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar, "Hyperband: A novel bandit-based approach to hyperparameter optimization," Journal of Machine Learning Research, 2017.

[33] M. Friedman, "The use of ranks to avoid the assumption of normality implicit in the analysis of variance," Journal of the American Statistical Association, pp. 675–701, 1937.

[34] S. Holm, "A simple sequentially rejective multiple test procedure," Scandinavian Journal of Statistics, vol. 6, no. 2, pp. 65–70, 1979.

[35] H. Ismail Fawaz, G. Forestier, J. Weber, L. Idoumghar, and P. Muller, "Deep learning for time series classification: a review," Data Mining and Knowledge Discovery, vol. 33, pp. 917–963, 2019.

[36] I. Rodríguez-Fdez, A. Canosa, M. Mucientes, and A. Bugarín, "STAC: a web platform for the comparison of algorithms using statistical tests," in IEEE International Conference on Fuzzy Systems (FUZZ-IEEE), pp. 1–8, 2015.

[37] F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Parallel algorithm configuration," in International Learning and Intelligent Optimization Conference, pp. 55–70, 2012.

[38] B. van Stein, H. Wang, and T. Bäck, "Automatic configuration of deep neural networks with parallel efficient global optimization," in International Joint Conference on Neural Networks, pp. 1–7, 2019.

[39] D. Ginsbourger, R. Le Riche, and L. Carraro, "Kriging is well-suited to parallelize optimization," in Computational Intelligence in Expensive Optimization Problems, Springer series in Evolutionary Learning and Optimization, pp. 131–162, springer, 2010.

[40] J. Bergstra, B. Komer, C. Eliasmith, D. Yamins, and D. Cox, "Hyperopt: a Python library for model selection and hyperparameter optimization," Computational Science & Discovery, vol. 8, no. 1, p. 014008, 2015.

[41] J. Bergstra, D. Yamins, and D. D. Cox, "Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures," in International Conference on Machine Learning, vol. 28, pp. 115–123, 2013.

[42] M. Feurer, K. Eggensperger, S. Falkner, M. L., and F. Hutter, "Auto-sklearn 2.0: The next generation," CoRR, vol. abs/2007.04074, 2020.

[43] R. S. Olson and J. H. Moore, TPOT: A Tree-Based Pipeline Optimization Tool for Automating Machine Learning, pp. 151–160. Cham: Springer International Publishing, 2019.

[44] T. Swearingen, W. Drevo, B. Cyphers, A. Cuesta-Infante, A. Ross, and K. Veeramachaneni, "ATM: A distributed, collaborative, scalable system for automated machine learning," in IEEE International Conference on Big Data (Big Data), pp. 151–162, 2017.

[45] E. LeDell and S. Poirier, "H2O AutoML: Scalable automatic machine learning," ICML Workshop on Automated Machine Learning, 2020.

[46] M. Zöller and M. Huber, "Benchmark and survey of automated machine learning frameworks," Journal of Artificial Intelligence Research, vol. 70, pp. 409–472, 2021.

[47] J. Vanschoren, J. N. van Rijn, B. Bischl, and L. Torgo, "OpenML: Networked science in machine learning," SIGKDD Explorations, vol. 15, no. 2, p. 49–60, 2013.
[48] M. Feurer, J. van Rijn, A. Kadra, P. Gijsbers, N. Mallik, S. Ravi, A. Müller, J. Vanschoren, and F. Hutter, "OpenML-Python: an extensible python api for OpenML," Journal of Machine Learning Research, vol. 22, 2021.
[49] G. Lemaître, F. Nogueira, and C. K. Aridas, "Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning," Journal of Machine Learning Research, vol. 18, no. 17, pp. 1–5, 2017.
[50] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," Journal of Machine Learning Research, vol. 12, pp. 2825–2830, 2011.
[51] F. Wilcoxon, "Individual comparisons by ranking methods," Biometrics Bulletin, vol. 1, no. 6, pp. 80–83, 1945.
[52] J. Alcalá-Fdez, L. Sánchez, S. Garcia, M. J. del Jesus, S. Ventura, J. M. Garrell, J. Otero, C. Romero, J. Bacardit, V. M. Rivas, et al., "KEEL: a software tool to assess evolutionary algorithms for data mining problems," Soft Computing, vol. 13, no. 3, pp. 307–318, 2009.

DUC ANH NGUYEN (M'21) is currently a PhD Student at the Leiden Institute of Advanced Computer Science (LIACS) - Leiden University, The Netherlands, where he is working as an Early-Stage Researcher of the European research project "ECOLE"– which is an Innovative Training Network funded by the EU Horizon 2020– in collaboration with Honda Research Institute Europe (HRI)-Germany, NEC Labs Europe (NEC) – Germany and University of Birmingham – UK. His research interests include Automated Machine Learning (AutoML) and optimization algorithms.

ANNA V. KONONOVA (M'04) is currently an Assistant Professor at the Leiden Institute of Advanced Computer Science. She received her MSc degree in Applied Mathematics from Yaroslavl State University (Russia) in 2004 and her PhD degree in Computer Science from University of Leeds (UK) in 2010. After the total of 5 years of postdoctoral experience at Technical University Eindhoven (Netherlands) and Heriot-Watt University (Edinburgh, UK), Anna spent 5 years working as an engineer and a mathematician in industry. Her current research interests include analysis of optimization algorithms and machine learning.

STEFAN MENZEL received the Dipl.-Ing. degree in civil engineering from RWTH Aachen, Aachen, Germany, in 1998 and the Ph.D. degree in civil engineering from Technical University Darmstadt, Darmstadt, Germany, in 2004.

Since 2004, he is with the Honda Research Institute Europe, Offenbach, Germany, where he is currently a Chief Scientist with the Optimization and Creativity Group. His current research interests include evolutionary optimization with special focus on adaptive representations, machine learning for knowledge transfer and multidisciplinary optimization for real-world applications.

BERNHARD SENDHOFF (SM'05) received the Ph.D. degree in applied physics from Ruhr-Universität Bochum, Bochum, Germany, in 1998.

He was with Honda Research Institute Europe GmbH, Offenbach, Germany, from 2003 to 2010, as a Chief Technology Officer, and from 2011 to 2017, as a President. Since 2017, he has been an Operating Officer with Honda Research and Development Ltd., Tokyo, Japan, and the Head of the Global Operation, Honda Research Institutes. He is an Honorary Professor with the Technical University of Darmstadt, Darmstadt, Germany. He has authored or coauthored over 180 scientific publications. Dr. Sendhoff is a Senior Member of ACM and a member of SAE.

THOMAS BÄCK (M'95-SM'19-F'22) received the Diploma degree in computer science from the University of Dortmund, Germany, in 1990 and the Ph.D. degree in computer science in 1994, also from the University of Dortmund.

Since 2002, he is Full Professor of computer science at the Leiden Institute of Advanced Computer Science (LIACS), Leiden University, The Netherlands. He is author of Evolutionary Algorithms in Theory and Practice (OUP, 1996) and co-editor of the Handbook of Evolutionary Computation (CRC Press, 1997) and Handbook of Natural Computing (Springer, 2012). His research interests include evolutionary computation, machine learning and their real-world applications, especially in sustainable smart industry and health.

Prof. Bäck's awards and honors include membership in the Royal Netherlands Academy of Arts and Sciences (KNAW, 2021), the IEEE Computational Intelligence Society Evolutionary Computation Pioneer Award (2015), Fellow of the International Society of Genetic and Evolutionary Computation (2003), and best Ph.D. thesis award of the German society of Computer Science (GI, 1995).

APPENDIX A ADDITIONAL DESCRIPTION ON THE SPLITTING APPROACH

In this section, we provide a short description on splitting function. AutoML search space is complex due to the number of operators and their choice of algorithms. In practice, the search space can lead up to thousands of algorithm combinations over operators. Since the tuning budget is relatively small vs. a large number of possible pipelines over operators, [13] proposed grouping them based on their similarities with the assumption that a good choice for one algorithm in the group can also serve as a good choice for other algorithms in the group. Consequently, the sampler can maximize coverage of the search space by sampling at the group level instead of the algorithm level.

[13] was mainly focused on initial sampling, where the budget is typically much smaller than the number of combinations of algorithm's groups. As a result, the group's level is limited to only 1, i.e., the group's item is a specific choice of algorithm. In this work, we consider a scenario that a set of algorithms under a group might be slightly different. For example, while algorithms RandomOverSampler and SMOTE are both Oversampling techniques (see the bottom plot in Figure 9), they differ significantly: RandomOverSampler randomly generates more data for minority classes, while SMOTE is based on interpolation. To account for the possible hierarchical groupings of the algorithms, we have extended Algorithm 2 in [13] to allow any group at any level can have child groups. Therefore, the needed groups are produced by downing (or upping) level to minimize randomness. Consequently, the resulting sub-spaces are purer, i.e., The difference between items in a group is minimized, representing their actual relationship.

APPENDIX B ADDITIONAL DETAILS FOR EXPERIMENTAL SETUP

In this section, we present the detailed information on examined datasets, i.e., datasets used in the first experiment (Appendix-Section B-A) and the second experiment (Appendix-Section B-B), as mentioned in Section IV of the main paper.

A. DATASETS USED IN THE FIRST EXPERIMENTAL

The examined datasets in this experiment are taken from the KEEL repository [52] and summarised in Table 3. For each dataset, we include the *Imbalance Ratio* (IR), which is the ratio of the number of majority class instances to that of minority class instances.

B. DATASETS USED IN THE SECOND EXPERIMENTAL

The examined datasets in the second experiment is presented in Table 4. This table includes 73 datasets, taken from OpenML repository [47]. For each task, we include the *OpenML ID* (#task id) and the corresponding dataset (#ID, Name), number of classes (#Classes), number of instances(#Instances), number of features for one instance –Total features (#total), number of numeric (#num) and categorical features (#cate), number of missing values (#Missing values), and number of instances with missing value (#Incomplete instances).

TABLE 3: The number of positive, negative classes, attributes (#Att) and the imbalance ratio (IR) of the KEEL Datasets, ordered by increasing IR value.

DataSets	# Negative	# Positive	#Att	IR
glass1	138	76	9	1.82
ecoli-0_vs_1	77	143	7	1.86
wisconsin	444	239	9	1.86
pima	500	268	8	1.87
iris0	100	50	4	2
glass0	144	70	9	2.06
yeast1	1055	429	8	2.46
haberman	225	81	3	2.78
vehicle2	628	218	18	2.88
vehicle1	629	217	18	2.9
vehicle3	634	212	18	2.99
glass-0-1-2-3_vs_4-5-6	163	51	9	3.2
vehicle0	647	199	18	3.25
ecoli1	259	77	7	3.36
new-thyroid1	180	35	5	5.14
new-thyroid2	180	35	5	5.14
ecoli2	284	52	7	5.46
segment0	1979	329	19	6.02
glass6	185	29	9	6.38
yeast3	1321	163	8	8.1
ecoli3	301	35	7	8.6
page-blocks0	4913	559	10	8.79
yeast-2_vs_4	463	51	8	9.08
yeast-0-5-6-7-9_vs_4	477	51	8	9.35
vowel0	898	90	13	9.98
glass-0-1-6_vs_2	175	17	9	10.29
glass2	197	17	9	11.59
shuttle-c0-vs-c4	1706	123	9	13.87
yeast-1_vs_7	429	30	7	14.3
glass4	201	13	9	15.46
ecoli4	316	20	7	15.8
page-blocks-1-3_vs_4	444	28	10	15.86
abalone9-18	689	42	8	16.4
glass-0-1-6_vs_5	175	9	9	19.44
shuttle-c2-vs-c4	123	6	9	20.5
yeast-1-4-5-8_vs_7	663	30	8	22.1
glass5	205	9	9	22.78
yeast-2_vs_8	462	20	8	23.1
yeast4	1433	51	8	28.1
yeast-1-2-8-9_vs_7	917	30	8	30.57
yeast5	1440	44	8	32.73
ecoli-0-1-3-7_vs_2-6	274	7	7	39.14
yeast6	1449	35	8	41.4
abalone19	4142	32	8	129.44

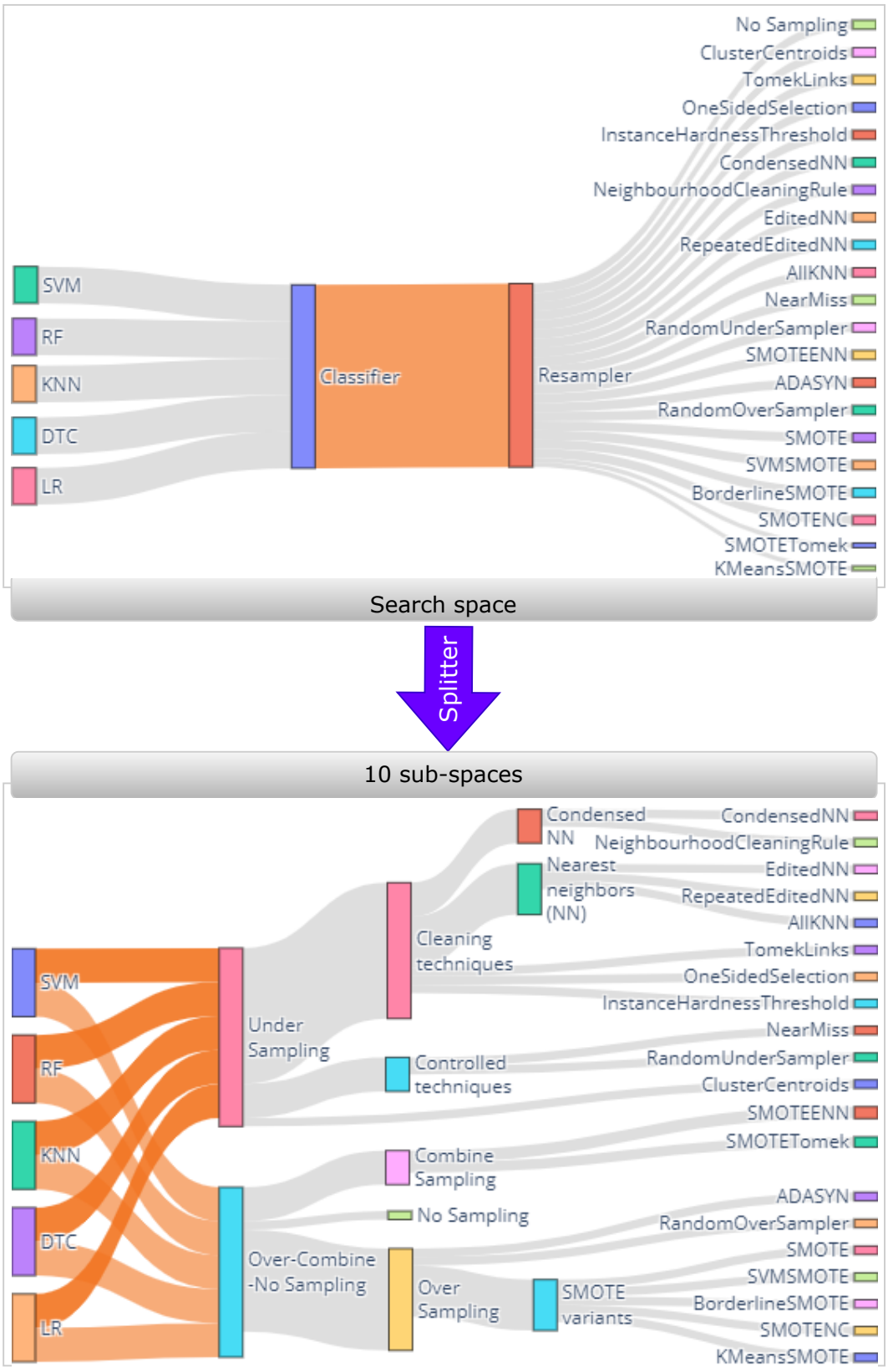

FIGURE 9: Illustration of the Splitting approach on a search space of two operators, i.e., Classifier and Resampler, used in our first experiment. The connection in orange indicates a search space/ sub-space.

TABLE 4: List of 73 datasets used in our second experiment, ordered by increasing OpenML ID.

OpenML	Dataset		#Classes	#Instances	Features			#Missing	#Incomplete
	#ID	name			# total	#num	#cate	values	instances
3	3	kr-vs-kp	2	37	3196	0	37	0	0
12	12	mfeat-factors	10	217	2000	216	1	0	0
15	15	breast-w	2	10	699	9	1	16	16
23	23	cmc	3	10	1473	2	8	0	0
24	24	mushroom	2	23	8124	0	23	2480	2480
29	29	credit-approval	2	16	690	6	10	67	37
31	31	credit-g	2	21	1000	7	14	0	0
3021	38	sick	2	30	3772	7	23	6064	3772
41	42	soybean	19	36	683	0	36	2337	121
53	54	vehicle	4	19	846	18	1	0	0
2079	188	eucalyptus	5	20	736	14	6	448	95
3543	451	irish	2	6	500	2	4	32	32
3560	469	analcatdata_dmf	6	5	797	0	5	0	0
3561	470	profb	2	10	672	5	5	1200	666
3904	1053	jm1	2	22	10885	21	1	25	5
3917	1067	kc1	2	22	2109	21	1	0	0
3945	1111	KDDCup09_appete	2	231	50000	192	39	8024152	50000
3946	1112	KDDCup09_churn	2	231	50000	192	39	8024152	50000
3948	1114	KDDCup09_upsell	2	231	50000	192	39	8024152	50000
189354	1169	airlines	2	8	539383	3	5	0	0
14965	1461	bank-marketing	2	17	45211	7	10	0	0
10101	1464	blood-transfusi	2	5	748	4	1	0	0
9981	1468	cnae-9	9	857	1080	856	1	0	0
9985	1475	first-order-the	6	52	6118	51	1	0	0
9977	1486	nomao	2	119	34465	89	30	0	0
9952	1489	phoneme	2	6	5404	5	1	0	0
9955	1492	one-hundred-pla	100	65	1600	64	1	0	0
7592	1590	adult	2	15	48842	6	9	6465	3620
7593	1596	covertype	7	55	581012	10	45	0	0
9910	4134	Bioresponse	2	1777	3751	1776	1	0	0
34539	4135	Amazon_employee	2	10	32769	0	10	0	0
14952	4534	PhishingWebsite	2	31	11055	0	31	0	0
14969	4538	GesturePhaseSeg	5	33	9873	32	1	0	0
34538	4550	MiceProtein	8	82	1080	77	5	1396	528
14954	6332	cylinder-bands	2	40	540	18	22	999	263
14968	6332	cylinder-bands	2	40	540	18	22	999	263
14967	23380	cjs	6	35	2796	32	3	68100	2795
125920	23381	dresses-sales	2	13	500	1	12	835	401
146606	23512	higgs	2	29	98050	28	1	9	1
167120	23517	numerai28.6	2	22	96320	21	1	0	0
146607	40536	SpeedDating	2	123	8378	59	64	18372	7330
146195	40668	connect-4	3	43	67557	0	43	0	0
167140	40670	dna	3	181	3186	0	181	0	0
146212	40685	shuttle	7	10	58000	9	1	0	0
167141	40701	churn	2	21	5000	16	5	0	0
167121	40923	Devnagari-Scrip	46	1025	92000	1024	1	0	0
167124	40927	CIFAR_10	10	3073	60000	3072	1	0	0
146800	40966	MiceProtein	8	82	1080	77	5	1396	528
146821	40975	car	4	7	1728	0	7	0	0
167125	40978	Internet-Advert	2	1559	3279	3	1556	0	0
146824	40979	mfeat-pixel	10	241	2000	240	1	0	0
146818	40981	Australian	2	15	690	6	9	0	0
146817	40982	steel-plates-fa	7	28	1941	27	1	0	0
146820	40983	wilt	2	6	4839	5	1	0	0
146822	40984	segment	7	20	2310	19	1	0	0
146819	40994	climate-model-s	2	21	540	20	1	0	0
146825	40996	Fashion-MNIST	10	785	70000	784	1	0	0
167119	41027	jungle_chess_2p	3	7	44819	6	1	0	0
168868	41138	APSFailure	2	171	76000	170	1	1078695	75244
168908	41142	christine	2	1637	5418	1599	38	0	0
168911	41143	jasmine	2	145	2984	8	137	0	0
168912	41146	sylvine	2	21	5124	20	1	0	0
189356	41147	albert	2	79	425240	26	53	2734000	425159
168335	41150	MiniBooNE	2	51	130064	50	1	0	0
168337	41159	guillermo	2	4297	20000	4296	1	0	0
168338	41161	riccardo	2	4297	20000	4296	1	0	0
168909	41163	dilbert	5	2001	10000	2000	1	0	0
168910	41164	fabert	7	801	8237	800	1	0	0
168332	41165	robert	10	7201	10000	7200	1	0	0
168331	41166	volkert	10	181	58310	180	1	0	0
189355	41167	dionis	355	61	416188	60	1	0	0
168330	41168	jannis	4	55	83733	54	1	0	0
168329	41169	helena	100	28	65196	27	1	0	0

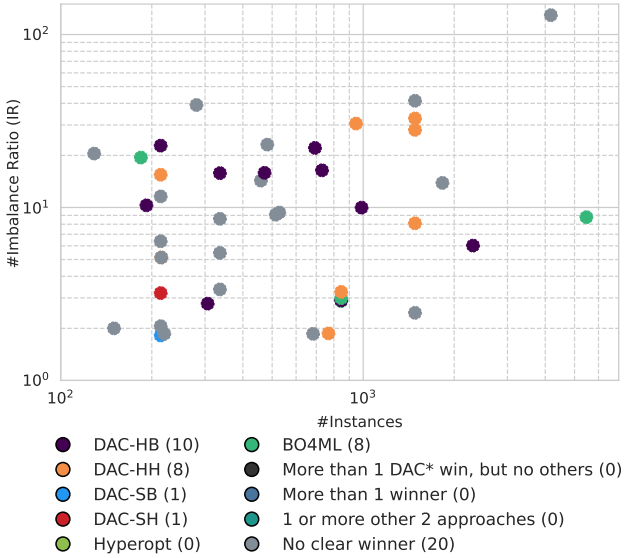

FIGURE 10: Overview of winner across 10 repetitions on 44 binary-class imbalanced benchmark datasets. The scatter plot distribution of Imbalance Ratio (#IR) vs. the number of samples (#instances) on a logarithms scales. The color indicates the corresponding winner. Figure best viewed in color.

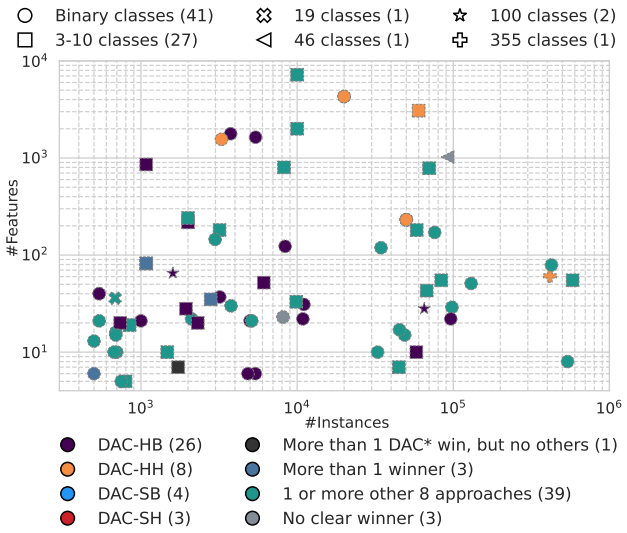

FIGURE 11: Overview of winner across 10 repetitions for the 73 AutoML benchmark examined datasets. The scatter plot distribution of the number of features (#features) vs. the number of samples (#instances) on a logarithms scale. The color indicates the corresponding winner. Figure best viewed in color.

APPENDIX C ADDITIONAL DETAILS FOR EXPERIMENTAL RESULTS

In this section, we provide the detailed results for the two experimental setups as shown in Figure 6 and Figure 8 in the main paper. In subsequent Tables 5 and 6 for both experiments, the highest performance for the corresponding dataset is highlighted in **bold**. The method performs significantly worse than the best according to the Wilcoxon sign-rank test with $\alpha = 0.05$ is underlined. Two extra rows at the end of the corresponding table display additional summaries. The first extra row shows the number of times each scenario got the highest value over tested datasets. The last extra row indicates the number of times each approach was significantly better than the other in group.

A. FIRST EXPERIMENT RESULTS

The results of the first experiment are presented in Table 5 to illustrate the performance between 2 BO variants based on TPE surrogate model with and without proposed contesting procedures using 2 elimination criteria – highest performance and statistical test procedure, i.e., DAC-HB, DAC-HH, DAC-SB, DAC-SH, BO4ML and Hyperopt, respectively. Additionally, the distribution of the best found optimizer over 44 tested datasets is summarized in Figure 10.

B. SECOND EXPERIMENT RESULTS

In this experiment, we compared all approaches used in the first experiment to the current state-of-the-art AutoML frameworks, i.e., Auto-sklearn-SMAC (Auto-sklearn) and Auto-sklearn-Random search (Random), HPsklearn [11], TPOT [43], ATM [44], H20 [45], based on the results obtained by [13], [46]. The detailed results of the second tested scenarios are presented in Table 6. We note that entries with missing values in the last 6 columns indicates arbitrary fails reported by [46]. Additionally, the distribution of the best found optimizer over 73 examined datasets is summarized in Figure 11.

TABLE 5: Average Geometric mean (rounded to 4 decimals) based on six approaches, i.e., DAC-HB, DAC-HH, DAC-SB, DAC-SH, BO4ML and Hyperopt, over 10 repetitions for the 44 examined datasets, ordered by increasing imbalance ratio (#IR) value.

Dataset	#IR	DAC-HB	DAC-HH	DAC-SB	DAC-SH	BO4ML	Hyperopt
glass1	1.82	0.8015	0.8004	**0.804**	0.7945	0.7922	0.7968
ecoli-0_vs_1	1.86	0.9864	0.9864	0.9864	0.9864	**0.9868**	0.9864
wisconsin	1.86	0.9813	0.9816	0.9813	0.9814	0.9814	**0.9819**
pima	1.87	0.769	**0.7725**	0.768	0.7719	0.7694	0.7705
iris0	2.0	**1**	**1**	**1**	**1**	**1**	**1**
glass0	2.06	0.876	**0.8777**	0.8736	0.8757	0.8736	0.8745
yeast1	2.46	0.7332	0.7333	0.7322	0.7321	**0.7335**	0.7325
haberman	2.78	**0.7057**	0.701	0.7023	0.6974	0.6968	0.7012
vehicle2	2.88	0.9903	0.991	0.9902	0.9898	**0.9912**	0.991
vehicle1	2.9	**0.8709**	0.8707	0.8512	0.8445	0.862	0.8701
vehicle3	2.99	0.84	0.8476	0.8166	0.8202	**0.848**	0.8461
glass-0-1-2-3_vs_4-5-6	3.2	0.9571	0.9568	0.9545	**0.9572**	0.9562	0.9514
vehicle0	3.25	0.9865	**0.9868**	0.9837	0.9837	0.9864	0.9855
ecoli1	3.36	0.9034	0.9047	0.9036	0.9029	0.9031	**0.9047**
new-thyroid1	5.14	0.9975	**0.9986**	0.9966	0.9972	0.9972	0.9978
new-thyroid2	5.14	0.9975	**0.9978**	0.9972	0.9972	0.9975	0.9978
ecoli2	5.46	0.9375	**0.9375**	0.9365	0.9362	0.9361	0.9358
segment0	6.02	**0.9993**	0.9993	0.9992	0.9992	0.9991	0.9993
glass6	6.38	0.952	**0.9547**	0.9516	0.9503	0.9489	0.9524
yeast3	8.1	0.9436	**0.9437**	0.9422	0.942	0.9428	0.9425
ecoli3	8.6	0.9075	0.9079	0.907	0.9072	0.9054	**0.9091**
page-blocks0	8.79	0.9471	0.9467	0.9468	0.9463	**0.948**	0.9472
yeast-2_vs_4	9.08	0.9535	0.952	0.9533	0.951	**0.9538**	0.9533
yeast-0-5-6-7-9_vs_4	9.35	0.8145	**0.8258**	0.8146	0.8169	0.8238	0.8195
vowel0	9.98	**0.9628**	0.9569	0.9598	0.9554	0.9564	0.9555
glass-0-1-6_vs_2	10.29	**0.8515**	0.8424	0.845	0.8359	0.8436	0.8342
glass2	11.59	0.8593	0.8546	**0.8601**	0.8534	0.8578	0.856
shuttle-c0-vs-c4	13.87	**1**	**1**	**1**	**1**	**1**	**1**
yeast-1_vs_7	14.3	0.8017	**0.8043**	0.8003	0.8026	0.8017	0.8001
glass4	15.46	0.9355	**0.9372**	0.9291	0.935	0.9192	0.9334
ecoli4	15.8	**0.9743**	0.9709	0.9701	0.9637	0.9661	0.9698
page-blocks-1-3_vs_4	15.86	**0.9944**	0.9889	0.9929	0.9882	0.9901	0.99
abalone9-18	16.4	**0.8951**	0.8864	0.8834	0.8846	0.8873	0.8838
glass-0-1-6_vs_5	19.44	0.9655	0.9535	0.9588	0.9597	**0.9681**	0.9644
shuttle-c2-vs-c4	20.5	**1**	**1**	**1**	**1**	**1**	**1**
yeast-1-4-5-8_vs_7	22.1	**0.7166**	0.7037	0.7155	0.7011	0.6979	0.7011
glass5	22.78	**0.9716**	0.9699	0.9659	0.96	0.9625	0.96
yeast-2_vs_8	23.1	0.8242	0.8259	0.8135	0.8117	**0.828**	0.8254
yeast4	28.1	0.8794	**0.8812**	0.8694	0.8726	0.8708	0.8773
yeast-1-2-8-9_vs_7	30.57	0.7515	**0.7546**	0.7416	0.7459	0.7391	0.7429
yeast5	32.73	0.9801	**0.9806**	0.9795	0.9796	0.9801	0.9798
ecoli-0-1-3-7_vs_2-6	39.14	0.866	0.8799	0.8892	0.9035	0.9034	**0.9057**
yeast6	41.4	0.9007	**0.9018**	0.8994	0.9003	0.897	0.9004
abalone19	129.44	**0.8059**	0.8031	0.8022	0.8003	0.8049	0.8021
Cases achieved the highest values		14	18	5	4	11	7
Significant wins over other approaches		10	8	1	1	4	0

TABLE 6: Average accuracy (rounded to 5 decimals) over 10 repetitions for the 73 OpenML datasets, ordered by #Task id. The first fourth columns after "Dataset" shows our experimental results, i.e., 4 variants of the contesting procedure. The remaining columns contain results obtained by other AutoML frameworks according to [13] and [46].

OpenML IDs		DACOpt contesting procedure				[13]		[46]					
#TaskID	Dataset(ID)	DAC-HB	DAC-HH	DAC-SB	DAC-SH	BO4ML	Hyperopt	Auto sklearn	Random search	HP sklearn	TPOT	ATM	H2O
3	kr-vs-kp(3)	**0.99802**	0.99666	0.99802	0.99666	0.99656	0.9951	0.98986	0.99062	0.99051	0.99431	0.99326	0.99426
12	mfeat-factors(12)	**0.98617**	0.98383	0.98617	0.98383	0.98417	0.98117	0.97767	0.97633	0.94758	0.97333	0.98178	0.97433
15	breast-w(15)	0.98095	0.97524	0.98095	0.97524	0.97952	0.97048	0.96875	0.95873	0.96	0.96571	**0.98474**	0.96286
23	cmc(23)	0.57376	0.57805	0.57376	0.57805	0.57285	0.55158	0.54638	0.53262	0.53047	0.55882	**0.581**	0.53733
24	mushroom(24)	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**0.99993**	**1**	**1**	**1**	0.99848
29	credit-approval(29)	0.88647	0.87778	0.88647	0.87778	0.88744	0.86522	0.87289	0.85507	0.85956	0.86377	**0.89133**	0.86184
31	credit-g(31)	**0.767**	0.74533	0.767	0.74533	0.766	0.72733	0.73433	0.724	0.70121	0.744	0.76578	0.74867
41	soybean(42)	0.94927	0.93659	0.94927	0.93659	**0.95171**	0.92878	0.91954	0.91911	0.92585	0.92732	0.94504	0.93122
53	vehicle(54)	0.86299	0.8374	0.86299	0.8374	**0.86457**	0.83858	0.82008	0.81969	0.75787	0.81811	0.81522	0.82717
2079	eucalyptus(188)	**0.69864**	0.69548	0.69864	0.69548	0.69502	0.66018	0.63886	0.6267	0.64072	0.65566	0.6419	0.6557
3021	sick(38)	0.99134	0.9909	0.99134	0.9909	**0.99152**	0.98737	0.98288	0.9855	0.97438	0.98746		0.98419
3543	irish(451)	**1**	**1**	**1**	**1**	**1**	**1**	0.99019	0.99081	0.99404	0.99091	**1**	0.97967
3560	analcatdata_dmf(469)	0.2375	0.22417	0.2375	0.22417	0.23042	0.21125	0.20365	0.20382	0.19139	0.20833	**0.27028**	0.19542
3561	profb(470)	0.69901	0.67228	0.69901	0.67228	0.63119	0.64752	0.65687	0.64563	0.63762	0.66832	**0.71221**	0.71089
3904	jm1(1053)	**0.82535**	0.82171	0.82535	0.82171	0.82404	0.81393	0.81344	0.81126	0.80998	0.8181	0.821	0.74819
3917	kc1(1067)	0.87172	0.86572	0.87172	0.86572	**0.87393**	0.85972	0.85118	0.8534	0.84044	0.86019	0.86856	0.80869
3945	KDDCup09_appete(1111)	**0.98325**	0.98285	0.98325	0.98285	0.98323	0.98197	0.98244	0.98228	0.98189	0.98182		0.96555
3946	KDDCup09_churn(1112)	0.92863	0.92809	0.92863	0.92809	**0.92901**	0.92624	0.92725	0.92586	0.92599	0.92624		0.78802
3948	KDDCup09_upsell(1114)	0.9506	**0.95137**	0.9506	0.95137	0.94345	0.94116	0.95094	0.9503	0.95068	0.95085		0.93415
7592	adult(1590)	0.86906	0.86527	0.86906	0.86527	0.86251	0.85769	0.86938	0.87013	0.86727	**0.87089**	0.85448	0.86656
7593	covertype(1596)	0.87738	0.93199	0.87738	0.93199	0.70278	0.80902	**0.96395**	0.89143	0.95227	0.94542	0.6639	0.92908
9910	Bioresponse(4134)	**0.80595**	0.80062	0.80595	0.80062	0.80107	0.78073	0.7889	0.77762	0.77798	0.80249	0.77087	0.80044
9952	phoneme(1489)	**0.91726**	0.91196	0.91726	0.91196	0.91319	0.90826	0.89716	0.89205	0.89273	0.9045	0.89963	0.89205
9955	one-hundred-pla(1492)	**0.69562**	0.6775	0.69562	0.6775	0.67167	0.65146	0.65172	0.62795	0.54667	0.61146	0.61097	0.56435
9977	nomao(1486)	0.97132	0.97098	0.97132	0.97098	0.96525	0.95924	0.96903	0.96656	0.96891	0.97026	0.96055	**0.97146**
9981	cnae-9(1468)	**0.96235**	0.95432	0.96235	0.95432	0.95093	0.94228	0.94167	0.93117	0.94012	0.94784	0.96049	0.95216
9985	first-order-the(1475)	**0.61983**	0.61133	0.61983	0.61133	0.61029	0.59853	0.59695	0.58601	0.58293	0.61291	0.60272	0.61656
10101	blood-transfusi(1464)	0.80622	0.79644	0.80622	0.79644	0.80667	0.76578	0.76667	0.77778	0.78044	0.78711	**0.81956**	0.73378
14952	PhishingWebsite(4534)	**0.97422**	0.97272	0.97422	0.97272	0.97094	0.96623	0.9659	0.96244	0.96964	0.96913	0.96464	0.9716
14954	cylinder-bands(6332)	0.83395	0.82037	0.83395	0.82037	**0.83642**	0.81111	0.79012	0.76173	0.76667	0.81009	0.81701	0.78333
14965	bank-marketing(1461)	0.90695	0.90625	0.90695	0.90625	0.90307	0.90007	0.90447	0.90398	0.90451	**0.90705**	0.89957	0.9006
14967	cjs(23380)	**1**	**1**	**1**	**1**	**1**	**1**	0.98265	0.99841	0.97131	**1**		**1**
14968	cylinder-bands(6332)	**0.84259**	0.84074	0.84259	0.84074	0.8358	0.80432	0.77353	0.77058	0.75823	0.81173	0.79155	0.8
14969	GesturePhaseSeg(4538)	0.66722	0.6812	0.66722	0.6812	0.64001	0.61864	0.67733	0.65004	0.67272	0.67586	0.66217	**0.70165**
34538	MiceProtein(4550)	**1**	**1**	**1**	**1**	**1**	0.99907	**1**	0.99907	0.99983	**1**	**1**	**1**
34539	Amazon_employee(4135)	0.94915	0.94702	0.94915	0.94702	0.94825	0.94557	0.94761	0.94444	0.9475	0.94891	0.94606	**0.95114**
125920	dresses-sales(23381)	0.61533	0.6	0.61533	0.6	0.63133	0.562	0.56667	0.55556	0.56844	0.56867	**0.66978**	0.584
146195	connect-4(40668)	0.82042	0.82628	0.82042	0.82628	0.77358	0.77321	0.82109	0.79628	0.82886	0.84123	0.77698	**0.865**
146212	shuttle(40685)	**0.9999**	0.99989	0.9999	0.99989	0.99965	0.99945	0.99978	0.99968	0.99253	0.99974	0.99955	0.99987
146606	higgs(23512)	0.7115	0.71645	0.7115	0.71645	0.70605	0.69761	**0.72296**	0.7193	0.70743	0.72031	0.67135	0.71281
146607	SpeedDating(40536)	**0.87383**	0.8708	0.87383	0.8708	0.86611	0.85871	0.86291	0.86225	0.86661	0.86392	0.86128	0.84968
146800	MiceProtein(40966)	**1**	**1**	**1**	**1**	0.99969	0.99321	0.99043	0.99506	0.9638	0.99506	**1**	0.99551
146817	steel-plates-fa(40982)	**0.80892**	0.79966	0.80892	0.79966	0.80497	0.78216	0.78268	0.76364	0.75955	0.79091	0.76415	0.78062
146818	Australian(40981)	0.88889	0.88261	0.88889	0.88261	0.88647	0.85845	0.87053	0.85556	0.86913	0.86184	**0.8905**	0.87633
146819	climate-model-s(40994)	0.96728	0.96605	0.96728	0.96605	0.95802	0.93951	0.94074	0.92407	0.92593	0.94547	**0.96975**	0.93642
146820	wilt(40983)	**0.98864**	0.98747	0.98864	0.98747	0.97355	0.97906	0.98612	0.98581	0.95289	0.9854	0.98657	0.98574
146821	car(40975)	**0.99981**	0.99884	**0.99981**	0.99884	0.99441	0.98748	0.97264	0.97958	0.98786	0.99422	0.96763	0.99191
146822	segment(40984)	**0.94603**	0.94055	0.94603	0.94055	0.94473	0.93189	0.93088	0.93333	0.90664	0.94055	0.92564	0.94185
146824	mfeat-pixel(40979)	0.9835	0.98233	0.9835	0.98233	**0.98433**	0.98117	0.97783	0.97367	0.98121	0.96883	0.9775	0.976
146825	Fashion-MNIST(40996)	0.85193	0.86744	0.85193	0.86744	0.843	0.83891	**0.87844**	0.8445	0.8506	0.78089	0.82114	0.87341
167119	jungle_chess_2p(41027)	0.84892	0.86674	0.84892	0.86674	0.84647	0.83956	0.86775	0.85378	0.88691	0.88735	0.8754	**0.90047**
167120	numerai28.6(23517)	**0.52457**	0.52385	0.52457	0.52385	0.52257	0.52134	0.51926	0.51939	0.52033	0.52082	0.51941	0.50635
167121	Devnagari-Scrip(40923)	0.7812	0.89001	0.7812	0.89001	0.86652	0.7491	0.74009	0.02169	0.86438		**0.8947**	0.5822
167124	CIFAR_10(40927)	0.33284	**0.40956**	0.33284	0.40956	0.39675	0.37813			0.32093	0.29429	0.32001	0.36389
167125	Internet-Advert(40978)	0.97856	**0.97876**	0.97856	0.97876	0.97713	0.97033	0.97774	0.97114	0.97358	0.97398	0.969	
167140	dna(40670)	0.96475	0.96287	0.96475	0.96287	0.96485	0.95397	0.95962	0.95889	0.96109	0.95931	0.95282	**0.96904**
167141	churn(40701)	**0.9636**	0.96133	0.9636	0.96133	0.96273	0.95367	0.9562	0.95313	0.94533	0.96	0.95007	0.9537
168329	helena(41169)	**0.33594**	0.32871	0.33594	0.32871	0.3169	0.29294	0.30692	0.29566	0.28741	0.33576	0.32108	
168330	jannis(41168)	0.68921	0.70161	0.68921	0.70161	0.68479	0.6667	**0.71814**	0.69273	0.68494	0.69642	0.63788	0.71786
168331	volkert(41166)	0.6319	0.65658	0.6319	0.65658	0.60445	0.5952	0.66933	0.63762	0.65451	0.65075	**0.6794**	0.67841
168332	robert(41165)	0.39963	0.44613	0.39963	0.44613	0.42507	0.38497	**0.44843**	0.39922	0.34203		0.35252	
168335	MiniBooNE(41150)	0.93435	0.93889	0.93435	0.93889	0.92248	0.91035	0.94334	0.92891	0.87477	0.9385	0.90234	**0.94604**
168337	guillermo(41159)	0.7521	0.81453	0.7521	0.81453	0.78205	0.72707	0.64227		0.74347	0.72548	0.66063	**0.81928**
168338	riccardo(41161)	0.9701	**0.99552**	0.9701	0.99552	0.98303	0.98035	0.74757	0.75042	0.82518	0.98495	0.90729	0.95625
168868	APSFailure(41138)	0.99183	0.99318	0.99183	0.99318	0.98985	0.989	0.99287	0.99137	0.9936	0.99339	0.97097	**0.99369**
168908	christine(41142)	**0.75609**	0.75166	0.75609	0.75166	0.73659	0.72565	0.74754	0.73081	0.7163	0.72645	0.72169	0.72811
168909	dilbert(41163)	0.96513	0.983	0.96513	0.983	0.9504	0.94437	**0.98357**	0.94793	0.97243	0.96254	0.95391	0.96988
168910	fabert(41164)	0.69563	0.69308	0.69563	0.69308	0.67565	0.66177	0.70255	0.67395	0.69104	0.68336	0.67357	**0.71752**
168911	jasmine(41143)	0.82578	0.81942	0.82578	0.81942	**0.83259**	0.80748	0.82009	0.80603	0.80078	0.82366	0.79911	0.80906
168912	sylvine(41146)	0.95033	0.95273	0.95033	0.95273	**0.95709**	0.94655	0.93921	0.94753	0.94675	0.95533	0.93476	0.9251
189354	airlines(1169)	0.65866	0.6634	0.65866	0.6634	0.64626	0.63957	0.66665	0.59845	0.6508	**0.66895**	0.63671	0.61266
189355	dionis(41167)	0.81678	**0.82433**	0.81678	0.82433	0.73916	0.68112			0.77971		0.38666	
189356	albert(41147)	0.65874	0.66576	0.65874	0.66576	0.6481	0.64737	0.68314	0.66709	0.66694	0.6611	**0.80064**	0.64798
Cases with the highest values achieved		28	10	6	5	13	3	8	1	1	6	16	12
Significant wins over other approaches		26	8	4	3	11	2	6	0	0	4	13	11