A Hierarchical Dissimilarity Metric for Automated Machine Learning Pipelines, and Visualizing Search Behaviour

Angus Kenny[1(✉)], Tapabrata Ray[1], Steffen Limmer[2], Hemant Kumar Singh[1], Tobias Rodemann[2], and Markus Olhofer[2]

[1] University of New South Wales, Canberra, Australia
angus.kenny@unsw.edu.au
[2] Honda Research Institute Europe, Offenbach, Germany

Abstract. In this study, the challenge of developing a dissimilarity metric for machine learning pipeline optimization is addressed. Traditional approaches, limited by simplified operator sets and pipeline structures, fail to address the full complexity of this task. Two novel metrics are proposed for measuring structural, and hyperparameter, dissimilarity in the decision space. A hierarchical approach is employed to integrate these metrics, prioritizing structural over hyperparameter differences. The Tree-based Pipeline Optimization Tool (TPOT) is utilized as the primary automated machine learning framework, applied on the *abalone* dataset. Novel visual representations of TPOT's search dynamics are also proposed, providing some deeper insights into its behaviour and evolutionary trajectories, under different search conditions. The effects of altering the population selection mechanism and reducing population size are explored, highlighting the enhanced understanding these methods provide in automated machine learning pipeline optimization.

Keywords: AutoML · TPOT · Visualization · Search characteristics

1 Introduction

Automated machine learning (AutoML) is a rapidly growing field, focusing on automating the application of machine learning to classification and regression tasks [6]. In this domain, machine learning models can function independently or sequentially, with one model's output feeding into the next. Such configuations, called machine learning *pipelines* [9], harness the combined strengths of multiple models to enhance overall performance. Despite growing interest, fitness landscape analysis in AutoML, especially regarding pipeline optimization, remains an area with significant unresolved questions [18]. A primary issue is

Supplementary Information The online version contains supplementary material available at https://doi.org/10.1007/978-3-031-56855-8_7.

S. Smith et al. (Eds.): EvoApplications 2024, LNCS 14635, pp. 115–129, 2024.
https://doi.org/10.1007/978-3-031-56855-8_7

the development of an effective metric for measuring the dissimilarity between solutions in the decision space. This task is complex, as pipeline hyperparameters can take continuous, discrete or categorical values, are often conditional, and typically have hierarchical relationships.

State-of-the-art investigations in this area typically operate on a severely reduced operator set or rely on restricting pipeline complexity [3,11,13,15–17]. These approaches are very limiting, and do not generalize to arbitrary pipelines. So far, there are no attempts (to the authors' knowledge) to define a dissimilarity metric for arbitrarily complex machine learning pipelines, and addressing this gap in the literature would represent a significant contribution to both the AutoML and broader machine learning communities.

The tree-based pipeline optimization tool (TPOT) [10], a well known Python library for automating machine learning, exemplifies this. It employs genetic programming to optimize machine learning pipelines, searching for the best combination of data preprocessing and modeling steps. Section 2 explains how TPOT produces new pipelines through mutation and crossover, highlighting unique aspects of its methodology. Within TPOT, mutations can alter both pipeline operators and hyperparameters. Consequently, this paper proposes two metrics: one for measuring the dissimilarity between pipeline structures, and another for hyperparameter differences, as detailed in Sect. 3. These metrics are integrated using a hierarchical approach to emphasize structural over hyperparameter changes, reflecting their greater impact on pipeline behaviour. To this end, the concept of pipeline structure is formally defined.

In Sect. 4, novel methods of visually representing the behaviour and evolutionary trajectory of TPOT search are introduced. Experiments conducted on the well-studied *abalone* dataset [4] demonstrate the utility of these visualizations under various conditions, including the effects of changing the population selection mechanism to focus solely on cross-validation error and reducing population size. These experiments underscore the enhanced insights into TPOT's search dynamics offered by the proposed visual representations.

2 Background

2.1 TPOT Pipeline Representation

As its name suggests, the tree-based pipeline optimization tool (TPOT) uses a tree-based representation to manage its pipelines. The pipelines are composed of a combination of transformation operators—each with their own set of hyperparameters. These operators take the form:

```
OpA(input_matrix, OpA__paramA1=True, OpA__paramA2=2.7).
```

Here, the operator is called `OpA` and has an input and two hyperparameters. The input (`input_matrix`) is always first, followed by the associated hyperparameters, which use the naming convention `operatorName__hyperparameterName`.

The set of operators—along their associated hyperparameters and possible hyperparameter values—are maintained in an object called the `pset`. When the

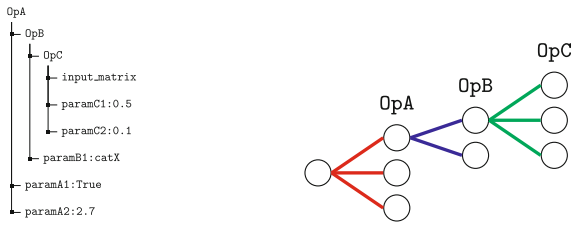

Fig. 1. Nested tree (a) and tree graph (b) representation of pipeline p.

pipeline is evaluated on some set of input data, the data is transformed by each operator in turn, passing the output of one operator to the input of the next, until the root of the tree is reached. During search, TPOT maintains the set of evaluated pipelines using a Python dictionary, which employs a nested bracket string representation of each pipeline as its keys, e.g.,

$$p = \texttt{OpA(OpB(OpC(input\_matrix, OpC\_\_paramC1=0.5, OpC\_\_paramC2=0.1),}$$
$$\texttt{OpB\_\_paramB1=catX), OpA\_\_paramA1=True, OpA\_\_paramA2=2.7).}$$

Figure 1 gives two visual representations of pipeline p: a nested tree representation showing the operators and their respective hyperparameter values; and an abstract tree graph representation, emphasizing the connections between operators. In the tree graph representation, the input nodes are always uppermost.

Although the pipeline representations in Fig. 1 have the appearance of being tree-like, the "branches" of these trees are simply the hyperparameters of each operator. Stripped of these hyperparameters, the pipeline structures shown are essentially linear. In order to allow for more complex structures, TPOT makes use of the `CombineDFs` operator. This operator puts the "tree" in TPOT, and has the form,

$$\texttt{CombineDFs(input\_matrix, input\_matrix).}$$

Each of the `input_matrix` nodes may contain an entire subtree, the output matrices of which are merged—by horizontal stacking—to produce a combined input for the next operator in the pipeline. Figure 2(a) illustrates this principle.

2.2 Producing New Pipelines

Throughout its execution, TPOT uses the genetic programming (GP) tools available in the DEAP Python library [2] to search the space of possible pipelines. With each generation, a population of N parent solutions is chosen using a non-dominated sort, with cross-validation error and number of operators as the selection criteria. A set of N offspring solutions is produced from this population, using standard one-point crossover, or one of three mutation operations.

The INSERT operation creates a new tree by slicing the main tree at an arbitrary input node and inserting a new operator between the two halves of the sliced tree.

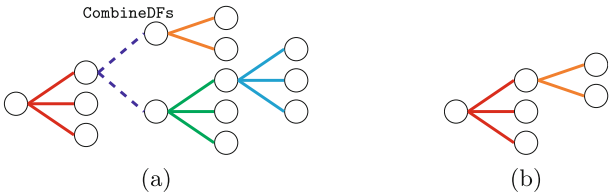

Fig. 2. The `CombineDFs` operator allows more complex pipeline structures to be constructed (a). However, when the SHRINK operation is applied to it (b), the entire subtree in its second input is removed as well.

The SHRINK operation reduces the size of the pipeline by arbitrarily removing one of its operators. It does this by slicing the pipeline tree at an arbitrary (non-leaf) input node, deleting the subtree of the operator that the slice occurred at, and repairing the tree.

Finally, the REPLACE operation can be used to substitute both operator and hyperparameter nodes in the pipeline tree. An arbitrary node in the pipeline is selected and, depending on whether it is an input or hyperparameter node, the `pset` is used to replace the existing operator subtree with a new operator, or the value of the hyperparameter node with another hyperparameter value.

The default probabilities of crossover and mutation are 0.1 and 0.9, respectively. In the case of crossover, two pipelines that share at least one operator are selected, but only the first offspring is retained.

A special case for `CombineDFs`: Typically, the mutation operations modify the pipeline by at most 1 operator. INSERT adds an operator, SHRINK removes an operator and REPLACE has no effect on the number of operators in the resulting pipeline. The exception to this is when the SHRINK, or REPLACE, mutation is applied to the `CombineDFs` operator. As the `CombineDFs` operator merges the output of two entire subtrees, there is no simple way to reconcile them, once `CombineDFs` is removed. TPOT addresses this issue by only treating the first input of `CombineDFs` as a "true" input, with the second input being considered a *de facto* hyperparameter. The consequence of this is that, when `CombineDFs` is removed from a pipeline, any subtree in its second input is also removed. Figure 2 illustrates the effect of removing `CombineDFs` on subtrees in both input positions.

Because the subtree in either input of `CombineDFs` can be arbitrarily large, arbitrarily large jumps in pipeline complexity can occur. A similar phenomenon is also observed for the REPLACE mutation operation.

2.3 The Tree Edit Distance Algorithm

In algorithmic graph theory, a common method of quantifying the dissimilarity between two arbitrary, labelled trees is through application of the tree edit distance algorithm [14]. Similar to the string edit distance algorithm, tree edit distance algorithms count the number of transformation operations needed to

convert one tree to another. Typically, tree edit distance algorithms consider the following three operations:

- Insertion: Inserting a node into one of the trees.
- Deletion: Deleting a node from one of the trees.
- Substitution: Replacing a node in one tree with a differently labelled one.

Formally, let T_1 and T_2 be two labelled trees with m and n nodes, respectively. A tree edit script is a sequence of edit operations that transforms T_1 into T_2. Each edit operation has an associated cost. The tree edit distance between T_1 and T_2 is then defined as the minimum total cost of any tree edit script that transforms T_1 into T_2. Typically, the cost of inserting or deleting a node is 1 and the cost of substituting two nodes is 1 if their labels are different, and 0 otherwise.

3 A Metric for Pipeline Dissimilarity

3.1 Pipeline Structures

Although pipelines produced by crossover and mutation may be unique to each other, they are not necessarily *structurally* unique. During its operation, TPOT can be thought about as searching a hierarchy of two distinct classes of spaces. The first is the space of all possible combinations of operators, which it explores by using genetic programming to mutate and recombine tree representations of previously evaluated pipelines. The second is the subspace of all possible hyperparameter combinations for each unique combination of operators, which it explores using a grid-based search (having discretized any continuous parameter spaces). It is not possible to keep improving a pipeline by optimizing its hyperparameters alone; eventually a ceiling will be reached, and the only way to achieve further improvement is to change the combination of operators. This implies that search in the space of operator combinations is more influential than search in the space of hyperparameter combinations, so it is useful to have a method of grouping evaluated pipelines together by their so-called *pipeline structure*.

Let p, and q be two pipelines. Pipelines p and q are said to be unique to each other if they have at least one dissimilar hyperparameter value, or do not share the same configuration of operators. A pipeline structure is a subset in the set of evaluated pipelines, partitioned such that every pair of pipelines within a subset are unique and share the same set of operators, in the same configuration. As a given operator will have the same hyperparameters, regardless of its position in the pipeline, a pipeline structure can be represented by its configuration of operators and inputs alone. For example, let p and q be pipelines represented by the two TPOT nested bracket strings:

$p =$ `OpA(OpB(input_matrix, OpB__paramB1=0.1, OpB__paramB2=0.5),`
`OpA__paramA1=True, OpA__paramA2=0.7);`

$q =$ `OpA(OpB(input_matrix, OpB__paramB1=0.6, OpB__paramB2=0.5),`
`OpA__paramA1=False, OpA__paramA2=0.2).`

These pipelines are clearly unique to each other, as they do not share the same hyperparameter values. However, both have the same set of operators, organized in the same way, and therefore share the same structure. This is denoted with a bar over the pipeline symbol, and the tree-bracket string representation:

$$\bar{p} = \bar{q} = \{\texttt{OpA}\{\texttt{OpB}\{\texttt{input\_matrix}\}\}\}.$$

3.2 Quantifying Pipeline Dissimilarity

Comparing two machine learning pipelines in the decision space presents numerous challenges. When applied to a pipeline, the mutation operations provided by TPOT can modify either the entire structure or simply change the value of one or more hyperparameters. The effects of structural and hyperparameter mutations are typically asymmetrical, with structural changes having a much greater effect on the behaviour of the resulting pipeline than hyperparameter ones.

This section proposes metrics for quantifying both structural dissimilarity (denoted with $\bar{\delta}$) and hyperparameter dissimilarity (denoted with $\hat{\delta}$). These metrics are combined using a hierarchical approach to produce a general metric for tree-based machine learning pipeline dissimilarity (denoted with Δ).

Quantifying Structural Dissimilarity: With the exception of the `CombineDFs` operator, the INSERT operation typically adds one operator to a given pipeline, SHRINK removes one operator and REPLACE makes no change to its structure. In the tree edit distance algorithm, inserting a node into the tree increases its size by one, deleting a node from the tree reduces its size by one and substituting a node has no effect on the size of the tree. This indicates that there is a direct analogy between the INSERT, SHRINK and REPLACE mutation operations, and the insertion, deletion and substitution operations of tree edit distance algorithm—implying that tree edit distance is an appropriate metric by which to approximate the dissimilarity between pipeline structures, in the decision space.

Let p and q be the unique pipelines:

$p =$ `OpA(OpB(input_matrix, OpB__paramB1=0.8, OpB__paramB2=catX),`
`OpA__paramA1=True, OpA__paramA2=2.7);`

$q =$ `OpC(OpB(OpD(input_matrix, OpD__paramD1=6, OpD__paramD2=0.1),`
`OpB__paramB1=0.8, OpB__paramB2=catX), OpC__paramC1=0.3).`

The structural representations for p, q are:

$$\bar{p} = \{\texttt{OpA}\{\texttt{OpB}\{\texttt{input\_matrix}\}\}\};$$
$$\bar{q} = \{\texttt{OpC}\{\texttt{OpB}\{\texttt{OpD}\{\texttt{input\_matrix}\}\}\}\}.$$

The structural dissimilarity metric $\bar{\delta}$ uses the tree edit distance algorithm to count the minimum number of mutations required to transform pipeline structure $\bar{p}$ into $\bar{q}$. In this case $\bar{\delta} = 2$, as the transformation can be performed with a REPLACE mutation operation (`OpA` $\rightarrow$ `OpC`) and an INSERT operation (`OpD`).

Quantifying Hyperparameter Dissimilarity: The direct analogies that exist between its node operations and the TPOT mutation operations, suggest that tree edit distance is an appropriate metric to approximate the distance between pipeline structures in the decision space. However, this relationship does not extend to approximating the dissimilarity at the hyperparameter level. The reasons for this are illustrated through the following example. Let p and q be the pipelines:

$$p = \texttt{OpA(OpB(input\_matrix, OpB\_\_paramB1=0.8, OpB\_\_paramB2=catX),}$$
$$\texttt{OpA\_\_paramA1=True, OpA\_\_paramA2=2.7);}$$
$$q = \texttt{OpC(OpB(input\_matrix, OpB\_\_paramB1=0.8, OpB\_\_paramB2=catX),}$$
$$\texttt{OpC\_\_paramC1=0.3).}$$

Naively applying the tree edit distance metric in this situation suggests that two node substitutions (`OpA` $\rightarrow$ `OpC` and `OpA__paramA1=True` $\rightarrow$ `OpC__paramC1=0.3`) and one node deletion operation (`OpA__paramA2=2.7`) are required to transform p into q; when, in reality, this transformation could be achieved with a single REPLACE mutation operation (`OpA` $\rightarrow$ `OpC`). When `OpA` is replaced by `OpC`, all of the hyperparameters are automatically removed, and the hyperparameters for `OpC` are randomly assigned and inserted. The fact that changes at a structural level can have a large impact on the pipeline at a hyperparameter level, reinforces the notion that there is a hierarchical relationship between the two.

In the case where the structural dissimilarity between two unique pipelines is 0, then they must be distinguished by the differences in their hyperparameter values. Let r, s be the unique pipelines:

$$r = \texttt{OpC(OpD(input\_matrix, OpD\_\_paramD1=3, OpD\_\_paramD2=0.8),}$$
$$\texttt{OpC\_\_paramC1=1.4);}$$
$$s = \texttt{OpC(OpD(input\_matrix, OpD\_\_paramD1=1, OpD\_\_paramD2=0.3),}$$
$$\texttt{OpC\_\_paramC1=0.6).}$$

By inspection, both share the same structure: $\bar{r} = \bar{s} = \{\texttt{OpC}\{\texttt{OpD}\{\texttt{input\_matrix}\}\}\}$. Normalizing the hyperparameter values to the interval $[0, 1]$, these two pipelines can be represented as unique points contained within the unit hypercube, denoted with the notation, $\hat{r}, \hat{s}$. The hyperparameter dissimilarity between r and s is then the magnitude of the displacement vector between points $\hat{r}$ and $\hat{s}$:

$$\hat{\delta}(r, s) = ||\hat{r} - \hat{s}||,$$

where $||x||$ is the Euclidean norm of vector x. For individual hyperparameters r_i, s_i that take Boolean or categorical values, the component-wise difference, $\hat{r}_i - \hat{s}_i$, is set to 0 if the values are the same, and 1 otherwise.

A Hierarchical Dissimilarity Metric: While it is reasonable to directly compare two different pipeline structures, it does not make sense to compare the difference between the values of two different hyperparameters. Therefore, a hierarchical approach must be adopted when combining structural dissimilarity $\bar{\delta}$, and hyperparameter dissimilarity $\hat{\delta}$, to compute the overall dissimilarity between two pipelines, Δ. If any structural dissimilarity exists ($\bar{\delta} > 0$), then the overall dissimilarity is set to the structural dissimilarity ($\Delta = \bar{\delta}$). However, if there are no structural differences ($\bar{\delta} = 0$), then a scaled version of the hyperparameter dissimilarity is used.

Because the structural dissimilarity counts the minimum number of mutation operations required to transform one pipeline into another, $\bar{\delta}$ will always take a positive integer value. In general, the maximum hyperparameter dissimilarity between two pipelines occurs at the extremes of the hypercube, e.g., $\hat{t} = (0, 0, \ldots, 0)$, $\hat{u} = (1, 1, \ldots, 1)$. In this case, $\hat{\delta}(t, u) = \sqrt{n}$, where n is the number of hyperparameters under consideration, is the geometric length of the main diagonal of the hypercube, and is always greater than 1 for $n > 1$. This creates potential confusion as to whether a value of $\Delta > 1$ indicates that the change is hyperparameter-based, or structural. To address this ambiguity between hyperparameter and structural changes, a scaling factor can be introduced to adjust the hyperparameter dissimilarity relative to the number of hyperparameters under consideration:

$$\alpha = \left(1 - \frac{1}{n+1}\right).$$

Multiplying the hyperparameter dissimilarity by this term ensures the scaled hyperparameter dissimilarity will always satisfy $0 \leq \alpha \cdot \hat{\delta} < 1$. This means that the hierarchical dissimilarity metric explicitly indicates a structural transformation when $\Delta(p, q) \geq 1$, and explicitly indicates a hyperparameter transformation when $\Delta(p, q) < 1$.

Therefore, given two unique pipelines p, q, the formula for the hierarchical pipeline dissimilarity metric can be expressed as the piecewise function,

$$\Delta(p, q) = \begin{cases} \bar{\delta}(p, q) & \text{if } \bar{\delta}(p, q) > 0 \\ \alpha \cdot \hat{\delta}(p, q) & \text{if } \bar{\delta}(p, q) = 0. \end{cases}$$

4 Visual Representations of TPOT Search

Having established a metric for comparing pipelines in the decision space, it can be used to create a visual representation of the pipelines explored by TPOT throughout its search process, in the decision space. Experiments on the *abalone* [4] dataset are used to demonstrate these ideas.

4.1 Experimental Design

A modified version of TPOT v0.11.7 was employed for the experiments. The core algorithmic functionality of TPOT was retained; however, modifications crucial

for enabling deeper insights into the evolutionary process were implemented. These adjustments allowed for the tracking of several key properties throughout the search process, such as the composition of the selected parent population for each generation, and the specific details of each mutation operation.

The default TPOT values for crossover and mutation rates (0.1 and 0.9, respectively) were used, along with the default population size and number of generations, 100. Additionally, parallel experiments with a reduced population size of 10 were conducted, with the aim of improving the readability, and interpretability, of the visual representations. The default time-out per evaluation of 5 min was used for all experiments, evaluated using mean-squared error and five-fold cross-validation error.

Moreover, the impact of altering the selection mechanism was explored. Typically, TPOT employs a multi-objective selection process, based on cross-validation (CV) error and pipeline complexity—i.e., number of operators in the pipeline. In these experiments, a variation was introduced where this was replaced by a single-objective criterion, based purely on CV performance. This modification allowed for the comparison of the evolutionary trajectories of TPOT search, under different selection pressures.

Finally, the search data was collected using the *abalone* dataset [4]. This choice was influenced by limitations in space and the intent to demonstrate techniques rather than perform an empirical evaluation. The *abalone* dataset, a well-studied regression benchmark dataset, comprises 4,177 data points, each with eight features including categorical, integer, and floating-point types. To ensure consistency, all examples in this section were initialized with the same random seed. Yet, the observations were consistent across multiple experiments with different initial seeds. Further examples, including those from experiments with randomly generated and synthetically produced data exhibiting a strong linear relationship, are provided in the supplementary materials, offering additional perspectives and validation of the techniques presented.

4.2 Results and Discussion

Figure 3 provides information about the pipeline structures explored during one such search execution. The top plot in this figure illustrates the distribution and quality of evaluated pipelines, by structure. The x axis gives the index for each unique structure, in the order it was explored during the search, with the number of unique pipelines for each structure on the y axis. Each point in the plot represents a unique pipeline, with colour indicating its evaluated CV error, sorted from worst at the bottom, to best at the top. To improve distinction in the CV ranges of interest, the colour map is cut off at the 75th percentile value. The bottom plot in this figure tracks the composition of the selected parent population, for each generation, over the course of the search. The x axis gives the structure indices, corresponding to the values in the top plot. The colour of each point in this plot represents the number of pipelines from each structure, selected in each generation, given on the y axis. Both plots have a red triangle indicating the structure that produced the pipeline with the lowest CV error.

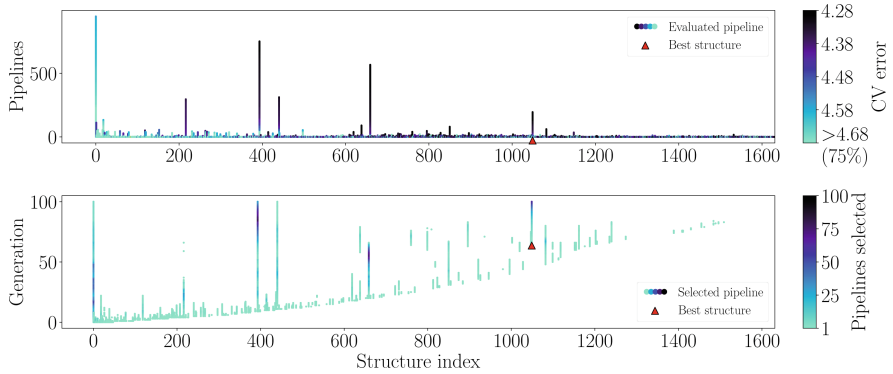

Fig. 3. Pipelines per structure (top) and selected pipeline structures per generation (bottom), for default population selection mechanism.

The frequency plot in Fig. 3 suggests that although TPOT is exploring over 1600 unique structures, its search is focused on 5 or 6 main ones. The most frequently evaluated structure contains 948 unique pipelines, with the average being 6.05 and the median being 1. This is supported by the population tracking plot underneath, where the spikes in the frequency plot correspond to structures which were selected from heavily at some stage in the search.

Performing a non-dominated sort on the evaluated pipelines enables TPOT to maintain control over the complexity of the pipelines. Minimizing both CV error, and number of operators, when selecting the parent population for each generation means that very complex pipelines are only selected when they are also high-performing. This is important, as "bloat" is a well-documented phenomena in many GP-based algorithms [12]; also, highly complex pipelines take longer to evaluate and are prone to over-fitting data [5,7]. However, one significant drawback of using this method to control pipeline complexity growth is exemplified by the largest structure in Fig. 3. Here, nearly 10% of the entire search budget was spent evaluating pipelines with this structure, but the best CV error that was achieved by any pipeline within it was around 4.56. The bracket representation for this pipeline structure is {`RandomForestRegressor{input_matrix}`}, which only contains a single operator and is therefore unlikely to ever be dominated, even when there are much better performing pipelines available. This can also be observed in the population tracking plot beneath, where this structure is selected from quite heavily for the first 50 or so generations, before fading out—but never totally disappearing—from the selected parent population.

Figure 4 provides the structure frequency and population tracking plots for a TPOT search where the selection pressure to minimize pipeline complexity has been removed, by choosing the parent population based on CV error alone. In these plots it can be seen that more than triple the number of structures were explored (4959 vs. 1630, in the previous example) when using this selection criteria. The budget is spread much more uniformly across the structures as

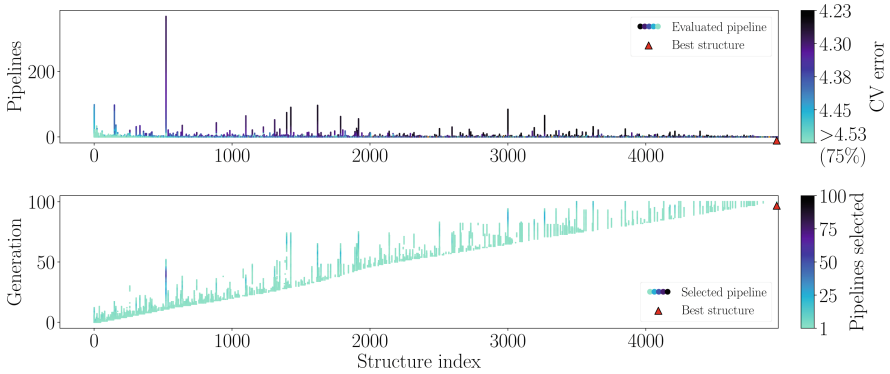

Fig. 4. Pipelines per structure (top) and selected pipeline structures per generation (bottom), for single objective population selection mechanism.

well, with the largest structure having 368 pipelines, and the average number of pipelines per structure being 2.0. The second example provided better quality pipelines over all as well, with the best CV error being 4.23—compared to 4.28 with multi-objective search—and the 75th percentile being 4.53—compared to 4.68. While employing a single objective approach to population selection does seem to yield pipelines with better CV errors, it also produced more complex pipelines, with the most complex pipeline comprising 20 operators—compared to a maximum complexity of 8 operators, in the previous example.

In order to gain further insight, the hierarchical pipeline dissimilarity metric as described in Sect. 3 can be used to create a visual representation of the evolutionary trajectory of the search, called a dissimilarity map. Figure 5 provides dissimilarity maps for both the default and single objective population selection mechanisms (larger versions are available as Figures S1 and S2 in the supplementary materials for this paper). Having partitioned the evaluated pipelines by structure, a pairwise structural dissimilarity ($\bar{\delta}$) matrix M is computed for each explored structure. The multi-dimensional scaling (MDS) algorithm [8] from the Scikit-learn Python package [1] is used to compute a 2D embedding of points, representing structures, such that the distances between the points preserve the values in M, as much as possible. The points are coloured based on the best CV error achieved for each structure, and their size is scaled relative to the number of pipelines each structure contains. The tracking data from the search is used to draw in directed connections between the structures, with the colour indicating how that structure was produced, and operations which transform a structure into an existing structure denoted with dashed lines. The result is a directed graph, which is acyclic when the dashed connections are removed. Inset into the corner is a subplot which provides a 2D embedding that preserves the pairwise hyperparameter dissimilarity ($\hat{\delta}$) for all the pipelines in the largest structure, coloured with respect to their CV errors. This serves as a reminder that each point in the dissimilarity map is representative of a set of pipelines.

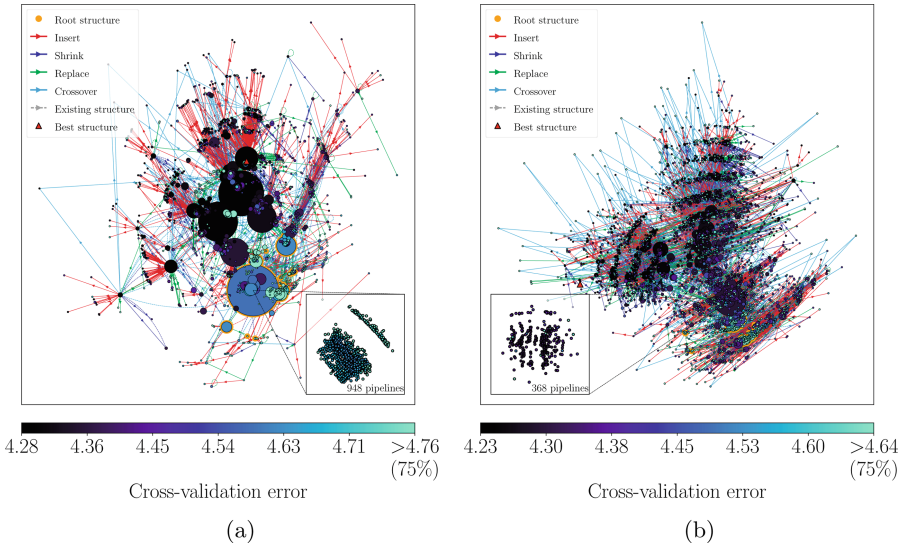

Fig. 5. Pipeline dissimilarity maps for default population selection mechanism (a) and single objective mechanism (b). Size indicates number of pipelines for each structure and colour indicates best CV. Inset illustrates hyperparameter dissimilarity and CV for all pipelines in largest structure.

The structures which comprise the initial population are highlighted with an orange outline, and a red triangle is used to indicate the structure containing the best pipeline over all.

The visual distinction between the pipeline dissimilarity maps in Fig. 5 supports observations made from Figs. 3 and 4. In Fig. 5(a), the default selection mechanism appears to focus on a few large structures in central locations. Conversely, Fig. 5(b) demonstrates that the single objective mechanism yields a greater number of structures, more evenly distributed in size. Notably, the individual transformation chains, defined as the longest, non-cyclic paths in the graph, vary in length, depending on the selection mechanism. Under the default mechanism, the longest transformation chain reaches 10 steps, with an average of 4.83 steps. In the single objective case, these numbers increase significantly, with the longest chain at 19 steps, and the average at 9.97.

These findings are consistent with the calculated pipeline dissimilarity metric. For the default mechanism, the maximum dissimilarity between any two pipelines is 11, while the single objective mechanism records a maximum dissimilarity of 22. However, it is important to recognize the limitations of the structural dissimilarity metric. This metric uses the tree edit distance to measure the shortest possible transformation chain between two pipelines, assuming a single transformation operation alters the pipeline complexity by at most one operator. Such an approach does not account for the pipeline complexity jumps achievable through crossover operations and the `CombineDFs` operator, as discussed in

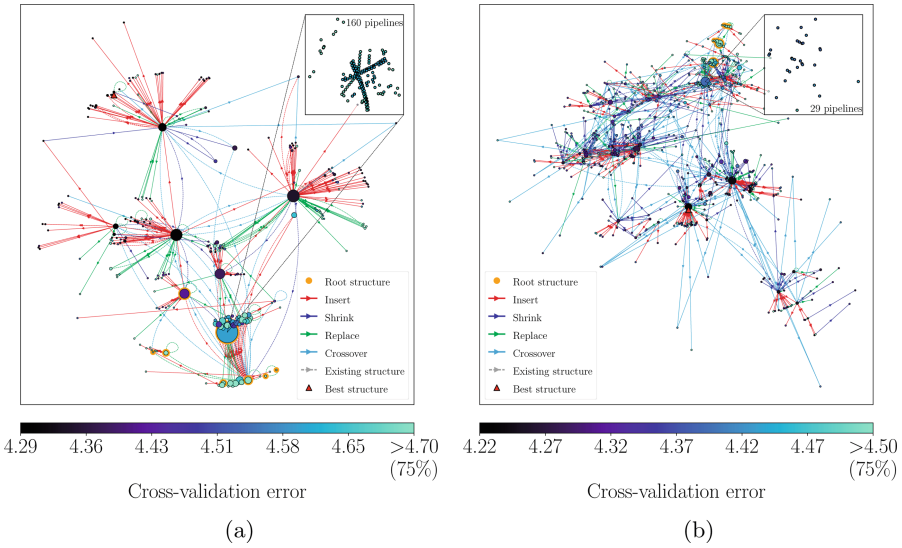

Fig. 6. Pipeline dissimilarity maps for default population selection mechanism (a) and single objective mechanism (b), with reduced population. Size indicates number of pipelines for each structure and colour indicates best CV. Inset illustrates hyperparameter dissimilarity and CV for all pipelines in largest structure.

Sect. 2. Consequently, the metric often overestimates the *actual* shortest transformation path. Since the metric must disregard the varied pipeline outcomes producible by crossover - contingent on the context of the other pipelines in the parent population - or by the `CombineDFs` operator, this overestimation becomes an intrinsic limitation of any metric that compares pipelines in isolation.

While the pipeline dissimilarity maps in Fig. 5 provide an intuitive global view of the evolutionary trajectory of TPOT search, there are so many elements, making it difficult to analyse the behaviour at a local level. To improve readability, Fig. 6 provides the pipeline dissimilarity maps for experiments conducted with a reduced population size of 10 (larger versions are available as Figures S7 and S8 in the supplementary materials). Similar patterns are observed in these experiments as for the full-size one. The default selection mechanism results in larger centrally-located structures, with shorter transformation chains. In the case of this reduced population experiment, the longest transformation chain produced by the default selection mechanism was 5 steps long, and the average was 3.3 steps, whereas the longest was still 19 for the single objective selection mechanism, with the average being 11.37. The dissimilarity metric correlated with this again, with the largest values for default and single objective selection mechanisms being 7 and 22, respectively.

Somewhat apparent in Fig. 5, but made much more clear in the reduced population size experiments, is the observation that a lot more unique structures were produced using crossover when the single objective selection mechanism is used,

compared to the default. This is likely because the parent populations had more diversity across generations when using this mechanism, so more opportunities exist to make new combinations of operators; whereas selecting a similar set of (and fewer) structures each generation is more likely to produce new pipelines from existing structures. It can also be observed in both sets of dissimilarity maps that some structures produced by crossover appear to only have a single parent structure. When determining the crossover points, TPOT finds all the operators shared by each parent, and then randomly selects one. In the case where the parent structure contains two instances of the selected operator, this can sometimes produce two unique crossover points—resulting in the production of a new structure.

5 Conclusion

This paper combined the concepts of structural and hyperparameter dissimilarity to produce a hierarchical metric, providing an intuitive reflection of evolutionary changes throughout TPOT search. This metric was found to effectively distinguish between structural transformations and hyperparameter optimizations, providing clearer insights into the decision space navigated by TPOT. The importance of considering pipeline architectures in a holistic manner, as opposed to focusing solely on individual component adjustments, was underlined by these findings.

Through experiments on the *abalone* dataset, it was observed that TPOT's search behavior is predominantly influenced by the exploration of different operator combinations, rather than just hyperparameter tweaking. The utilization of the metric in visual representations for tracing and interpreting the evolution of pipeline configurations was also presented, providing deeper insights into TPOT's search process. These observations were consistent with those of the developed metric, suggesting it to be an appropriate approximation of pipeline dissimilarity—with some limitations, as discussed in Sect. 4.

Building on the findings and methodologies established in this paper, several key areas for future work have been identified. An extensive fitness landscape analysis using the developed hierarchical metric could provide deeper insights into the nature of evolutionary machine learning pipeline optimization. Additionally, the creation of an interactive tool for visualizing TPOT's search process would significantly enhance the usability and interpretability of the findings. Such a tool could allow users to dynamically explore the evolutionary trajectories of machine learning pipelines, offering a more intuitive understanding of the search process and its outcomes.

References

1. Buitinck, L., et al.: API design for machine learning software: experiences from the scikit-learn project. In: ECML PKDD Workshop: Languages for Data Mining and Machine Learning, pp. 108–122 (2013)

2. De Rainville, F.M., Fortin, F.A., Gardner, M.A., Parizeau, M., Gagné, C.: DEAP: a Python framework for evolutionary algorithms. In: Proceedings of the 14th Annual Conference Companion on Genetic and Evolutionary Computation, pp. 85–92 (2012)
3. Garciarena, U., Santana, R., Mendiburu, A.: Analysis of the complexity of the automatic pipeline generation problem. In: 2018 IEEE Congress on Evolutionary Computation (CEC), pp. 1–8. IEEE (2018)
4. Gijsbers, P., et al.: AMLB: an AutoML benchmark. arXiv preprint arXiv:2207.12560 (2022)
5. Hastie, T., Tibshirani, R., Friedman, J.H., Friedman, J.H.: The Elements of Statistical Learning. Data Mining, Inference, and Prediction, vol. 2. Springer, New York (2009). https://doi.org/10.1007/978-0-387-84858-7
6. Hutter, F., Kotthoff, L., Vanschoren, J.: Automated Machine Learning. Methods, Systems, Challenges. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-05318-5
7. Kenny, A., Ray, T., Limmer, S., Singh, H.K., Rodemann, T., Olhofer, M.: Hybridizing TPOT with Bayesian optimization. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 502–510 (2023)
8. Kruskal, J.B.: Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. Psychometrika **29**(1), 1–27 (1964)
9. Müller, A.C., Guido, S.: Introduction to Machine Learning with Python: A Guide for Data Scientists. O'Reilly Media, Inc. (2016)
10. Olson, R.S., Bartley, N., Urbanowicz, R.J., Moore, J.H.: Evaluation of a tree-based pipeline optimization tool for automating data science. In: 2016 Proceedings of the Genetic and Evolutionary Computation Conference, pp. 485–492 (2016)
11. Pimenta, C.G., de Sá, A.G.C., Ochoa, G., Pappa, G.L.: Fitness landscape analysis of automated machine learning search spaces. In: Paquete, L., Zarges, C. (eds.) EvoCOP 2020. LNCS, vol. 12102, pp. 114–130. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-43680-3_8
12. Poli, R., Langdon, W.B., McPhee, N.F.: A Field Guide to Genetic Programming. Lulu Enterprises, UK Ltd. (2008)
13. Pushak, Y., Hoos, H.: AutoML loss landscapes. ACM Trans. Evol. Learn. **2**(3), 1–30 (2022)
14. Selkow, S.M.: The tree-to-tree editing problem. Inf. Process. Lett. **6**(6), 184–186 (1977)
15. Teixeira, M.C., Pappa, G.L.: Understanding AutoML search spaces with local optima networks. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 449–457 (2022)
16. Teixeira, M.C., Pappa, G.L.: On the effect of solution representation and neighborhood definition in AutoML fitness landscapes. In: Pérez Cáceres, L., Stützle, T. (eds.) Evolutionary Computation in Combinatorial Optimization. EvoCOP 2023. LNCS, vol. 13987, pp. 227–243. Springer, Cham (2023). https://doi.org/10.1007/978-3-031-30035-6_15
17. Teixeira, M.C., Pappa, G.L.: Fitness landscape analysis of TPOT using local optima network. In: Naldi, M.C., Bianchi, R.A.C. (eds.) Intelligent Systems, BRACIS 2023. LNCS, vol. 14197, pp. 65–79. Springer, Cham (2023). https://doi.org/10.1007/978-3-031-45392-2_5
18. Tong, H., Minku, L.L., Menzel, S., Sendhoff, B., Yao, X.: What makes the dynamic capacitated arc routing problem hard to solve: insights from fitness landscape analysis. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 305–313 (2022)