Hybridizing TPOT with Bayesian Optimization

Angus Kenny, Tapabrata Ray, Steffen Limmer, Hemant Singh, Tobias Rodemann, Markus Olhofer

2023

Preprint:

Hybridizing TPOT with Bayesian Optimization

Angus Kenny
The University of New South Wales
Canberra, Australia
angus.kenny@adfa.edu.au

Tapabrata Ray
The University of New South Wales
Canberra, Australia
t.ray@adfa.edu.au

Steffen Limmer
Honda Research Institute Europe
Offenbach, Germany
steffen.limmer@honda-ri.de

Hemant Kumar Singh
The University of New South Wales
Canberra, Australia
h.singh@adfa.edu.au

Tobias Rodemann
Honda Research Institute Europe
Offenbach, Germany
tobias.rodemann@honda-ri.de

Markus Olhofer
Honda Research Institute Europe
Offenbach, Germany
markus.olhofer@honda-ri.de

ABSTRACT

Tree-based pipeline optimization tool (TPOT) is used to automatically construct and optimize machine learning pipelines for classification or regression tasks. The pipelines are represented as trees comprising multiple data transformation and machine learning operators — each using discrete hyper-parameter spaces — and optimized with genetic programming. During the evolution process, TPOT evaluates numerous pipelines which can be challenging when computing budget is limited. In this study, we integrate TPOT with Bayesian Optimization (BO) to extend its ability to search across continuous hyper-parameter spaces, and attempt to improve its performance when there is a limited computational budget. Multiple hybrid variants are proposed and systematically evaluated, including (a) sequential/periodic use of BO and (b) use of discrete/continuous search spaces for BO. The performance of these variants is assessed using 6 data sets with up to 20 features and 20,000 samples. Furthermore, an adaptive variant was designed where the choice of whether to apply TPOT or BO is made automatically in each generation. While the variants did not produce results that are significantly better than "standard" TPOT, the study uncovered important insights into the behavior and limitations of TPOT itself which is valuable in designing improved variants.

CCS CONCEPTS

• **Computing methodologies** → **Search methodologies**; **Randomized search**;

ACM Reference Format:
Angus Kenny, Tapabrata Ray, Steffen Limmer, Hemant Kumar Singh, Tobias Rodemann, and Markus Olhofer. 2023. Hybridizing TPOT with Bayesian Optimization. In *Genetic and Evolutionary Computation Conference (GECCO '23), July 15–19, 2023, Lisbon, Portugal.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3583131.3590364

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

ACM ISBN 979-8-4007-0119-1/23/07…$15.00
https://doi.org/10.1145/3583131.3590364

1 INTRODUCTION

Researchers and practitioners have a significant interest in developing numerical models that have good predictive capability to deal with classification or regression tasks. Given that the choice of model(s) and the ways they can be combined to accomplish these tasks is virtually unlimited, the choice is best dealt using automated machine learning tools (AutoML) [8] in lieu of manual selection. Machine learning models can be employed individually, or sequentially; using the output of one model as the input to the next, harnessing the strengths of multiple models simultaneously. When combined in this manner, the models are collectively known as a machine learning *pipeline* [13]. Each model has its own unique set of hyper-parameters which must be prescribed before application of the model. The hyper-parameters control various aspects of the model behavior and can take continuous, discrete, binary or even categorical values. In designing a machine learning pipeline, one needs to decide on the following:

(1) Which models to use in the pipeline?
(2) How are they organised relative to each other?
(3) What are the values of their respective hyper-parameters?

Tree-based pipeline optimization tool (TPOT) [13] is a Python library, built on top of the Distributed Evolutionary Algorithms (DEAP) [5](https://github.com/deap) library, designed to automate this machine learning pipeline development process for classification or regression tasks[1]. It represents pipelines as tree-based data structures, constructed using the genetic programming (GP) methods provided by DEAP.

One significant limitation of TPOT is that, for all its sophistication in utilizing GP, it still relies on a random-search method for tuning hyper-parameters using discretized hyper-parameter spaces. When a new model is added to a pipeline during the initialization or mutation, its hyper-parameters are uniformly randomly sampled from their respective value ranges. While many hyper-parameters are essentially continuous variables, TPOT discretizes the continuous search spaces with an arbitrary granularity. Although this approach is somewhat effective, the fact still remains that unless the global optimum value for a given hyper-parameter lies on the exact point of discretization, TPOT will not find it. Another issue, is related to the use of TPOT in settings where the computational budget is limited, i.e., there is a limit on the number of pipelines that can be evaluated during the course of search. Since pipelines

[1]The methods proposed in this paper were developed using the TPOTRegressor class, however they are equally applicable to the TPOTClassifier class as well.

in TPOT are evolved using an evolutionary algorithm (GP), many evaluations are required in order to achieve convergence, which is critical if the ML task involves long training times.

Optuna [1](https://optuna.org/) is a hyper-parameter optimisation library for Python which employs Bayesian optimisation (BO) [14] to tune hyper-parameter values. It uses a Tree-based Parzen Estimator (TPE) [2, 4] as the underlying surrogate model, built using historical information to estimate and suggest promising values. Unlike TPOT, Optuna has no limitations on the type of values the hyper-parameters can take, however it is not as effective at constructing/selecting models as TPOT.

Based on the above, TPOT can be thought of as an effective tool for *exploration* of the space of possible pipelines, and Optuna as an effective means for *exploitation* of those pipelines, by fine-tuning their hyper-parameters — harnessing the strengths of both. In this study, we follow this line of inquiry to construct three hybrid variants of TPOT and BO — along with a "standard" TPOT base-line — and evaluate them on a range of datasets to draw insights that will be valuable for designing improved optimization approaches.

2 BACKGROUND AND RELATED WORK

There has been significant growth and interest in industrial applications of machine learning in recent years. Identification of a model to deal with a regression or a classification task involves repetitive and computationally demanding evaluation of various models and their combinations with potentially numerous hyper-parameter settings. Automated machine learning (AutoML) has emerged as an effective approach to deal with the above problem where optimal pipelines consisting of data pre-processing, feature engineering, model selection with tuned hyper-parameters are sought. A review of the history of AutoML tool development along with a comprehensive analysis of existing tools and their functionalities appears in [16]. Some prominent open source developments include Auto-Weka [11], TPOT [13], auto-sklearn [6], and H2O [15]. The developments have typically focused on two specific areas: how to represent and combine multiple models in a pipeline and how to find the best hyper-parameter settings for the underlying models. TPOT adopts a tree-based representation for its pipelines and modifies the trees through GP. This scheme in TPOT can potentially lead to the generation of infeasible, or duplicate, pipelines which are not evaluated. As for the hyper-parameters of the models, TPOT relies on the use of random search within a discretized parameter space. This is motivated by the observation reported in [3] where it was shown that such a scheme can discover high-performing parameter sets faster than a more comprehensive search.

Hyper-parameter optimization methods evaluate promising configurations such as via successive halving [10] or via BO. For example, Gaussian process models constructed using historical data have long been employed in BO [9]. Alternatively, a tree structured Parzen estimator (TPE) [2, 4] can be used where the models are represented as trees involving a large number of continuous, discrete, binary and categorical hyper-parameters. A recent review of BO methods can be found in [17]. Optuna [1] is an open source hyper-parameter optimization framework that supports BO via TPE. While the original implementation used an independent TPE sampler, which ignores the interaction among the hyper-parameters, a multivariate TPE sampler has recently been added which has significantly improved the efficiency of the optimization process.

Building upon the above developments, we present here a framework that is aimed at combining the strengths of TPOT and BO. Within the framework, multiple variants have been constructed, including those that use BO sequentially or periodically and those that use discrete or continuous search spaces for BO. The framework is outlined in Section 3, followed by numerical experiments and detailed analysis in Sections 4 and 5. Based on the observations, another variant that includes the automatic selection of the TPOT/BO step has also been studied and presented in Section 5. The summary of the findings along with additional challenges and potential future directions are discussed in Section 6.

3 BO-TPOT SUITE

The proposed BO-TPOT[2] is a suite of methods which aim to augment the current capabilities of TPOT using BO. Three variants of BO-TPOT are presented below: TPOT-BASE, TPOT-BO-S and TPOT-BO-ALT. Briefly, TPOT-BASE executes baseline TPOT for the entire computational budget of pipeline evaluations (i.e., cross-validation on the input data set), TPOT-BO-S executes TPOT until a certain point and then switches to BO for the remainder of the pipeline evaluations, while TPOT-BO-ALT alternates between TPOT and BO steps a few times over the run for a pre-defined number of pipeline evaluations each. More details are given below.

3.1 TPOT-BASE

To evaluate the performance of the variants, the baseline performance of TPOT is first established. TPOT-BASE loads the training data D for the given problem and using the TPOT Python library, a given population of nP pipelines is constructed and evolved using a set of genetic programming (GP) operators ρ over nG_t generations. The result is a set S of $nG_t \times nP$ evaluated pipelines, minus any which were not evaluated due to time-out, or other issues. Algorithm 1 provides the pseudo-code for this baseline procedure. The output is given as a Python dictionary object, the keys for which are the string representations of the pipelines in S. Hyper-parameter values can be drawn from (discretized) continuous spaces, integers, categorical or boolean values.

Algorithm 1: TPOT-BASE

Input: D: training data; nP: population size; nG_t: total generations; ρ: GP parameters
Output: S: set of evaluated pipelines
1: $T \leftarrow$ TPOTRegressor object created using nP and ρ
2: $S \leftarrow$ result of fitting T on D for nG_t generations
3: **return** S

3.2 TPOT-BO-S: Single instance BO step

At its core, TPOT employs random search for hyper-parameters based on a discrete search space. The use of discretized search spaces for all the hyper-parameters, and random search to identify them, could turn out inefficient choices — especially when the

[2]Code available at https://github.com/AngusKenny/BO-TPOT.

computing budget is limited. Furthermore, since TPOT evaluates multiple pipelines in every generation, the computational budget can easily be exhausted within a few generations.

Bayesian optimization methods are typically used when the computational budget in terms of candidate evaluations is limited. BO methods build a surrogate model to approximate the fitness function and maximize an acquisition function, typically based on predicted mean and associated uncertainties, to determine the next sampling location. This means that combining BO with TPOT will allow TPOT-BO-S to be more selective in terms of sampling the hyper-parameter values, compared to TPOT on its own.

Some AutoML tools (e.g., auto-sklearn) employ Bayesian optimization (BO) for model selection. However, these tools typically deal with fixed pipeline structures — not with flexible ones like TPOT does — meaning BO is more frequently used in hyper-parameter optimization. Therefore, TPOT-BO-S is applied to fine-tune the hyper-parameters of the best pipeline identified by TPOT at a prescribed point, e.g., after some number (nG_s) of generations.

The pseudo-code for TPOT-BO-S is given in Algorithm 2. It uses S, typically the output generated by TPOT-BASE at a prescribed stopping point nG_s, to determine s, the pipeline with the smallest cross-validation (CV) error, which is chosen as the candidate for improvement by BO. All pipelines with a matching structure to this candidate pipeline are extracted from S to make a new set of pipelines $\bar{S}$ (Line 1). Two unique pipelines are said to have matching structures if they contain the same operators, organised in the same way, but with different hyper-parameter values.

The hyper-parameter values for the pipelines in $\bar{S}$ are used to initialise a model M (Line 2) and BO is allowed to evaluate a total of $nE = (nG_t - nG_s) \times nP$ pipelines, where nG_t is the total number of TPOT generations and nP is the population size. This ensures that the total computing budget consumed by TPOT-BO-S and the initial 80 TPOT-BASE generations is equivalent to that consumed by the full TPOT-BASE execution[3]. Pipelines with hyper-parameters suggested by BO are successively evaluated (Line 7), and M updated (Line 13), until the computational budget has been exhausted, at which point the total set of evaluated pipelines is returned.

Occasionally, there do not exist sufficient unique combinations of hyper-parameter values to satisfy the entire budget, which can result in TPOT-BO-S falling into an infinite loop. To avoid this, a counter is maintained which triggers a second stopping condition if no *new* pipelines are evaluated within 100 consecutive attempts (Line 9). If this condition is reached, the BO step is prematurely terminated, returning the best pipeline so far.

Unlike TPOT operating with discretized hyper-parameter values, BO is not limited to discrete spaces and can undertake a much finer-grained hyper-parameter search. Although many of the models supported by TPOT have continuous hyper-parameters, TPOT discretizes them in its standard implementation. Hence, on its own, TPOT might not be able to find the optimal value for a given hyper-parameter unless it happens to exist in the discretized set of values. In contrast, the surrogate model and acquisition function of BO based on TPE can operate seamlessly within discrete, continuous and categorical hyper-parameter search spaces.

[3]Noting that it is possible for TPOT-BASE to evaluate marginally fewer than $nG_t \times nP$ pipelines, since some might time-out or already exist in the set of evaluated pipelines.

Algorithm 2: TPOT-BO-S

Input: D: training data; S: input pipelines; nE: BO evaluations; ρ: GP parameters
Output: S: updated set of evaluated pipelines

1: $\bar{S} \leftarrow$ best pipeline s in S and all matching pipelines
2: $M \leftarrow$ surrogate model constructed from $\bar{S}$
3: $T \leftarrow$ TPOTRegressor with population size 1 and GP parameters ρ
4: $c \leftarrow 0$: counter for unsuccessful evaluations
5: $nE_t \leftarrow nE + |\bar{S}|$
6: **while** $|\bar{S}| < nE_t$ **do**
7: $\quad s' \leftarrow$ pipeline built from hyper-parameters suggested by M, evaluated by T on D and $\bar{S}$
8: $\quad$ **if** s' not evaluated **then**
9: $\quad\quad c \leftarrow c + 1$, **break** if $c = 100$
10: $\quad$ **else**
11: $\quad\quad c \leftarrow 0$, $\bar{S} \leftarrow \bar{S} \cup s'$
12: $\quad$ **end if**
13: $\quad M \leftarrow$ update surrogate model with s'
14: **end while**
15: **return** $S \cup \bar{S}$

3.3 TPOT-BO-ALT: Alternating TPOT and BO steps

An alternative strategy to TPOT-BO-S, but with similar intent, is referred here as TPOT-BO-ALT. TPOT-BO-S invests a large portion of its computational budget to improve a single candidate pipeline in its BO step. However, this is only effective when the pipeline selected has enough room for improvement in the first place. As there is no way of knowing this *a priori*, TPOT-BO-S essentially locks-in its choice of pipeline structure and hopes that further improvements materialize during the BO step.

TPOT-BO-ALT, presented in Algorithm 3, incorporates the ideas developed in TPOT-BO-S, but addresses the issue raised by its potentially high-risk strategy of investing the computational budget in a single pipeline. It does so by dividing the total computational budget into a series of alternating TPOT (Line 5) and TPOT-BO-S (Line 6) steps, incrementally evolving the population, and improving the best candidate, as it goes. Fewer BO evaluations are thus invested in any single pipeline, but the risk that a large portion of the computational budget will be wasted at the end is significantly reduced.

The total computing budget is divided into nI iterations, with each further divided into a TPOT and a TPOT-BO-S step. The total number of TPOT generations nG_t, and the number of generations per TPOT step nG_s are specified as part of its input. To ensure a fair comparison, nG_t is set the same as for TPOT-BASE and nG_s is set the same as for TPOT-BASE, divided by nI. This means the total sum of TPOT and BO evaluations is the same as the initial (full) TPOT-BASE execution. Minor variations may result due to any skipped evaluations during the TPOT step, or a prematurely terminated BO step.

4 NUMERICAL EXPERIMENTS

This section discusses the test problems and experimental settings used to benchmark the BO-TPOT variants presented so far.

Algorithm 3: TPOT-BO-ALT

Input: D: training data; nG_t: total generations; nP: population size; nI: iterations; nG_s: generations in TPOT step; ρ: GP parameters
Output: S: set of evaluated pipelines

$T \leftarrow$ TPOTRegressor initialized using nP and ρ
$nE \leftarrow (nG_t - nI \times nG_s) \times \frac{nP}{nI}$, number of TPOT-BO-S evaluations
$S \leftarrow \emptyset$, evaluated pipeline set
for nI iterations **do**
  $S \leftarrow$ update with result of T on D and S for nG_s generations†
  $S \leftarrow$ TPOT-BO-S(D, S, nE, ρ)
end for
return S

†initially, T is fit for $nG_s - 1$ generations, to account for the starting population.

4.1 Test problems

The experiments were conducted using six well-studied data sets drawn from the literature [7]. These data sets were chosen to have a good representation of real-world regression problems with a diverse number of features. They are presented in Table 1. Categorical features in the data are encoded as integer values.

Table 1: Six data sets used in this study, obtained from the OpenML repository (https://openml.org/). Note that `brazilian_houses` has been abbreviated to `b_h`.

Problem	#features	#samples	Problem	#features	#samples
quake	3	2,178	socmob	5	1,156
abalone	8	4,177	b_h	12	10,692
house_16h	16	22,784	elevators	18	16,599

4.2 Experimental design

The code was implemented in Python, with the TPOT Python library [13] providing the main TPOT implementation and Optuna Python library [1] providing the Bayesian optimisation (BO) tools. Implementation details and documentation for the BO-TPOT suite of tools along with the source code are available via GitHub (link withheld for anonymity during the review).

To observe their statistical behaviour, the algorithms in the BO-TPOT suite were compared based on the results obtained from 21 runs, against the baseline provided by TPOT-BASE with default TPOT parameters including population size $nP = 100$, evolved for a total of $nG_t = 100$ generations, with mutation rate of 0.9, crossover rate of 0.1 and 5-fold cross-validation of mean squared error during fitting. For TPOT-BO-S, the set of input pipelines was the output for TPOT-BASE after 80 generations and the number of evaluations was $nE = 2000$. For TPOT-BO-ALT, the number of generations per TPOT step was $nG_s = 8$ and the number of iterations was $nI = 10$.

All runs were carried out using both discrete and continuous hyper-parameter spaces during the BO steps, in order to assess any advantage that a fine-grained search might provide.

Because so many pipeline evaluations must be performed, the run time of a single experiment can potentially span days. Therefore, for problems with a large number of features and samples (b_h, house_16h, elevators), the evaluation time per pipeline was restricted to 1 minute; the rest were allowed the TPOT default of 5 minutes per pipeline.

5 RESULTS AND DISCUSSION

The results from the experiments are presented in Table 2 for each problem, sorted by the size of the problem (smallest to largest). For each problem, the best, worst, median and mean cross validation (CV) error and the standard deviation is listed based on 21 runs. For TPOT-BO-S and TPOT-BO-ALT, the results are presented for both discrete and continuous hyper-parameter spaces.

At a high-level, Table 2 shows that there is usually some benefit gained by performing BO to fine-tune hyper-parameters, though not necessarily a lot. With the exception of the single best run for socmob and house_16h, and the median run for elevators; the best, median and mean CV error values are all found in the TPOT-BO-S and TPOT-BO-ALT columns. Of these, the majority were obtained when continuous parameter spaces were used in the BO step, with the results split evenly between TPOT-BO-S and TPOT-BO-ALT. TPOT-BO-S fared better when applied to the quake, abalone and elevators problems, but not so well on b_h and house_16h. Table 2 also provides the win/tie/loss results from a pairwise comparison of both TPOT-BO-S and TPOT-BO-ALT (for discrete and continuous parameter spaces) against TPOT-BASE, using the Mann-Whitney-Wilcoxon (MWW) statistical test [12]. Here, method A is said to "win" over method B (and B "lose" to A), if the mean CV error of A is less than that of B, with a p-value less-than-or-equal-to 0.05. If the p-value is greater than 0.05, we cannot say with any confidence that the samples are drawn from different distributions, and the result is deemed a "tie".

Tables 3 and 4 provide some deeper insights into the reasons for the obtained results. Table 3 provides the results for each problem from a single run and compares the results obtained from TPOT-BASE, TPOT-BO-S and TPOT-BO-ALT for that same run. The run chosen for this comparison is the run for which the pipeline selected after 80 TPOT generations had the median CV error value across all 21 runs. Table 4 gives statistics about the number of pipelines evaluated up to 80 TPOT generations, number of pipelines with unique structures among them, the number of non-dominated solutions (i.e., solutions for which there does not exist another solution that is better or equal in all objectives, and strictly better in at least one) considering CV error and number of operators as the metric, the number of operators in the chosen pipeline, the number of hyper-parameters in the pipeline and the number of matching pipelines evaluated so far. One can note that the best pipeline (smallest CV error) for quake and house_16h had 6 operators each, but their associated number of hyper-parameters are different, i.e., 30 and 22. The number of hyper-parameters dictates the size of the BO search space as opposed to the number of operators, although TPOT uses number of operators (and CV error) as a metric.

Results of the median runs of abalone and socmob provide some reasons as to why TPOT-BO-S might perform well on some problem instances, but not others. During its operation, TPOT maintains a set of pipelines representing the Pareto front approximation that demonstrates the trade-off between pipeline complexity (number of operators) and predictive performance (CV error). By the time the population (of size 100) has been evolved by TPOT for 80 generations, approximately 8000 pipelines have been evaluated. Of these ~8000 pipelines, many have identical structures and can be grouped together. When selecting a pipeline for the BO step, TPOT-BO-S

Table 2: Experimental results. Values in parentheses represent (number of variables/sample points) for a given data set. Also given are wins/ties/losses against TPOT-BASE with a p-value less-than-or-equal-to 0.05 considered significant.

	TPOT-BASE		TPOT-BO-S		TPOT-BO-ALT	
	at 80 gens	Complete	Discrete	Continuous	Discrete	Continuous
quake (3/2178)						
best	3.519215e-02	3.518110e-02	3.519215e-02	3.507130e-02	**3.503905e-02**	3.511628e-02
median	3.534442e-02	3.532573e-02	3.533886e-02	**3.532163e-02**	3.540126e-02	3.537806e-02
mean	3.534209e-02	3.532918e-02	3.533423e-02	**3.531965e-02**	3.534641e-02	3.533432e-02
std dev	8.501796e-05	8.567492e-05	8.072723e-05	9.721603e-05	1.041078e-04	9.508804e-05
socmob (5/1156)						
best	1.344280e+02	**1.317632e+02**	1.344280e+02	1.344280e+02	1.356928e+02	1.346952e+02
median	1.624169e+02	1.598869e+02	1.586976e+02	**1.579499e+02**	1.676150e+02	1.608166e+02
mean	1.594125e+02	1.577441e+02	1.584519e+02	1.574076e+02	1.600386e+02	**1.557128e+02**
std dev	1.023297e+01	1.045449e+01	9.799378e+00	1.043645e+01	1.411075e+01	1.140616e+01
abalone (8/4177)						
best	4.219117e+00	4.216435e+00	4.196940e+00	4.201264e+00	4.184635e+00	**4.181162e+00**
median	4.254765e+00	4.251526e+00	**4.239837e+00**	4.240244e+00	4.249577e+00	4.253169e+00
mean	4.254398e+00	4.250179e+00	4.241122e+00	**4.238250e+00**	4.240930e+00	4.250218e+00
std dev	1.684278e-02	1.760437e-02	1.845866e-02	1.851458e-02	2.434576e-02	2.005804e-02
b_h (12/10692)						
best	4.077523e+01	4.077434e+01	4.076721e+01	4.076779e+01	4.076762e+01	**4.075871e+01**
median	4.081545e+01	4.081435e+01	4.080794e+01	4.081545e+01	4.081387e+01	**4.080403e+01**
mean	4.080993e+01	4.080751e+01	4.080497e+01	4.080556e+01	4.080048e+01	**4.079465e+01**
std dev	1.952899e-02	2.063165e-02	2.136754e-02	2.186512e-02	2.069696e-02	2.440020e-02
house_16h (16/22784)						
best	8.897154e+08	**8.857484e+08**	8.897154e+08	8.890021e+08	8.902213e+08	8.713201e+08
median	9.280501e+08	9.255052e+08	9.245483e+08	9.199194e+08	9.195092e+08	**9.144056e+08**
mean	9.278689e+08	9.256198e+08	9.235675e+08	9.187529e+08	9.126147e+08	**9.024381e+08**
std dev	1.380190e+07	1.491871e+07	1.498829e+07	1.682181e+07	1.356531e+07	1.196513e+07
elevators (18/16599)						
best	3.446479e-06	3.442113e-06	**3.438447e-06**	3.439498e-06	3.491153e-06	3.488053e-06
median	3.621078e-06	**3.581124e-06**	3.593705e-06	3.593557e-06	3.634452e-06	3.641495e-06
mean	3.583750e-06	3.575972e-06	**3.570815e-06**	3.573468e-06	3.589411e-06	3.595229e-06
std dev	6.418178e-08	6.544894e-08	6.791904e-08	6.622412e-08	5.055332e-08	5.341751e-08
wins/ties/losses:			0/6/0	**1/5/0**	**1/5/0**	**1/5/0**

Table 3: Detailed results by problem, for the run with pipeline selected for BO step after 80 TPOT generations having median CV error value from 21 runs, and the results obtained from TPOT-BASE, TPOT-BO-S and TPOT-BO-ALT on the same run.

	TPOT-BASE		TPOT-BO-S		TPOT-BO-ALT	
Problem	at 80 gens	Complete	Discrete	Continuous	Discrete	Continuous
quake	3.534442e-02	3.534442e-02	3.531071e-02	3.528877e-02	3.528311e-02	**3.525003e-02**
socmob	1.624169e+02	1.620516+e02	1.579280+e02	1.624169e+02	1.661091e+02	**1.578522e+02**
abalone	4.254765e+00	4.252000e+00	4.229589e+00	**4.225819e+00**	4.261785e+00	4.253169e+00
b_h	4.081545e+01	4.081435e+01	4.081545e+01	4.081545e+01	**4.076762e+01**	4.078809e+01
house_16h	9.280501e+08	9.274642e+08	9.228110e+08	9.199194e+08	9.057001e+08	**9.015522e+08**
elevators	3.621078e-06	3.621078e-06	3.621078e-06	3.621078e-06	3.619228e-06	**3.615875e-06**

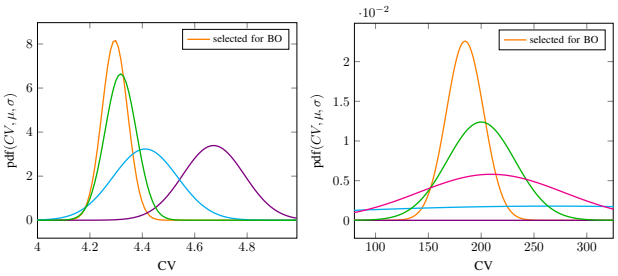
Table 4: Statistics at 80 TPOT generations for the same runs as detailed in Table 3. Given are the total numbers of: evaluated pipelines; unique pipeline structures; non-dominated pipeline structures, based on CV and number of operators; operators, hyper-parameters of the pipeline selected by TPOT-BO-S and pipelines with a structure matching it.

	Pipelines			Selected pipeline		
Problem	Evaluated	Unique	ND	Operators	HPs	Matching
quake	7525	1031	6	6	30	9
socmob	7791	2948	5	5	27	16
abalone	7816	938	4	4	17	75
b_h	7536	1540	6	7	0[†]	1
house_16h	7495	1176	5	6	22	1
elevators	7763	1823	4	4	20	63

[†]Selected b_h pipeline consisted only of RidgeCV, CombineDFs, StandardScaler and MaxAbsScaler operators — none of which require any hyper-parameters.

chooses the one with the lowest CV error. However, the lowest CV error is not the only consideration that is important. Partitioning the evaluated pipelines by unique structure means that the mean CV error, standard deviation and probability density function can be computed for each group. Figure 1 provides plots of these probability density functions for all of the unique, non-dominated pipeline structure groups for abalone and socmob, respectively — along with an indication of which group the pipeline selected for the BO step of TPOT-BO-S belongs to.

Assuming the pipeline CV errors are normally distributed (as CV is bounded to one side, it is unlikely the distribution is truly normal, but the assumption is made for illustrative purposes), these plots can be used to estimate the likelihood a pipeline may yield a better CV error value after improvement. This likelihood is represented by the area under the curves to the left of a given CV error. Figure 1(a) indicates that the choice of pipeline selected by TPOT-BO-S was probably a good one, as there is no other curve which has a significantly larger area beneath it to the left of the orange curve. Conversely, Figure 1(b) suggests that the choice of



(a) abalone (b) socmob

Figure 1: Estimated distribution of non-dominated, unique pipelines for (a) abalone and (b) socmob after 80 generations.

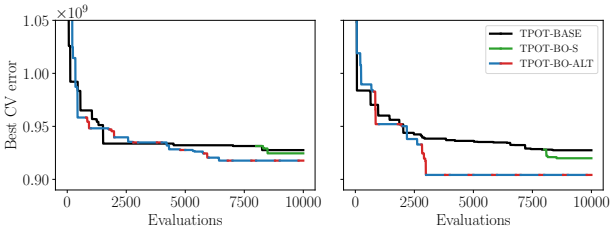


Figure 2: house_16h convergence plots for median run of TPOT-BO-S and TPOT-BO-ALT with discrete (left) and continuous (right) parameter spaces. The phases of TPOT-BO-ALT are coloured blue (TPOT) and red (BO).

pipeline by TPOT-BO-S for this instance of socmob was probably not the best since both the green and pink curves have significant areas below them to the left of the orange curve (the blue one too, but that has a too large standard deviation to provide any reliable predictions). This indicates that there is a likelihood that better CV errors can be achieved using those structures. This demonstrates there is a significant amount of risk involved when selecting a pipeline for improvement using best CV error as the sole criteria. These inferences are both supported by the results in Table 3, where TPOT-BO-S performed best for this run on abalone but not as well on socmob. During the course of evolution, TPOT ranks pipelines based on CV error value and the number of operators, but neither of these attributes are necessarily a good indicator of how well the BO step is likely to perform in TPOT-BO-S. Without a significant number of samples, it is difficult to make any predictions about the distribution of pipeline CV errors, and the more samples that are provided to a BO model during its construction, the more reliable its predictions will likely be. This is reflected in the results for abalone and house_16h in Table 4. The BO model for abalone was constructed using information from 75 matching pipelines; in contrast to the model for house_16h, which was initialised with a single data point. Here, a lot of the budget must be invested in sampling the parameter space to improve the fidelity model.

The performance of a BO model is strongly related to the accuracy of its surrogate. The number of variables and the number of possible values those variables can take are the two key considerations when applying BO. In the case of TPOT-BO-S and TPOT-BO-ALT, the variables are the hyper-parameters for the machine learning models which comprise the operators for a given pipeline. Once a pipeline is selected for a BO step, its structure is effectively frozen and cannot change until the BO step is complete. The structure of a pipeline is determined by its operators and the order in which they appear, but an arbitrary operator can require any number of hyper-parameters; which means the number of operators for a pipeline is unlikely to be a useful predictor of BO complexity. This is visible from the b_h results in Table 4, where the pipeline selected for the BO step has 7 operators but, notably, no hyper-parameters at all. Clearly, there is nothing BO could do to improve this pipeline.

Another example of the importance of number of hyper-parameters is visible from the results for abalone in Table 4. Here the chosen pipeline has few hyper-parameters, and this is also the problem instance where TPOT-BO-S performed the best. However, one should not take the number of hyper-parameters, or number of matching pipelines, to be an absolute measure to predict the potential success of the BO step. The results in Table 4 for elevators cautions against this: for the median run, the selected pipeline had 20 hyper-parameters and 63 matching pipelines, but TPOT-BO-S was unable to achieve any improvements. This suggests that a more sophisticated approach is required.

Given that TPOT-BO-S only gets one chance to make its choice, and then must invest its entire allocated BO computing budget in whichever pipeline it selects, it makes sense that having a smaller number of choices to select from will increase its chances of performing well. TPOT-BO-ALT "hedges its bets" by investing smaller portions of its BO budget incrementally, meaning it will not spend long improving any single pipeline, but instead mitigates its risk by improving pipelines periodically, e.g., every 8 generations.

Even when TPOT-BO-S did not perform as well, as with the house_16h problem, there were differences in the performance between the discrete and the continuous variants of the BO search. Figure 2 presents the convergence plots for the median run of TPOT-BO-S and TPOT-BO-ALT on the house_16h problem. This figure demonstrates the behaviour observed when the BO-TPOT suite is applied to this problem with both discrete (left) and continuous (right) parameter spaces. The plots show that there is clear and noticeable improvement in the CV error when a continuous parameter space is used. This cannot be attributed to the effect of BO alone, as this improvement is not as dramatic when BO is applied to the same pipeline using a discrete parameter space. It is harder to judge the difference between the two parameter spaces when used by TPOT-BO-ALT, as the populations diverge after the first BO step. However, the initial improvement shown after this first BO step using a continuous space is noticeably greater than for the discrete space. This is also supported by Tables 2, 3 and 4.

Finally, although searching a continuous parameter space does appear to provide some benefit, it is also worth noting that TPOT already uses a reasonably fine grained descretization — steps of 0.05 in most cases — so there is only a limited room to improve through this modification alone.

Algorithm 4: TPOT-BO-AUTO

Input: D: training data; nG_t: total number of generations; nP: population size; ρ: GP parameters
Output: S: set of evaluated pipelines

$T \leftarrow$ TPOTRegressor object initialized using nP and ρ
$S \leftarrow \emptyset$, evaluated pipeline set
$\Delta_T, \Delta_B \leftarrow$ arbitrarily large values such that $\Delta_T > \Delta_B$
DoTPOT $\leftarrow$ True
for $nG_t - 1$ generations† **do**
  if $\Delta_T = \Delta_B$ **then**
    DoTPOT $\leftarrow$!DoTPOT — toggle from most recent operation
  else
    DoTPOT $\leftarrow \Delta_T > \Delta_B$
  end if
  if DoTPOT = True **then**
    $S \leftarrow$ update with result of fitting T on D and S for 1 generation
  else
    $S \leftarrow$ TPOT-BO-S(D, T, nP, ρ)
  end if
  $\Delta_T, \Delta_B \leftarrow$ update gradients
end for
return S

†$nG_t - 1$ as nP evaluations required to construct the initial population.

Adaptive variant : TPOT-BO-AUTO

While dividing the computational budget into nI iterations allows TPOT-BO-ALT to avoid investing all its resources into improving just one pipeline, the choice of nI is still arbitrary and there is a risk that the budget is not being used as efficiently as possible. TPOT-BO-AUTO aims to address this by deciding whether to use a TPOT or TPOT-BO-S step in each generation.

A population S with size nP is initialized and evolved for one generation using TPOT. BO is then applied for nP pipeline evaluations, the equivalent of one TPOT generation. Then, for the remaining $nG_t - 3$ generations, the gradient of the best-so-far CV error values is computed for the most recent execution of each process (Δ_T, Δ_B) and the process that produced the steepest gradient is selected for the next nP evaluations. If both gradients are equal, then the process which was *not* the most recent is selected. The pseudo-code for TPOT-BO-AUTO is given in Algorithm 4.

As with the experiments in Section 4, TPOT-BO-AUTO was applied to the six OpenML problems for 21 runs each. The results are presented in Table 5 along with the results of TPOT-BASE and TPOT-BO-ALT, and their MWW statistical significance tests. They show that TPOT-BO-AUTO has competitive, but ultimately slightly worse over-all performance than its two counterparts. However, notably, it performed best for the two largest instances. Figure 3 plots the convergence of TPOT-BO-AUTO against TPOT-BASE and TPOT-BO-ALT, for the `socmob` (top) and `abalone` (bottom) problems using discrete (left) and continuous (right) parameter spaces.

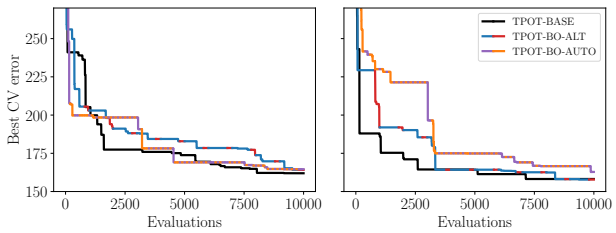
It can be seen from these plots that, typically, TPOT-BO-AUTO alternates between TPOT and BO while searching for improvements that can be made. When a potential area for improvement is identified using either process, the plots also show that it is able to exploit this change, by persisting with the same process until there is a local convergence. TPOT-BO-ALT deterministically switches between TPOT and BO processes, introducing an extra parameter that must be tuned, and pays no attention to whether a given process is likely to produce further improvements, if it were given more time. These results demonstrate that TPOT-BO-AUTO works to address this in principle, even if though its design requires some further improvements to improve the quality of results.

Table 5: TPOT-BO-AUTO results. Values in parentheses represent (number of variables/sample points) for a given data set. Also given are wins/ties/losses against TPOT-BASE with a p-value less-than-or-equal-to 0.05 considered significant.

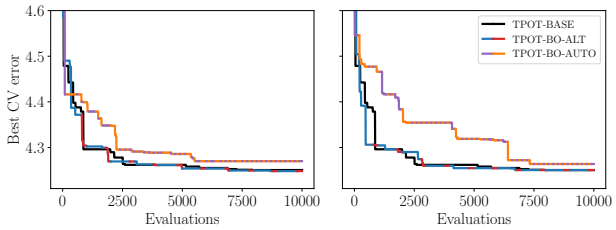
	TPOT-BASE	TPOT-BO-ALT		TPOT-BO-AUTO	
		Discrete	Continuous	Discrete	Continuous
`quake` (3/2178)					
best	3.518110e-02	**3.503905e-02**	3.511628e-02	3.514637e-02	3.528579e-02
median	3.532573e-02	3.540126e-02	3.537806e-02	**3.528434e-02**	3.535106e-02
mean	3.532918e-02	3.534641e-02	3.533432e-02	**3.523471e-02**	3.534614e-02
std dev	8.567492e-05	1.041078e-04	9.508804e-05	7.993021e-05	3.322743e-05
`socmob` (5/1156)					
best	**1.317632e+02**	1.356928e+02	1.346952e+02	1.454596e+02	1.447175e+02
median	**1.598869e+02**	1.676150e+02	1.608166e+02	1.646004e+02	1.628094e+02
mean	1.577441e+02	1.600386e+02	**1.557128e+02**	1.649793e+02	1.611582e+02
std dev	1.045449e+01	1.411075e+01	1.140616e+01	9.099695e+00	7.553073e+00
`abalone` (8/4177)					
best	4.216435e+00	4.184635e+00	**4.181162e+00**	4.232186e+00	4.233097e+00
median	4.251526e+00	**4.249577e+00**	4.253169e+00	4.270114e+00	4.263818e+00
mean	4.250179e+00	**4.240930e+00**	4.250218e+00	4.269536e+00	4.263825e+00
std dev	1.760437e-02	2.434576e-02	2.005804e-02	1.456893e-02	1.322812e-02
`b_h` (12/10692)					
best	4.077434e+01	4.076762e+01	**4.075871e+01**	4.077620e+01	4.076998e+01
median	4.081435e+01	4.081387e+01	**4.080403e+01**	4.078665e+01	4.081343e+01
mean	4.080751e+01	4.080048e+01	**4.079465e+01**	4.080467e+01	4.080363e+01
std dev	2.063165e-02	2.069696e-02	2.440020e-02	2.299884e-02	2.323473e-02
`house_16h` (16/22784)					
best	**8.857484e+08**	8.902213e+08	8.713201e+08	9.110699e+08	8.877822e+08
median	9.255052e+08	9.195092e+08	9.144056e+08	9.204535e+08	**8.972125e+08**
mean	9.256198e+08	9.126147e+08	9.024381e+08	9.211489e+08	**9.021325e+08**
std dev	1.491871e+07	1.356531e+07	1.196513e+07	5.188537e+06	1.243686e+07
`elevators` (18/16599)					
best	**3.442113e-06**	3.491153e-06	3.488053e-06	3.477236e-06	3.448599e-06
median	3.581124e-06	3.634452e-06	3.641495e-06	3.592937e-06	**3.577999e-06**
mean	3.575972e-06	3.589411e-06	3.595229e-06	3.606481e-06	**3.575885e-06**
std dev	6.544894e-08	5.055332e-08	5.341751e-08	5.653632e-08	5.003173e-08
wins/ties/losses:		**1/5/0**	**1/5/0**	1/3/2	1/4/1

Limitations of TPOT and single BO selection

Although competitive, the results obtained by any of the methods were not significantly better than TPOT-BASE. They also suggest that by 80 generations, TPOT-BASE has already done a lot of the "heavy-lifting" — leaving little space for any improvement. TPOT-BO-ALT and TPOT-BO-AUTO attempted to improve on this by beginning their search from scratch, however all three methods are limited by two main factors. Figures 4 and 5 illustrate these two principles on the `abalone` data set. The first is the fact choosing a single pipeline for improvement by BO means that some amount of luck is involved in determining the quality of the result. Figure 4 plots the best 20 unique pipeline structures after 80 TPOT generations, ordered by CV and indicated by large, pale green dots. The remaining budget was applied to all 20 structures individually, the results of which are given by the smaller dark purple dots. As highlighted by the yellow and red circles, TPOT-BO-S selected the structure with the best CV at 80 TPOT generations and improved it from around 4.244 to 4.227; however, the plot shows there are several other choices which have worse initial CV values, but which



(a) socmob



(b) abalone

Figure 3: Convergence plots for the median runs of TPOT-BO-ALT and TPOT-BO-AUTO on socmob (top) and abalone (bottom) for discrete (left) and continuous (right) parameter spaces. The phases of TPOT-BO-ALT are coloured blue (TPOT) and red (BO); the phases of TPOT-BO-AUTO are coloured purple (TPOT) and orange (BO).

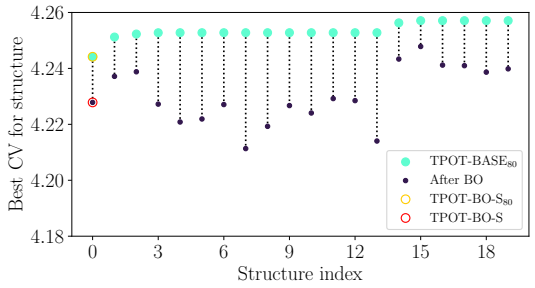


Figure 4: BO is applied to the best 20 pipeline structures.

produce a better result once BO was applied — demonstrating the limitations of selecting a single pipeline structure for improvement, especially when that selection is made by CV alone.

The second factor is the mechanism that TPOT itself employs when selecting pipelines to carry forward to the next generation. When deciding the active population for the next generation, TPOT orders the set of previously evaluated pipelines using a non-dominated sort on CV and number of operators and chooses the best nP pipelines to carry forward. While this method ensures that pipeline complexity is reduced — and important consideration for the efficient operation of its genetic programming procedures — the diversity of the population used to generate offspring can become severely limited. Figure 5 tracks unique structures selected

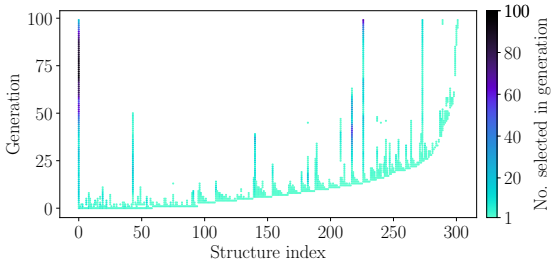


Figure 5: Tracked TPOT-BASE population diversity.

in each generation of TPOT-BASE. The horizontal axis represents the index of any pipeline structure included in an active population throughout the search, with the vertical axis being the generation it was included. A pale green dot indicates that only one pipeline with that structure was included in that generation's active population, and a dark purple dot indicates 100 was selected — i.e., the entire active population. This plot clearly shows that the majority of pipelines included in the active population for the entire search were drawn from one single structure, with very little diversity shown in the population after approximately 30 generations.

6 CONCLUSION AND FUTURE WORK

The experimental results presented in Section 5 suggest that CV and number of operators, as used in native TPOT, are not particularly effective when deciding where to apply BO. Due to the nature of TPOT's selection mechanism, the initial search is skewed towards pipeline structures with fewer operators, meaning more complex pipeline spaces are not explored. While this is important to keep the GP processes that TPOT is based on in check, it also results in less diversity in the set of evaluated pipelines, which is detrimental when constructing BO models. Even if a given pipeline structure is theoretically optimal, it is unlikely to produce a good CV error if very few sets of hyper-parameters have been sampled from it. Conversely, sampling a sub-optimal pipeline structure many times gives it more chances to produce a better CV error — especially if it has fewer operators. The results in this study show that there is often an initial "jump" in quality of pipelines when BO is applied, especially in the case of continuous parameter spaces, but this quickly tapers off. This could be capitalized on by dividing the BO budget among multiple pipelines that can be optimized simultaneously; competing for resources and improving the overall quality of the pipelines in the population. This leads to three fundamental questions to consider moving forward (a) How should the pipelines for BO be selected? (b) How many pipelines should be selected? and (c) How to best divide the allocated budget amongst them? The first has been partially addressed in the main experiments conducted as a part of this study. The other two still present interesting challenges and the insights gained from the study can be leveraged to develop more efficient hybridization strategies.

ACKNOWLEDGMENT

The authors acknowledge the financial support from Honda Research Institute Europe for this work.

REFERENCES

[1] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. 2019. Optuna: A Next-generation hyperparameter optimization framework. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2623–2631.

[2] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. 2011. Algorithms for hyper-parameter optimization. *Advances in Neural Information Processing Systems* 24 (2011), 1–9.

[3] James Bergstra and Yoshua Bengio. 2012. Random search for hyper-parameter optimization. *Journal of Machine Learning Research* 13, 2 (2012), 281–305.

[4] James Bergstra, Daniel Yamins, and David Cox. 2013. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In *International Conference on Machine Learning*. 115–123.

[5] François-Michel De Rainville, Félix-Antoine Fortin, Marc-André Gardner, Marc Parizeau, and Christian Gagné. 2012. DEAP: A Python framework for evolutionary algorithms. In *Proceedings of the 14th annual conference companion on Genetic and evolutionary computation*. 85–92.

[6] Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Springenberg, Manuel Blum, and Frank Hutter. 2015. Efficient and robust automated machine learning. In *Advances in Neural Information Processing Systems 28 (2015)*. 2962–2970.

[7] Pieter Gijsbers, Marcos LP Bueno, Stefan Coors, Erin LeDell, Sébastien Poirier, Janek Thomas, Bernd Bischl, and Joaquin Vanschoren. 2022. AMLB: an AutoML benchmark. *arXiv preprint arXiv:2207.12560* (2022).

[8] Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren. 2019. *Automated machine learning: methods, systems, challenges*. Springer Nature.

[9] Donald R Jones, Matthias Schonlau, and William J Welch. 1998. Efficient global optimization of expensive black-box functions. *Journal of Global optimization* 13, 4 (1998), 455–492.

[10] Zohar Karnin, Tomer Koren, and Oren Somekh. 2013. Almost optimal exploration in multi-armed bandits. In *International Conference on Machine Learning*. 1238–1246.

[11] Lars Kotthoff, Chris Thornton, Holger H Hoos, Frank Hutter, and Kevin Leyton-Brown. 2019. Auto-WEKA: Automatic model selection and hyperparameter optimization in WEKA. In *Automated Machine Learning*. Springer, Cham, 81–95.

[12] Henry B Mann and Donald R Whitney. 1947. On a test of whether one of two random variables is stochastically larger than the other. *The Annals of Mathematical Statistics* (1947), 50–60.

[13] Randal S Olson, Nathan Bartley, Ryan J Urbanowicz, and Jason H Moore. 2016. Evaluation of a tree-based pipeline optimization tool for automating data science. In *Proceedings of the genetic and evolutionary computation conference 2016*. 485–492.

[14] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. 2012. Practical bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems*. 2951–2959.

[15] P. Stetsenko. 2020. *Machine learning with Python and H2O*. http://docs.h2o.ai/h2o/latest-stable/h2o-docs/booklets/PythonBooklet.pdf

[16] Anh Truong, Austin Walters, Jeremy Goodsitt, Keegan Hines, C Bayan Bruss, and Reza Farivar. 2019. Towards automated machine learning: Evaluation and comparison of AutoML approaches and tools. In *2019 IEEE 31st International Conference on Tools with Artificial Intelligence (ICTAI)*. IEEE, 1471–1479.

[17] Xilu Wang, Yaochu Jin, Sebastian Schmitt, and Markus Olhofer. 2022. Recent advances in Bayesian optimization. *arXiv preprint arXiv:2206.03301* (2022).