Automated Machine Learning with Monte-Carlo Tree Search

Herilalaina Rakotoarison, Marc Schoenauer, Michèle Sebag

Automated Machine Learning with Monte-Carlo Tree Search

Herilalaina Rakotoarison , **Marc Schoenauer** and **Michèle Sebag**
TAU, LRI-CNRS–INRIA
Université Paris-Saclay, France
{herilalaina.rakotoarison, marc.schoenauer}@inria.fr, sebag@lri.fr

Abstract

The AutoML task consists of selecting the proper algorithm in a machine learning portfolio, and its hyperparameter values, in order to deliver the best performance on the dataset at hand. MOSAIC, a Monte-Carlo tree search (MCTS) based approach, is presented to handle the AutoML hybrid structural and parametric expensive black-box optimization problem. Extensive empirical studies are conducted to independently assess and compare: i) the optimization processes based on Bayesian optimization or MCTS; ii) its warm-start initialization; iii) the ensembling of the solutions gathered along the search. MOSAIC is assessed on the OpenML 100 benchmark and the Scikit-learn portfolio, with statistically significant gains over AUTO-SKLEARN, winner of former international AutoML challenges.

1 Introduction

The automated selection of the machine learning (ML) algorithm yielding the best performance on the problem at hand, referred to as AutoML, has attracted interest since the late 1980s [Brazdil and Giraud-Carrier, 2018]: there exists no killer ML algorithm dominating all others on all datasets [Wolpert, 1996], and ML algorithms demonstrate a high sensitivity w.r.t. their hyper-parameters. With the explosion of machine learning applications, the AutoML issue becomes even more acute. AutoML gradually extended to hyper-parameters optimization [Bergstra *et al.*, 2011], and finally tackles the optimization of the overall ML pipeline from data preparation to model learning [Feurer *et al.*, 2015; Li *et al.*, 2017; Olson *et al.*, 2016; Chen *et al.*, 2018]. Several AutoML international challenges have been organized in the last decade [Guyon *et al.*, 2015; Guyon *et al.*, 2018], spurring the development of efficient AutoML systems such as AUTO-WEKA [Kotthoff *et al.*, 2017], HYPERBAND [Li *et al.*, 2017], TPOT [Olson *et al.*, 2016] and the challenge winner AUTO-SKLEARN [Feurer *et al.*, 2015] (more in section 2).

AutoML systems tackle a black-box expensive optimization problem: For a given target dataset,

$$\text{Find } \mathbf{x}^* \in \arg\max_{x \in X} \mathcal{F}(\mathbf{x}), \tag{1}$$

where X is the structural and parametric space of ML configurations (containing categorical and continuous parameters with hierarchical dependencies), and $\mathcal{F}(\mathbf{x})$ the performance of the model learned from the dataset at hand using configuration $\mathbf{x}$. ML configurations and pipelines are used interchangeably in the following.

A main difficulty of the AutoML optimization problem lies in the search space: an ML pipeline is a series of components (algorithms), together with their own hyper-parameters. The task thus consists in solving the combinatorial optimization of the pipeline structure, the performance of which depends on the parametric optimization of its component hyper-parameters.

Most AutoML approaches tackle both problems using a single optimization approach technique (e.g., Bayesian Optimization or Evolutionary Algorithms) whereas both problems are of very different nature. The contribution of the paper, presenting the MOSAIC (*MOnte-Carlo tree Search for AlgoRIthm Configuration*) approach,[1] is to use the best approach for each problem while tightly coupling both optimizations (section 3).

Specifically, the optimization of the pipeline can be viewed as a sequential decision process; Monte-Carlo Tree Search (MCTS) [Kocsis and Szepesvári, 2006] has demonstrated its ability to efficiently solve such sequential problems. On the other hand, Bayesian optimization [Mockus *et al.*, 1978; Wang, 2016] has been very successful solving expensive optimization problems, in particular in the context of hyper-parameters tuning [Hutter *et al.*, 2011]. These two approaches are coupled in MOSAIC and their coupling relies on a surrogate model of the performance of the pipelines, as in AUTO-SKLEARN. However, this surrogate model is not only used to guide the local search of the hyper-parameters, it is also incorporated at the heart of the MCTS search of the best pipeline structure.

The paper is organized as follows. Section 2 discusses the state of the art in AutoML, and presents the MCTS formal background. Section 3 gives a detailed overview of the proposed MOSAIC approach. The experimental setting and the goals of experiments are presented in Section 4. Section 5 reports on the empirical validation[2] of MOSAIC on the OpenML

[1] MOSAIC is publicly available under an open source license at https://github.com/herilalaina/mosaic_ml.

[2] We warmly thank AUTO-SKLEARN authors, who kindly pro-

benchmark suite and the Scikit-learn portfolio, demonstrating statistically significant gains over AUTO-SKLEARN and TPOT [Olson *et al.*, 2016].

2 Related Work

This section briefly reviews previous work on the *per-instance* AutoML problem (Eq. (1)), first focusing on approaches using surrogate models and Bayesian Optimisation, then on MCTS and other approaches. Approaches focused on specific issues, e.g., neural architecture optimization [Wistuba, 2018], are omitted due to space limitations.

2.1 Surrogate Model-based Optimization

Most prominent approaches today proceed iteratively, learning and exploiting an estimate of the optimization objective $\mathcal{F}$, called *surrogate model*.

Learning a Surrogate Model

At step t, surrogate model $\widehat{\mathcal{F}}_t : X \mapsto \mathbb{R}$ is learned from the set $\{(x_u, \mathcal{F}(x_u)), u = 1 \dots t\}$ gathering the previously selected configurations and their associated performances. $\widehat{\mathcal{F}}_t$ is then used to determine the most promising candidate x_{t+1}, see below.

As said, a main difficulty lies in the structure of space X. In all generality, this space includes categorical features (e.g., the name of the ML algorithm, the type of pre-processing) and continuous or integer features, the number and range of which depend on the value of the categorical features (e.g. the algorithm or pre-processing method). Diverse surrogate model hypothesis spaces have been considered: the *Sequential Model-based Algorithm Configuration* (SMAC) [Hutter *et al.*, 2011], like AUTO-WEKA [Kotthoff *et al.*, 2017] and AUTO-SKLEARN [Feurer *et al.*, 2015], are based on Random Forests; [Bergstra *et al.*, 2011] use a Tree-structure Parzen Estimator (TPE); SPEARMINT [Snoek *et al.*, 2015] is based on Gaussian processes (GP). An extensive comparison of these approaches [Eggensperger *et al.*, 2013] shows that SMAC and TPE perform best for high dimensional and mixed hyperparameter optimization problems, while the GP-based SPEARMINT performs best on low dimensional continuous search spaces.

Surrogate Model-Based Optimization

Surrogate models are often exploited along *Bayesian optimization* (BO) [Mockus *et al.*, 1978; Wang, 2016]. Assuming that model $\widehat{\mathcal{F}}_t$ yields the performance distribution for any given $\mathbf{x}$, the most promising $\mathbf{x}^*_{t+1}$ is determined by maximizing the expected improvement on the current best value $\mathcal{F}(\mathbf{x}^*_t)$ [Mockus *et al.*, 1978], or more generally an acquisition function balancing performance expectation and variance [Wang, 2016].

A simple alternative is to learn a surrogate model as a random forest, yielding both a performance estimate and a variance estimate for any configuration. The next candidate $\mathbf{x}_{t+1}$ is the configuration maximizing the approximate acquisition function, out of a number of configuration samples. The key issue here is the distribution used to sample the configuration space. For instance, AUTO-SKLEARN, as it uses SMAC, considers a small number of configurations close to the best-so-far configuration, augmented with a large number of uniformly sampled configurations.

2.2 Monte-Carlo Tree Search

An alternative to Bayesian optimization is based on Monte-Carlo Tree Search [Kocsis and Szepesvári, 2006]. Considering a tree-structured search space X, MCTS iteratively explores the space, gradually biasing the exploration toward the most promising regions of the search tree. Each iteration, referred to as tree-walk (Fig. 1), involves four phases [Gelly and Silver, 2011]:

Down the MCTS tree. The first phase traverses the MCTS tree from the root node. In each (non-leaf) node $\mathbf{s}$ of the tree, the next node $\mathbf{s}.a$ to visit is classically selected among the child nodes of $\mathbf{s}$ using the multi-armed bandit Upper Confidence Bound criterion [Auer, 2002]:

$$\text{select } \arg\max_a \left\{ \hat{\mu}_{s.a} + C_{ucb} \sqrt{\frac{\log n(s)}{n(s.a)}} \right\} \qquad (2)$$

with $\hat{\mu}_{s.a}$ the average reward gathered over all tree-walks with prefix $\mathbf{s}.a$, $n(\mathbf{s})$ (resp. $n(\mathbf{s}.a)$) the number of visits to node $\mathbf{s}$ (resp. node $\mathbf{s}.a$), and C_{ucb} a problem-dependent constant that controls the exploitation *vs* exploration trade-off;

Expansion. When arriving at a leaf node, a new child node might be added. The choice of the new node can be guided using e.g. the *Rapid Action Value Estimate* [Gelly and Silver, 2011]. The number of child nodes is controlled and gradually extended along the Progressive Widening strategy [Auger *et al.*, 2013]: A new child node is added whenever the integer value of $n(s)^{PW}$ increases by one, PW being a user-defined parameter (typically 0.6).

Playout. After the expansion phase, a playout strategy is used to complete the tree-walk until reaching a terminal node and computing the associated reward;

Back-propagation. The reward value is back-propagated along the current path, incrementing $n(\mathbf{s})$ for all visited nodes and updating $\hat{\mu}_s$ accordingly.

Taking inspiration from ALPHAGO ZERO [Silver *et al.*, 2017], the ALPHAD3M system builds upon MCTS to explore the pipeline search space [Drori *et al.*, 2018]. The difference compared to mainstream AutoML systems is twofold. Firstly, ALPHAD3M explores the sequences of actions (insertion, deletion, replacement of pipeline parts) on pipelines, as opposed to directly exploring space X. Secondly, ALPHAD3M learns (resp. exploits) a recurrent neural net to encode the action probability of success (resp. probability of selection) conditioned on the current state, *in lieu* of surrogate model or selection rule.

vided many explanations together with their open source code. We also thank TPOT authors, who provide an open source easy-to-use software package.

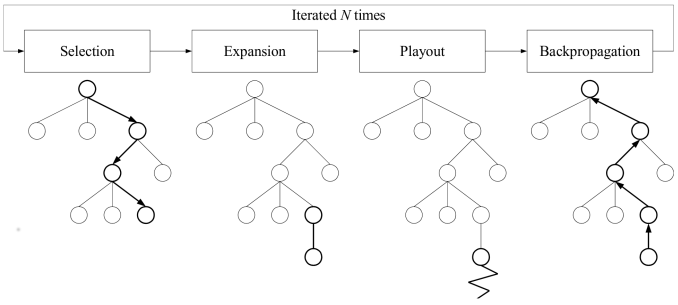

Figure 1: Monte-Carlo Tree Search: each iteration involves four phases [Chaslot *et al.*, 2008].

2.3 Expensive Optimization

As said, AutoML is an expensive black-box optimization problem: computing $\mathcal{F}(\mathbf{x})$ amounts to run the whole ML pipeline $\mathbf{x}$ on the considered dataset. Several approaches have been proposed to reduce the computational cost. A first one consists of sub-sampling the training dataset [Swersky *et al.*, 2014; Li *et al.*, 2017; Klein *et al.*, 2017]. Two surrogate models are built in [Klein *et al.*, 2017]: one for the performance reached depending on the configuration $\mathbf{x}$ and a fraction ρ of the training set considered, another one for the actual computational cost of running $\mathbf{x}$ on a fraction ρ of the data. Both models are jointly exploited to determine the most promising pipeline in terms of performance improvement and moderate computational cost.

Another approach is HYPERBAND [Li *et al.*, 2017], launching a large number of random candidate configurations, subject to a given cut-off time. HYPERBAND iteratively prunes the unpromising candidates, and re-examines the other candidates with a larger cut-off, until the best candidates are allowed to run with no computational cost constraint. After its authors, HYPERBAND outperforms SMAC and TPE for hyper-parameters optimization on neural networks and support vector machines, though its performances are sensitive to its own hyper-hyper-parameters.

Two evolutionary approaches (EAs) have been proposed, handling particular ML pipelines. TPOT uses Genetic Programming to evolve pipelines made of parallel preprocessing and feature construction branches, that feed some model building method. A comparative study [Balaji and Allen, 2018] reports that TPOT is outperformed by AUTO-SKLEARN on classification problems while the reverse is true on regression problems. AUTOSTACKER [Chen *et al.*, 2018] builds an ML pipeline by evolving new artificial features, and adding them to the original dataset. The whole stack is optimized using a vanilla EA with *ad hoc* mutation and crossover. AUTOSTACKER outperforms TPOT, and yields some better results than AUTO-SKLEARN, though both algorithms have very different ways of handling CPU time.

2.4 Search Initialization and Solution Aggregation

It is long known that initialization is a most critical step for ill-posed optimization problems. The selection of the first candidates $\mathbf{x}_u$ will govern the quality of the surrogate model (section 2.1) and the time-to-good configurations: the better the initial $\mathbf{x}_u$s, the more accurate the surrogate model will be *in the worthy part of the search space*[3].

The selection of the initial $\mathbf{x}_u$s in AUTO-SKLEARN is based on the so-called *MetaLearning* heuristics. Formally, AUTO-SKLEARN is provided with an archive, gathering pairs $(\mathbf{z}_i, \mathbf{x}_i)$ where the meta-feature[4] vector $\mathbf{z}_i$ describes the i-th dataset and $\mathbf{x}_i$ is the best known pipeline for this dataset. Letting $\mathbf{z}$ denote the meta-feature vector associated to the current dataset, its nearest neighbors in the archive (in the sense of the Euclidean distance on the meta-feature vector space) are computed and the $\mathbf{x}_i$s associated with these neighbors are used by AUTO-SKLEARN as first configurations [Feurer *et al.*, 2015].

Finally, the sequence of solutions found by an AutoML process can be exploited in the spirit of ensemble learning [Caruana *et al.*, 2004]. AUTO-SKLEARN.Ensemble delivers the compound model defined as the weighted sum of the models learned along the search, where the weights are optimized on a validation set.

3 MCTS-aided Algorithm Configuration

After introducing some notations, this section presents MOSAIC and discusses its components.

An ML pipeline $\mathbf{x}$ involves a fixed ordered sequence of ℓ decisions, respectively selecting the data preprocessing (including categorical variable encoding, missing value imputation, rescaling), feature selection, and learning algorithms. At the i^{th} decision step, some algorithm $a_i \in \mathcal{A}_i$ is selected (with $\mathcal{A}_i$ the finite set of possible algorithms at i^{th} step). Denoting $\Theta(a_i)$ the (possibly varying dimension) space of hyper-parameters associated with a_i, the eventual pipeline is described as $\mathbf{x} = (a_1, \theta_1), \ldots (a_\ell, \theta_\ell)$, with $\theta_i \in \Theta(a_i)$. A complete *pipeline structure* is an[5] ℓ-uple $\mathbf{a} = (a_1, \ldots a_\ell) \in \mathcal{A} = \mathcal{A}_1 \times \ldots \times \mathcal{A}_\ell$, with $\Theta(\mathbf{a}) = \Theta(a_1) \times \ldots \times \Theta(a_\ell)$ its associated hyper-parameter space. A k-pipeline structure (k-ps) is a k-tuple $\mathbf{s} = (a_1, \ldots a_k) \in \mathcal{A}_1 \times \ldots \times \mathcal{A}_k$, with $k \leq \ell$. Given a k-ps $\mathbf{s}$, any $\mathbf{x} \in X$ with same first k decisions as $\mathbf{s}$ is said to be compatible with $\mathbf{s}$ (noted $\mathbf{s} \preccurlyeq \mathbf{x}$) and the subset of pipelines compatible with $\mathbf{s}$ is noted $X(\mathbf{s}) = \{\mathbf{x} \in X; \mathbf{s} \preccurlyeq \mathbf{x}\}$.

A default distribution $\mathcal{D}$ is defined on X, involving a uniform distribution on all $\mathcal{A}_i$ and, conditionally to the selected a_i, a uniform distribution[6] on the (bounded) $\Theta(a_i)$. The default distribution on $X(\mathbf{s})$ is defined in the same way.

3.1 Two Intertwined Optimization Problems

The difficulty lies in simultaneously tackling the structural optimization of $\mathbf{a}$ in $\mathcal{A}$ and the parametric optimization of

[3] Moderate mistakes in the low-performing regions do not harm since these regions will not be much visited.

[4] Meta-features are used to describe datasets, using statistical, information theoretic and landmark-based measures [Muñoz *et al.*, 2018].

[5] Note that elements in $\mathcal{A}$ are not all admissible. Domain knowledge is used to early discard the non-admissible sequences $a_1 \ldots a_i$.

[6] Except for a few hyper-parameters such as the number of selected features in feature selection, for which the default distribution is biased toward small values.

the associated hyper-parameters $\theta(\mathbf{a})$ in $\Theta(\mathbf{a})$ where i) the optimization objective is non-separable[7]; ii) θ_j is of varying dimension, possibly depending on the value of some coordinates in θ_j (e.g. the number of neural layers controls the dimension of the neural layer size). At one extreme, one could optimize $\theta(\mathbf{a})$ for every considered $\mathbf{a}$ – an obviously intractable strategy. At the other extreme, one could estimate the performance of $\mathbf{a}$ from a few samples of $\theta(\mathbf{a})$.

MOSAIC achieves an intermediate strategy: A surrogate model $\widehat{\mathcal{F}}$ on X is maintained, generalizing all computed performances; During the optimization of the pipeline structure with MCTS, when considering incomplete structural pipeline $\mathbf{s} \in \mathcal{A}_1 \times \ldots \times \mathcal{A}_k$, a full pipeline $\mathbf{x}$ such that $\mathbf{s} \preccurlyeq \mathbf{x}$ is determined along the line of Bayesian optimization and the performance $\mathcal{F}(\mathbf{x})$ is computed. Thanks to MCTS backpropagation step, this allows to build a proxy for the performance of $\mathbf{s}$.

More formally, the novelty in MOSAIC is to tackle both structural and parametric optimization problems using two coupled strategies: MCTS is used to tackle the structural optimization of structure $\mathbf{a}$ and Bayesian optimization is used to tackle the parametric optimization of $\theta(\mathbf{a})$, where the coupling is ensured via the surrogate model(s). This hybrid strategy contrasts with that of AUTO-SKLEARN (resp. most other AutoML approaches), optimizing both $\mathbf{a}$ and $\theta(\mathbf{a})$ using Bayesian Optimization and a single surrogate model (resp. their own optimization methods). Note that in principle MCTS could be used to also achieve continuous optimization [Bubeck *et al.*, 2011]. However, the computational resource constraint on the AutoML problem, severely restricting the number of tree-walks, hinders a continuous MCTS optimization strategy.

3.2 Partial Surrogate Models

In MOSAIC as in AUTO-SKLEARN (section 2), a surrogate model $\widehat{\mathcal{F}}$ of the optimization objective is built from all computed performances $\mathcal{F}(\mathbf{x}_u = (\mathbf{a}_u, \theta(\mathbf{a}_u)))$.

A first step is to derive from $\widehat{\mathcal{F}}$ a surrogate model $Q_{\widehat{F}}$ on pipeline structures. For $k < \ell$, let $\mathbf{s}$ be a k-ps, and let $\mathbf{s}.a$ denote the $k+1$-ps built from $\mathbf{s}$ by selecting a as $k+1$-th decision. Then the surrogate $Q_{\widehat{F}}$ is defined as:

$$Q_{\widehat{F}}(\mathbf{s}, a) = \mathbb{E}_{\mathbf{x} \sim \mathcal{D}[X(s.a)]}\left(\widehat{\mathcal{F}}(\mathbf{x})\right) \approx \frac{1}{n_s}\sum_{j=1}^{n_s} \widehat{\mathcal{F}}(\mathbf{x}_j) \quad (3)$$

estimated from a number n_s ($n_s = 100$ in the experiments) of configurations sampled in $X(\mathbf{s}.a)$.

A probabilistic selection policy π can then be built from $Q_{\widehat{F}}$, with:

$$\pi(a|s) = \frac{\exp\left(Q_{\widehat{F}}(s,a)\right)}{\sum_{b \in \mathcal{A}_k} \exp\left(Q_{\widehat{F}}(s,b)\right)} \quad (4)$$

Taking inspiration from [Silver *et al.*, 2017], this policy is used to enhance the MCTS selection rule (below).

[7]That is, the marginal performance of a_j depends on all other $a_k, k \neq j$ and on $\theta(\mathbf{a})$. Likewise, the marginal performance of $\theta(a_j)$ depends on all a_k and $\theta(a_k)$ for $k \neq j$.

3.3 The MOSAIC Algorithm

MOSAIC (Alg. 1) follows the general MCTS scheme (section 2.2), where the main four phases have been modified as follows:

Down the MCTS Tree
In a non-leaf node $\mathbf{s}$ of the MCTS tree, with $\mathbf{s}$ a k-ps, the child node a is selected in $\mathcal{A}_k$ using the ALPHAGO ZERO criterion:

$$\underset{a}{\operatorname{argmax}}\left(\overline{Q}(\mathbf{s}, a) + C_{ucb} * \pi(a|\mathbf{s}) * \frac{\sqrt{n(\mathbf{s})}}{1 + n(\mathbf{s}.a)}\right) \quad (5)$$

where $\overline{Q}$ is the median[8] of $\mathcal{F}(\mathbf{x})$ for all $\mathbf{x}$ in $X(\mathbf{s}.a)$, $\pi(a|\mathbf{s})$ is defined by Eq.(4), $n(\mathbf{s})$ is the number of times s was visited, and C_{ucb} is the usual constant controlling the exploration *vs* exploitation trade-off.

Expansion
In a leaf node $\mathbf{s}$ of the MCTS tree, with $\mathbf{s}$ a k-ps, the child node a in $\mathcal{A}_k$ that maximizes the surrogate performance $Q_{\widehat{F}}(\mathbf{s}, a)$ is added to the MCTS tree.

Playout
Letting $\mathbf{s}$ be the (possibly complete) k-ps, a full pipeline $\mathbf{x}$ with $\mathbf{s} \preccurlyeq \mathbf{x}$ is defined using a sampling playout strategy. Three sampling strategies were considered: i) a configuration is sampled according to the default distribution $\mathcal{D}(X(\mathbf{s}))$; ii) a local search around the best recorded pipeline $(\mathbf{a}^*, \theta^*)$ in $X(\mathbf{s})$ is achieved and the best configuration according to $\widehat{\mathcal{F}}$ is retained; iii) a number of configurations is sampled after $\mathcal{D}(X(\mathbf{s}))$, together with a few configurations sampled via a local search around $(\mathbf{a}^*, \theta^*)$, and the sample $\mathbf{x}$ that maximizes the Expected Improvement of $\widehat{\mathcal{F}}$ is retained. In all cases, the true performance $\mathcal{F}(\mathbf{x})$ of the retained configuration is computed.

An empirical study (omitted for brevity) demonstrated that: the first sampling strategy is slow and prone to overfitting; the second strategy causes a loss of diversity of the considered pipelines, eventually resulting in a poor surrogate performance model $\widehat{\mathcal{F}}$. Hence only the third strategy is considered thereafter: the sampled configurations include n_r ($n_r = 1,000$ in the experiments) configurations sampled from default distribution $\mathcal{D}(X(\mathbf{s}))$, augmented with pipelines closest[9] to $(\mathbf{a}^*, \theta^*)$.

Back-propagation
Performance $\mathcal{F}(\mathbf{x})$ is back-propagated up the tree along the current path, updating the corresponding $\overline{Q}$ values. Example $(\mathbf{x}, \mathcal{F}(\mathbf{x}))$ is added to the surrogate training set, and the surrogate performance model $\widehat{\mathcal{F}}$ is retrained anew.

Stopping Criterion
The algorithm stops after the computational budget is exhausted (one hour per dataset in the experiments).

[8]The average was also considered, giving very similar results, except in rare cases of heavily failed runs.

[9]Formally, one selects every $(\mathbf{a}', \theta')$ such that either $\mathbf{a}' = \mathbf{a}^*$ and θ' differs from θ^* by a single hyper-parameter value; or $\mathbf{a}'$ differs from $\mathbf{a}^*$ by a single decision and θ' is the default hyper-parameter vector $\theta(\mathbf{a}')$.

Algorithm 1 MOSAIC Vanilla

procedure SELECTION(STATE $\mathbf{s}$)
  while state not terminal **do**
    $a \leftarrow$ Select action using Eq. 5
    return Selection($\mathbf{s}$.a)
  return $\mathbf{s}$
procedure EXPANSION(STATE $\mathbf{s}$)
  return $\text{argmax}_a \, Q_{\widehat{F}}(\mathbf{s}, a)$
procedure PLAYOUT(STATE)
  $P \leftarrow \mathcal{D}[X(state)] \cup Neighbor(x_l^*)$ // best configuration $x_i^* \in X(state)$
  return $\text{argmax}_{c \in P} \, EI(c)$ // Expected improvement
procedure MOSAIC(T, d)
  while $t < T$ **do**
    $s \leftarrow \emptyset$
    $s \leftarrow$ Selection(s)
    $a \leftarrow$ Expansion(s)
    x $\leftarrow$ Playout($s \cup \{a\}$)
    Observe performance r of x on d
    for $p \in ancestors(s)$ **do**
      Update Q at state p with r
      $n(p) \leftarrow n(p) + 1$

3.4 Initialization and Variants

The order of the decisions in the structural pipeline is key to the optimization: while MCTS yields asymptotic optimality guarantees, the discovery of good decisions can be delayed due to poorly informative or unlucky starts [Coquelin and Munos, 2007]. Accordingly, the order of decisions in the structural pipeline is fixed, and the first decision made in the root node of the tree is the choice of the learning algorithm. Note that each learning algorithm has an associated default complete pipeline.

MOSAIC**.Vanilla.** The initialization proceeds as follows: For each learning algorithm ($\mathbf{s} = (a)$ with $a \in \mathcal{A}_1$), its default complete pipeline is launched, together with κ (= 3 in the experiments) other pipelines sampled from $X(\mathbf{s})$, and their associated performances are computed. The initial surrogate model $\widehat{\mathcal{F}}$ is trained from the set of all such $(\mathbf{x}, \mathcal{F}(\mathbf{x}))$ and $Q_{\widehat{F}}(\emptyset, a)$ is initialized for a in $\mathcal{A}_1$.

MOSAIC**.MetaLearning.** It borrows AUTO-SKLEARN its better informed initialization, where the first 25 configurations are the best recorded ones for each of the nearest neighbors of the current dataset, in the sense of the meta-feature distance (section 2.4). The next configurations are selected as in MOSAIC.Vanilla, and the actual search starts thereafter.

MOSAIC**.Ensemble.** It is similar to MOSAIC.Vanilla, but returns the compound model defined as a weighted sum of the models computed along the AutoML search, using an online ensemble building strategy [Caruana *et al.*, 2004].

4 Experimental Setting

4.1 Goals of Experiment

The empirical validation of MOSAIC firstly aims to assess its performance compared to AUTO-SKLEARN [Feurer *et al.*, 2015], that consistently dominated other systems in the international AutoML challenges [Guyon *et al.*, 2015]. The other AutoML system used as baseline is the evolutionary optimization-based[10] TPOT (v0.9.5) [Olson *et al.*, 2016].

The second goal of experiments is to better understand the specifics of the AutoML optimization problem. A first issue regards the exploration *vs* exploitation trade-off on the structural *vs* parametric subspaces and the merits of using MCTS as opposed to Bayesian optimization on the structural space. A second issue regards the impact of the MetaLearning initialization. MCTS is notorious to achieve a consistent though moderate exploration, which as said might slow down the search due to unlucky early choices. The smart initialization tends to prevent such hazards. On the other hand, if the initialization is *very* effective, the more conservative AUTO-SKLEARN exploration strategy might be more appropriate.

The exploration strategies of MOSAIC and AUTO-SKLEARN are compared, and the diversity of the visited configurations is examined in an extended version[11].

4.2 Experimental Setting

Search Space

A fair comparison is ensured by assessing AUTO-SKLEARN and MOSAIC on the same *scikit-learn* portfolio [Pedregosa *et al.*, 2011]. The search space involves 16 ML algorithms, 13 pre-processing methods, 2 categorical encoding strategies, 4 missing values imputation strategies, 6 rescaling strategies and 2 balancing strategies. The size of the structural search subspace is 6,048 (due to parameter dependencies). The overall parametric search space has dimensionality 147 (93 categorical scalar hyper-parameters, 32 integer, 47 continuous). Each hyper-parameter ranges in a bounded discrete or continuous domain. For each configuration $\mathbf{x} = (\mathbf{a}, \theta(\mathbf{a}))$, $\theta(\mathbf{a})$ involves a dozen scalar hyper-parameters on average.

MOSAIC involves 2 hyper-hyper-parameters additionally to those of AUTO-SKLEARN: the number $n_s = 100$ of samples to compute $Q_{\widehat{F}}$ (Eq. 3), $C_{ucb} = 1.3$ controlling the exploration *vs* exploitation (Eq. (5)) and the coefficient of progressive widening $PW = 0.6$. Shared hyper-hyper-parameters include: number n_r of uniformly sampled configurations and variance $\epsilon = .2$ for the local search in the Playout phase (Section 3.3).

Benchmark Suite

The compared AutoML systems are assessed on the OpenML repository [Vanschoren *et al.*, 2013], including 100 classification problems. The overall computational budget is set to 1 hour for each dataset. Computational times are measured on an AMD Athlon 64 X2, 5GB RAM. For all systems, every considered $\mathbf{x}$ configuration is launched to learn a model from 70% of the training set with a cut-off time of 300 seconds, and performance $\mathcal{F}(\mathbf{x})$ is set to the model accuracy on the remaining 30%. The best configuration $\mathbf{x}^*$ is launched to learn a model on the whole training set and its performance on the

[10] ALPHAD3M [Drori *et al.*, 2018] and AUTOSTACKER [Chen *el al.*, 2018] could not be considered due to lack of information.

[11] See https://arxiv.org/abs/1906.00170

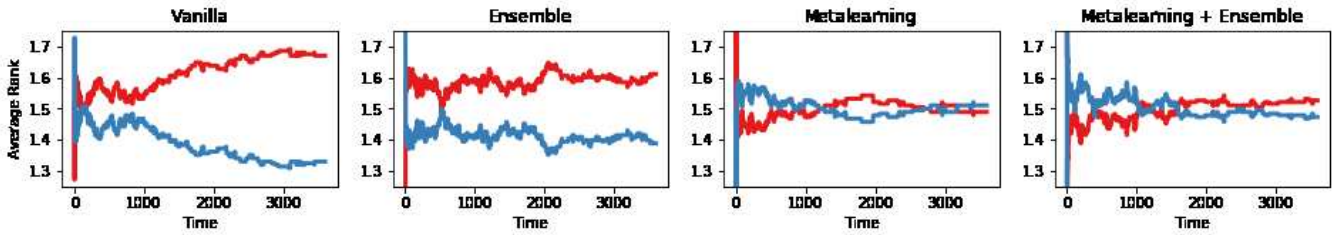

Figure 2: Comparative assessment of MOSAIC and AUTO-SKLEARN: Average performance rank (the lower the better) on OpenML-100 *vs* CPU time of the Vanilla, Ensemble, MetaLearning and Ensemble+MetaLearning variants (left to right). Better seen in color.

(unseen) test set is reported. Finally, this performance is averaged over 10 independent runs, and the average is reported as the system performance on this dataset. For the MetaLearning variant, the considered archive includes all datasets but not the one under examination.

For each dataset, the performances achieved by all systems are ranked. The overall performance of a system is its average rank over all (the lower the better). As the rank indicator might be blurred when many systems and their variants are considered together, duels between pairs of systems (MOSAIC.X against AUTO-SKLEARN.X, where X ranges in *Vanilla, Meta-Learning, Ensemble, Meta-Learning+Ensemble*, section 3.4), are considered.

5 Empirical Validation

Vanilla Variants

The comparative performances of Vanilla AUTO-SKLEARN, TPOT and MOSAIC *vs* computational time are displayed on Figs. 2-a and 3, showing that the hybrid optimization used in MOSAIC clearly improves on the Bayesian optimization-only used in AUTO-SKLEARN (and on the evolutionary optimization-only used in TPOT) from the early stages until the end.

The actual performances of the configurations respectively selected by AUTO-SKLEARN and MOSAIC are reported on Fig. 4. According to a Mann-Whitney-Wilcoxon test with 95% confidence, MOSAIC significantly outperforms AUTO-SKLEARN on 21 datasets out of 100; AUTO-SKLEARN outperforms MOSAIC on 6 datasets out of 100. Additionally, MOSAIC improves on AUTO-SKLEARN on 35 other datasets (though not in a statistically significant way), and the reverse is true on 18 datasets. Both are equal on 18 datasets and both systems crashed on 2 datasets.

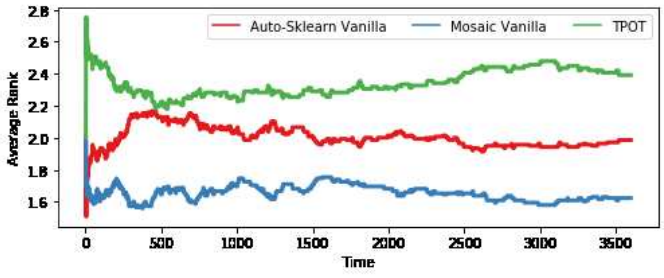

Figure 3: Average performance ranks (lower is better) on OpenML-100 *vs* CPU time of the Vanilla versions of MOSAIC (bottom), AUTO-SKLEARN (middle), and TPOT (top). Better seen in color.

MetaLearning and Ensemble Variants

The impacts of the MetaLearning and Ensemble variants are displayed on Fig. 2. While MOSAIC dominates AUTO-SKLEARN as long as the Vanilla variants are considered (Fig. 2-a), the difference decreases for the Ensemble variant (Fig. 2-b) and it becomes non-statistically significant for the MetaLearning variant (Fig. 2-c), as well as for the MetaLearning + Ensemble variant (Fig. 2-d).

A closer inspection of the results reveals that the best AUTO-SKLEARN configuration is almost always found during the initialization and AUTO-SKLEARN.MetaLearning thereafter mostly explores the close neighborhood of the initial configurations. In the meanwhile, MOSAIC follows a more thorough exploration strategy; this exploration might entail a bigger risk of overfitting, discovering configurations with better performance on the validation set, at the expense of the performance on the test set.

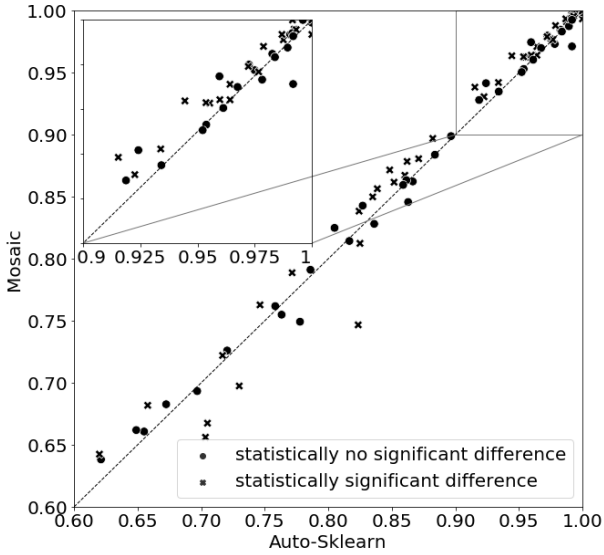

Figure 4: Performance of MOSAIC (y-axis) versus AUTO-SKLEARN (x-axis) on OpenML-100. Datasets for which the difference is statistically significant (resp. insignificant) after MWW test with confidence 5% are represented with a $\times$ (resp $\bullet$).

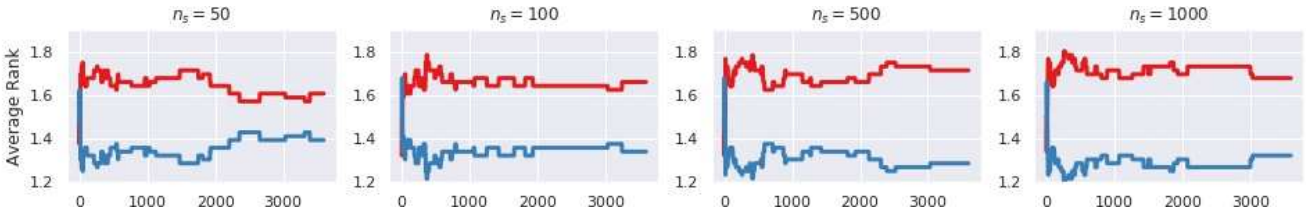

Figure 5: Sensitivity study w.r.t. hyper-parameters n_s for $C_{ucb} = 1.3$ and $PW = 0.6$: Average rank of MOSAIC.Vanilla against AUTO-SKLEARN.Vanilla. Better seen in color.

Sensitivity w.r.t. MOSAIC hyper-parameters
Complementary sensitivity studies have been conducted to assess the impact of MOSAIC hyper-parameters. For computational reasons, only 30 datasets out of 100 have been considered, and MOSAIC.Vanilla is run 5 times with a 1 hour budget on each dataset.

Fig. 5 displays the average rank vs time of MOSAIC.Vanilla for different values of n_s (50, 100, 500, 1000), showing the low sensitivity of the performance w.r.t. n_s in this range for $C_{ucb} = 1.3$ and $PW = .6$.

Fig. 6 displays the average rank of MOSAIC.Vanilla at the end of the learning curve, for C_{ucb} in $\{.1, .3, .6, 1, 1.3, 1.6\}$ and PW in $\{1, .8, .7, .6, .5\}$, showing that MOSAIC dominates Auto-Sklearn for 24 settings out of 30.

Exploration of the Search Space
The differences in the exploration strategies of AUTO-SKLEARN and MOSAIC become more visible at a later stage of the search: MOSAIC switches to the exploitation of the most promising MCTS subtrees (subspaces of the search space) and avoids regions where the last visited configurations were bad; on the other hand, AUTO-SKLEARN continues to explore even if the sub-space includes quite a few bad configurations.

6 Discussion and Perspectives

The main contribution of the paper is the new MOSAIC scheme, tackling the AutoML optimization problem through handling both the structural and the parametric optimization problems. The proposed approach is based on a novel coupling between Bayesian Optimization and MCTS strategies, that are tied by sharing the same surrogate model. In MCTS the surrogate model is used to estimate, in all nodes, the average performance of all subtrees (ends of pipeline) below this node, and thus to choose the next node. The same surrogate model is used during the roll-outs, to choose the optimal hyper-parameters of the pipeline using a Bayesian Optimization strategy.

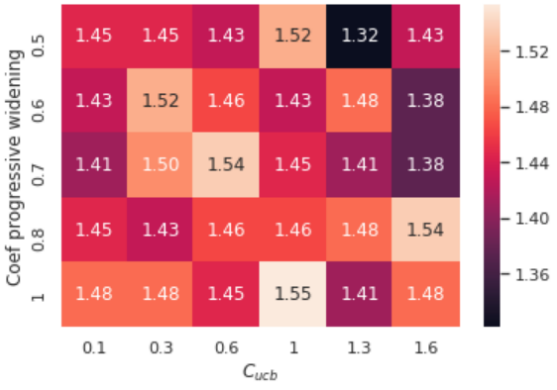

Figure 6: Sensitivity study w.r.t. hyper-parameters C_{ucb} and PW, for $n_r = 100$: Average rank of MOSAIC.Vanilla against AUTO-SKLEARN.Vanilla (the lower, the better).

Empirically, the results demonstrate that MOSAIC significantly outperforms the challenge winner AUTO-SKLEARN on the OpenML benchmark suite, at least as long as the Vanilla and Ensemble variants are considered. With the MetaLearning variant however, the difference becomes insignificant as the bulk of optimization is devoted to the initialization (all the more so for large datasets, due to the one hour cut-off time).

The limitation of such a smart initialization is twofold. On the one hand, it relies on preliminary expensive computations to build the archive (one day computation *per* dataset on OpenML-100); on the other hand, it assumes the representativity of the problems in the archive. On-going work is concerned with estimating the risk of overfitting the OpenML benchmark, through measuring the sensitivity of the AUTO-SKLEARN and MOSAIC MetaLearning variants when varying the fraction of the datasets in the archive.

In any case, the experimental evidence suggests that Vanilla MOSAIC offers a robust and efficient AutoML facility when tackling a new application domain, and/or in the absence of a comprehensive archive.

A long term research perspective is to reconsider the design of the meta-features [Muñoz *et al.*, 2018]. In principle, a binary classification problem can be associated to any ML algorithm, where a dataset belongs to class $+$ if the algorithm performs comparatively well on this dataset, and class $-$ otherwise. The perspective is to apply equivariant learning [Cohen and Welling, 2016] at the dataset level to tackle this binary classification problem, and use the resulting equivariant classifier as (cheap) meta-feature.

Acknowledgments

This work was funded by the ADEME #1782C0034 project *NEXT*.

References

[Auer, 2002] P. Auer. Using Confidence Bounds for Exploitation-Exploration Trade-offs. *Journal of Machine Learning Research*, 3:397–422, 2002.

[Auger *et al.*, 2013] David Auger, Adrien Couetoux, and Olivier Teytaud. Continuous upper confidence trees with polynomial exploration–consistency. In *ECML-KDD*, pages 194–209. Springer, 2013.

[Balaji and Allen, 2018] A. Balaji and A. Allen. Benchmarking Automatic Machine Learning Frameworks. *arXiv:1808.06492 [cs, stat]*, 2018.

[Bergstra *et al.*, 2011] J. S. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl. Algorithms for Hyper-Parameter Optimization. In *NIPS*, pages 2546–2554, 2011.

[Brazdil and Giraud-Carrier, 2018] P. Brazdil and Ch. Giraud-Carrier. Metalearning and Algorithm Selection: Progress, State of the Art and Introduction to the Special Issue. *Machine Learning*, 107(1):1–14, 2018.

[Bubeck *et al.*, 2011] S. Bubeck, R. Munos, G. Stoltz, and C. Szepesvári. X-armed bandits. *Journal of Machine Learning Research*, 12:1655–1695, 2011.

[Caruana *et al.*, 2004] R. Caruana, A. Niculescu-Mizil, G. Crew, and A. Ksikes. Ensemble selection from libraries of models. In *Proc.ICML*, page 18, 2004.

[Chaslot *et al.*, 2008] G. Chaslot, S. Bakkes, I. Szita, and P. Spronck. Monte-Carlo Tree Search: A New Framework for Game AI. In *AI and Interactive Digital Entertainment*, pages 216–217. AAAI Press, 2008.

[Chen *et al.*, 2018] B. Chen, H. Wu, W. Mo, I. Chattopadhyay, and H. Lipson. Autostacker: A Compositional Evolutionary Learning System. In *Proc. ACM-GECCO*, pages 402–409. ACM Press, 2018.

[Cohen and Welling, 2016] T. Cohen and M. Welling. Group Equivariant Convolutional Networks. In *Proc. ICML*, volume 48, pages 2990–2999. PMLR, 2016.

[Coquelin and Munos, 2007] P.-A. Coquelin and R. Munos. Bandit algorithms for tree search. In *Proc. UAI 2007*, pages 67–74, 2007.

[Drori *et al.*, 2018] I. Drori, Y. Krishnamurthy, et al. AlphaD3M: Machine Learning Pipeline Synthesis . In *ICML workshop on AutoML*, 2018.

[Eggensperger *et al.*, 2013] K. Eggensperger, M. Feurer, F. Hutter, et al. Towards an Empirical Foundation for Assessing Bayesian Optimization of Hyperparameters. In *NIPS wkp on BO in Theory and Practice*, 2013.

[Feurer *et al.*, 2015] M. Feurer, Klein A., Eggensperger K., M. Blum, and F. Hutter. Efficient and Robust Automated Machine Learning. In C. Cortes et al., editor, *NIPS 28*, pages 2962–2970, 2015.

[Gelly and Silver, 2011] S. Gelly and D. Silver. Monte-Carlo Tree Search and Rapid Action Value Estimation in Computer GO. *Artificial Intelligence*, 175:1856–1875, 2011.

[Guyon *et al.*, 2015] I. Guyon, K. Bennett, G. Cawley, et al. Design of the 2015 Chalearn AutoML challenge. In *Proc. IJCNN*, pages 1–8. IEEE, 2015.

[Guyon *et al.*, 2018] I. Guyon, W.-W. Tu, et al. Automatic Machine Learning Challenge 2018: Towards AI for Everyone. In *PAKDD 2018 Data Competition*, 2018.

[Hutter *et al.*, 2011] F. Hutter, H.H. Hoos, and K. Leyton-Brown. Sequential Model-based Optimization for General Algorithm Configuration. In *Proc. LION*, pages 507–523. Springer Verlag, 2011.

[Klein *et al.*, 2017] A. Klein, S. Falkner, et al. Fast Bayesian Optimization of Machine Learning Hyperparameters on Large Datasets. In *Proc. AISTAT*, pages 528–536, 2017.

[Kocsis and Szepesvári, 2006] L. Kocsis and C. Szepesvári. Bandit Based Monte-Carlo Planning. In Fürnkranz et al., editor, *Proc. ECML*, pages 282–293. Springer, 2006.

[Kotthoff *et al.*, 2017] L. Kotthoff, Ch. Thornton, H.H. Hoos, F. Hutter, and K. Leyton-Brown. Auto-WEKA 2.0: Automatic Model Selection and Hyperparameter Optimization in WEKA. *JMLR*, 18(25):1–5, 2017.

[Li *et al.*, 2017] L. Li, K. Jamieson, G. DeSalvo, et al. Hyperband: A novel bandit-based approach to hyperparameter optimization. *JMLR*, 18(1):6765–6816, 2017.

[Mockus *et al.*, 1978] J. Mockus, V. Tiesis, and A. Zilinskas. The application of Bayesian methods for seeking the extremum. In L. Dixon and G. Szego, editors, *Toward Global Optimization*, volume 2. Elsevier, 1978.

[Muñoz *et al.*, 2018] M. A. Muñoz, L. Villanova, D. Baatar, and K. Smith-Miles. Instance spaces for machine learning classification. *Machine Learning*, 107(1):109–147, 2018.

[Olson *et al.*, 2016] R. S. Olson, N. Bartley, R. J. Urbanowicz, and J. H. Moore. Evaluation of a Tree-based Pipeline Optimization Tool for Automating Data Science. In *Proc. ACM-GECCO*, pages 485–492. ACM Press, 2016.

[Pedregosa *et al.*, 2011] F. Pedregosa, G. Varoquaux, A. Gramfort, et al. Scikit-learn: Machine learning in Python. *JMLR*, 12:2825–2830, 2011.

[Silver *et al.*, 2017] D. Silver, J. Schrittwieser, et al. Mastering the Game of GO without Human Knowledge. *Nature*, 550(7676):354–359, 2017.

[Snoek *et al.*, 2015] J. Snoek, O. Rippel, K. Swersky, et al. Scalable Bayesian Optimization using Deep Neural Networks. In *Proc. ICML*, pages 2171–2180, 2015.

[Swersky *et al.*, 2014] K. Swersky, J. Snoek, and R. P. Adams. Freeze-thaw Bayesian optimization. *preprint arXiv:1406.3896*, 2014.

[Vanschoren *et al.*, 2013] J. Vanschoren, J.N. van Rijn, B. Bischl, and L. Torgo. OpenML: Networked Science in Machine Learning. *SIGKDD Expl.*, 15(2):49–60, 2013.

[Wang, 2016] Z. Wang. *Practical and Theoretical Advances in Bayesian Optimization*. PhD thesis, Univ. Oxford, 2016.

[Wistuba, 2018] M. Wistuba. Deep Learning Architecture Search by Neuro-Cell-Based Evolution with Function-Preserving Mutations. In Berlingerio, M. et al., editor, *Proc. ECML-PKDD*, pages 243–258. Springer, 2018.

[Wolpert, 1996] D. H. Wolpert. The Lack of A Priori Distinctions Between Learning Algorithms. *Neural Computation*, 8(7):1341–1390, 1996.