







Fuzzy Pattern Trees for Classification Problems Using Genetic Programming

Allan de Lima¹(✉) , Samuel Carvalho² , Douglas Mota Dias¹ ,
Jorge Amaral³ , Joseph P. Sullivan² , and Conor Ryan¹(✉) 

¹ University of Limerick, Limerick, Ireland

{Allan.Delima, Conor.Ryan}@ul.ie

² Technological University of the Shannon: Midlands Midwest, Limerick, Ireland

³ Rio de Janeiro State University, Rio de Janeiro, Brazil

Abstract. Fuzzy Pattern Trees (FPTs) are tree-based structures in which the internal nodes are fuzzy operators, and the leaves are fuzzy features. This work uses Genetic Programming (GP) to evolve FPTs and assesses their performance on 20 benchmark classification problems. The results show improved accuracy for most of the problems in comparison with previous works using different approaches. Furthermore, we experiment using Lexicase Selection with FPTs and demonstrate that selection methods based on aggregate fitness, such as Tournament Selection, produce more accurate models before analysing why this is the case. We also propose new parsimony pressure methods embedded in Lexicase Selection, and analyse their ability to reduce the size of the solutions. The results show that for most problems, at least one method could reduce the size significantly while keeping a similar accuracy. We also introduce a new fuzzification scheme for categorical features with too many categories by using target encoding followed by the same scheme for numerical features, which is straightforward to implement, and avoids a much higher increase in the number of fuzzy features.

Keywords: Fuzzy Pattern Trees · Genetic Programming · Bloat control · Lexicase Selection.

1 Introduction

As Artificial Intelligence (AI) continues to gain popularity by introducing many applications poised to become indispensable for humankind, the need to include explainable or interpretable traits in AI-based models has also grown. This is particularly crucial in sensitive fields such as healthcare, law and manufacturing. In addition, AI models' explanations can expand human experts' knowledge with completely new insights [21].

Fuzzy set theory can combine fuzzy terms and input features to separate the original set of features, representing them in a more comprehensible form [3]. Consequently, it can contribute to developing more interpretable solutions. Fuzzy

Pattern Trees (FPTs) extend those ideas for tree-based structures, where fuzzy operators are internal nodes, while fuzzy features are leaves [15,38]. The interpretability of FPTs was confirmed by healthcare experts in works using a heart disease dataset [27] and a sarcoidosis dataset [5].

Given the symbolic structure of GP, the solutions it evolves inherently possess a degree of interpretability, which can be further enhanced by factors such as the length of the expressions, the simplicity of the operators involved, the representation of the features, etc. For datasets of multiple types, Strongly Typed GP [23] improved the interpretability of the solutions by imposing structural constraints that prevent nonsensical operations, for instance, arithmetic operations between Boolean features. The evolution of FPTs facilitates this process because all features are represented as fuzzy features and can be used with the same sort of operators.

The work in [18] used Multi-Gene GP [9] to generate Fuzzy Inference Systems, but the first work [6] to go beyond traditional fuzzy-based rules used Cartesian GP [22] to evolve actual FPTs. Subsequently, that work was extended using a multi-objective approach, with accuracy and interpretability as objectives [29], highlighting the potential use of FPTs to bring more interpretability to the solutions. Later, Grammatical Evolution (GE) [30] was successfully used to evolve FPTs in a series of works [25,27,28,26], improving the accuracy over previous works using CGP, and contributing to the interpretability by including human knowledge [27,28]. Koza-style GP has also been used to evolve FPTs [5]. However, that work restricted their analysis to a single dataset and also represented their solutions with a single FPT using a threshold of 0.5 to discriminate between classes and, therefore, can only be used for binary classification problems.

In this work, we use GP in a generalised approach that can be applied to multi-class classification problems. Instead of using Multi-Gene GP to represent solutions as a set of tree structures, we adopt the method used in GE-based FPT research. This method represents solutions with a single tree where the root node determines the predicted class based on the highest value among its branches, with each branch corresponding to a different class. Furthermore, to improve the interpretability of FPTs, we evaluate different parsimony pressure methods to reduce the size of the solutions.

2 Background

2.1 Fuzzy Pattern Trees

FPTs are tree-like structures where internal nodes are logical and arithmetic operators constrained to yield results between 0 and 1. The leaves consist of fuzzy features, which are input features associated with fuzzy terms, valued according to the degree of membership of the input in each fuzzy set, in a process known as fuzzification [15,38]. Initially, FPTs were induced using a bottom-up approach, but later, a top-down algorithm was proposed [32]. This technique was successfully applied to classification [33] and regression problems [31].

Each FPT can be understood as a class description, which intrinsically contributes to interpretability. As a result, a solution consists of an ensemble of

FPTs. In this way, when using tree-based algorithms to evolve FPTs, one could consider using multi-tree algorithms. However, the first GE-based work on FPTs [25] introduced a more straightforward approach, where the root node of an FPT was defined as Winner Takes All (WTA). This node has the same number of branches as classes in the respective problem, each branch representing an FPT itself. It predicts the class according to the highest score FPT, performing, therefore, the same action as an ensemble of FPTs. It is worth highlighting this root node is always ignored for genetic operations such as mutation.

Figure 1 presents an example of an FPT, where the input features include Proline, Color-intensity, Hue, Malic-acid, and Flavonoids. The unary operator concentrator returns the square of its operand. Since the domain of fuzzy values is between 0 and 1, this operation will reduce the value, which justifies its name. On the other hand, the operator OWA (Ordered Weighted Average) has two operands and also a weight and calculates the average between its operands, giving the weight to the biggest operand and the complement of the weight to the smallest operand. Each feature is associated with a fuzzy term that represents a specific range within the domain of that feature. For example, Proline_Low will have a high value if the value of the input feature is low. In this case, we are using the terms ‘low’, ‘moderate’ and ‘high’, but we could use more specific terms, such as ‘cold’, ‘moderate’ and ‘hot’ for an input feature related to temperature, for example. This FPT models a solution for the wine problem, which consists of classifying the samples into three types of wine, labelled as ‘A’, ‘B’, and ‘C’; the expression in the left branch of the node WTA3 (Winner Takes All with three branches) represents the classification to class A, the central branch to class B, and the right branch to class C. The first branch uses only the feature Proline_Low to classify class A, and it will result in a low value if Proline_Low is high and a medium value if it is low because its complement is concentrated. In this way, we could say that if the input feature Proline presents a low value, the class is definitely not A. However, if it presents high values, the discrimination to class A will depend on the values of other branches being smaller than the first. The second branch is easier to understand because it has no operations and uses the feature Color-intensity_Low to classify class B. In this way, if the input feature Color-intensity presents low values, the sample is more likely to be class B. On the other hand, the third branch uses three features to classify class C:

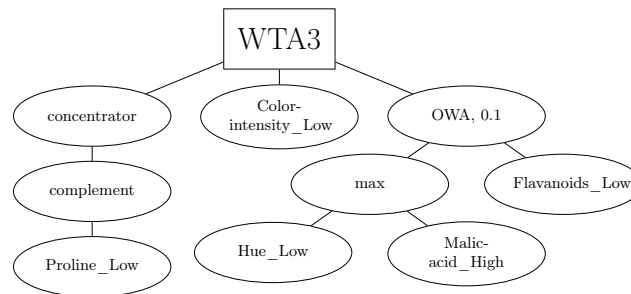


Fig. 1: Example of an FPT.

Hue_Low, Malic-acid_High and Flavanoids_Low, and it will result in a high value if Flavanoids_Low is high and at least one of the others is also high. This is because the node OWA will calculate the average, giving weight 0.1 to the maximum branch and 0.9 to the minimum branch, resulting in a high value only if both branches have high values.

2.2 Lexicase Selection

Lexicase Selection is a parent selection method, originally developed to solve modal [35] and uncompromising problems [13], which has been successfully applied to different domains [19,24,2]. This method selects parents based on the performance of individuals on random permutations of the training cases taken separately instead of on an aggregate fitness score. Briefly, each selection process starts by initialising a pool of candidates with the entire population of individuals. The training cases are considered in random order, and in each step, only the candidates with the best score for the current training case are maintained in the pool. This process continues until a single candidate remains in the pool, which is then selected. Since only the best solutions for the considered training case are kept in the pool, the process can be viewed as the selection of specialists [12]. Furthermore, specialists on distinct subsets of training cases are selected as offspring since the ordering of training cases differs for each selection process. Consequently, this approach keeps high diversity [11] and promotes a better exploration of the search space [14].

Lexicase performs very well in discrete error domains, but its selection pressure is too high when employed in continuous domains, such as for regression problems, since a single training case is likely to filter most of the candidates. To overcome this, ϵ -lexicase [19] redefined the pass condition for candidates on each filtering step by defining a threshold, ϵ , as a new parameter. Nonetheless, since its original approach, an automatic ϵ , notably the Median Absolute Deviation (MAD), presented better results.

Another approach, batch-Lexicase Selection, follows the same ideas as the original Lexicase, but each filtering step considers the aggregated fitness score on a subset of training cases, according to a new parameter, the *batch size*. This approach improved generalisation on classification problems [2], and it was subsequently assessed on the domain of program synthesis problems [34].

3 Methodology

The fuzzification scheme in Figure 2a defines triangular fuzzy membership functions that were used to fuzzify each numerical feature into three fuzzy features. Considering, for example, the input feature X , the respective fuzzy features are X_low , X_medium and X_high . Given the value $X_{example}$, the respective pertinence values are μ_{low} in X_low , μ_{medium} in X_medium and 0 in X_high . On the other hand, for categorical features, we used the singleton membership function, as defined in Equation 1 and Figure 2b. Here, x is a categorical feature, cat_i

is one of the N possible categories of that feature, and $\mu_i(x)$ is the pertinence value of a sample in a fuzzy feature that we refer to as x_i . This fuzzification scheme is analogous to one-hot encoding for categorical features.

$$\mu_i(x) \equiv \begin{cases} 1 & \text{if } x = \text{cat}_i \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

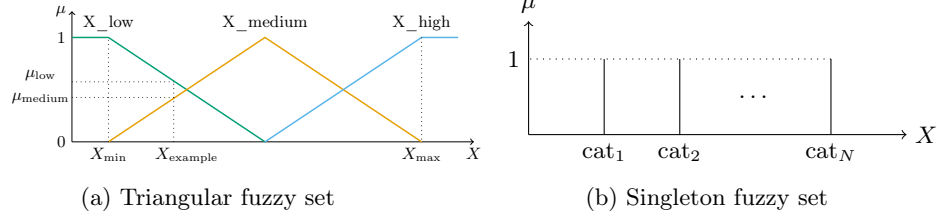


Fig. 2: Fuzzification schemes

Similarly, we use one-hot encoding to convert the target outputs into a fuzzy format, in which each sample is represented by a number of fuzzy outputs corresponding to the number of classes in the given problem. For instance, in a problem with three classes – ‘A’, ‘B’ and ‘C’ – ‘A’ is represented with 100, ‘B’ with 010 and ‘C’ with 001, each bit being considered a fuzzy output. In this example, the root node WTA3 predicts the class ‘A’, ‘B’ or ‘C’ based on the largest output value from the first, second, or third branch, respectively. However, we use Root Mean Squared Error (RMSE) as a fitness function between the expected fuzzy outputs and the actual values coming from each branch. This enhances the fitness score with additional information, providing more guidance during the evolutionary process. For our previous example, consider a sample with output ‘A’. An individual with branch values equal to 0.6, 0.55 and 0.5 will predict class ‘A’ correctly, as would another with branch values equal to 0.9, 0.2 and 0.1. Still, the latter solution is preferable, and RMSE will reflect that. Equation 2 defines RMSE as we use in our work, where BV_{ij} is the branch value evaluated in the j -th branch for the i -th sample, while FO_{ij} is the fuzzy output referred to the j -th class (encoded using one-hot) for the i -th sample.

$$\text{RMSE} = \sqrt{\frac{1}{n_{\text{samples}}} \sum_{i=1}^{n_{\text{samples}}} \left(\frac{1}{n_{\text{classes}}} \sum_{j=1}^{n_{\text{classes}}} (BV_{ij} - OH_{ij})^2 \right)} \quad (2)$$

We use Equation 2 to calculate an aggregated fitness score from all samples, which is then used with tournament selection. However, since we also run experiments with Lexicase-based approaches, we need to define a fitness case function. We tried some preliminary experiments using the equivalent RMSE as a fitness case function, and noticed the evolution was quite poor. This happened because the distance between small errors is bigger using RMSE than MSE, and since Lexicase considers the fitness cases separately, it increases the pressure too much in the filtering steps. Equation 3 defines MSE as we use in our work, where, for a given sample, BV_i is the branch value evaluated in the i -th branch, while FO_i is the fuzzy output referred to the i -th class.

$$\text{MSE} = \frac{1}{n_{\text{classes}}} \sum_{i=1}^{n_{\text{classes}}} (\text{BV}_i - \text{OH}_i)^2 \quad (3)$$

In addition to promoting accuracy, we also wish to encourage small solutions to aid interpretation. GP implementations usually use the maximum depth as a parameter. However, setting an optimal value for this parameter is too difficult since it is problem-dependent, while setting a too-small value could hamper the evolution by over-constraining the search space. Therefore, we set a high value for this parameter to give the algorithm freedom to explore the search space. Furthermore, we aim to analyse the performance of different parsimony methods in reducing size and setting the maximum depth with a low value to force the creation of small individuals would compromise our analysis.

The most common method to add parsimony pressure to the selection process involves penalising solutions in proportion to their size, introducing a penalty ratio as a new parameter. Another option is lexicographic parsimony pressure [20], which consists of preferring smaller individuals when there is a tie in the fitness score. Lexi² [4] applies this approach to Lexicase Selection [35]. Given the nature of our fitness evaluations, it is anticipated that each individual will possess unique fitness case scores. Consequently, employing Lexicase Selection could allow the selection of a parent based on a single sample, resulting in excessively high selection pressure. Thus, using epsilon-lexicase, which introduces a threshold of tolerance and applying lexicographic parsimony pressure to epsilon-lexicase, implementing epsilon-lexi² appears to be a viable alternative, even though we are tackling classification problems.

Following the recommendations of the original work for epsilon-lexicase, we use MAD as epsilon for epsilon-lexi². In addition, the original work also recommends calculating MAD once per generation instead of at each filtering step because the difference in the results is not significant, and it saves computational costs. In this way, MAD is computed for each sample across all individuals at the outset. Subsequently, imagine we have a vector for each sample containing the fitness case score for every individual with respect to that sample. We identify the minimum value of that vector as the best fitness case score for that sample. Given the corresponding MAD, we can replace a value in that vector with 0 if it is smaller or equal to (best + MAD) and with 1 otherwise.

After processing each sample in this manner, we obtain a vector of discrete fitness cases to represent each individual. Then, we initialise the pool of candidates using only individuals with unique error vectors. This prefiltering step reduces the time spent in the selection process by avoiding redundant iterations when filtering the pool of candidates, and it does not affect the outcome [10,12]. During this step, we also apply lexicographic parsimony pressure by selecting the smallest individual in cases where multiple individuals share an identical error vector. In terms of the individual selected, this has the same effect as if we have not prefiltered the pool and chosen the smallest individual after filtering the pool with all fitness cases.

1. **Initialise:**
 - (a) Place all individuals in a pool of **candidates**
 - (b) Create batches by taking different training cases in random order and list them in **batch_cases**. All batches have **batch_size** cases, except the last one, which has the remaining cases
 - (c) Calculate the aggregated fitness for each batch by averaging the fitness values of the respective samples
 - (d) For each batch, calculate MAD and identify the best fitness over all individuals
 - (e) For each batch in each individual, change the value to 0 if it is smaller or equal to the respective best value plus the respective MAD and to 1 otherwise, in order to create for each individual an error vector with discrete values
 - (f) Keep in the pool of **candidates** only individuals with unique error vectors
 - i. When individuals with the same error vector are found, keep the one with the smallest number of nodes
 - ii. If there is a tie within the number of nodes, keep a random individual
2. **Loop:**
 - (a) Replace **candidates** with the individuals currently in **candidates**, which presented the best fitness for the first batch in **batch_cases**
 - (b) If a single individual remains in **candidates**, return this individual
 - (c) Else eliminate the first element in **batch_cases** and re-run the **Loop**

Listing 1: Algorithm for selecting one individual with batch-epsilon-lexi².

In addition, we also used the epsilon as a threshold and applied lexicographic parsimony pressure to an approach based on batch-Lexicase, which we refer to as batch-epsilon-lexi². Listing 1 details the algorithm for selecting one individual with this approach. We begin by populating the batches with randomly selected cases. Next, we calculate the fitness batch scores based on adding the respective fitness case scores for each individual. Subsequently, we calculate the MAD and identify the best score for each batches. After that, we can replace the continuous values with 0 or 1 and prefilter individuals with the same error vectors, preferring the smallest ones. Finally, the **Loop** stage is similar to standard Lexicase Selection. This approach is more expensive than epsilon-lexi², because we need to calculate MAD every time for the event of selecting a parent instead of once per generation. On the other hand, the computational cost in the filtering process is reduced because the bigger the batch size, the smaller the number of filtering steps, as we show in our results. However, overall, this approach is still more expensive.

In summary, the scenarios we analyse in this work are as follows:

- tournament: this is our baseline approach, where we use tournament selection to select parents based on the RMSE;
- tournament with penalised fitness: in this scenario, we also use tournament selection, but the fitness score is the RMSE plus the number of nodes divided by 500. We use the number of nodes as size measurement as this gives more diverse values than depth. In relation to the penalty factor, we experimented

with various values. Preliminary results indicated that a factor of 500 effectively reduces the size of the solutions without adversely affecting the error rate;

- epsilon-lexi²: we use epsilon-lexicase to select parents based on the MSE for each sample and apply lexicographic parsimony pressure by using the number of nodes as a tie-breaking criterion to prefer the smallest individual;
- batch-epsilon-lexi² (batch size 2): we use batch-epsilon-lexicase to select parents based on the MSE for each batch of samples in this and in the remaining scenarios. We performed experiments in a range of small batch sizes, because for bigger sizes, the selection pressure is usually much smaller [34]. In addition, we apply lexicographic parsimony pressure by using the number of nodes as a tie-breaking criterion to prefer the smallest individual;
- batch-epsilon-lexi² (batch size 5);
- batch-epsilon-lexi² (batch size 10);
- batch-epsilon-lexi² (batch size 20).

4 Experimental Setup

We used Python 3.10.8 and DEAP 1.4 [8] to run our experiments, seeding each run with `random.seed(n)`, where `n` is an integer number in the interval `[1, 30]`, to replicate our results. All code is available in our repository⁴.

Table 1 provides a summary of the classification problems examined in this work. Some of them are “blacklisted” [36], but we still assessed the performance of our implementation with these datasets to compare our results with previous works using FPTs. In addition, we included some datasets from the CHIRP suite [37], following the recommendation of using it as an alternative to the “blacklisted” problems [36]. Consequently, our selection encompasses a diverse array of problems, varying in the number of classes, class balance, sample size, feature types, and so on. All datasets are accessible via the links provided in the references. In instances where the data were pre-partitioned into training and test sets, we adhered to this division and have indicated the respective sizes in the table. Conversely, for datasets that were not pre-divided, we used 75% for training and 25% for testing, performing a fresh split for each run. For these cases, the table displays the total size of the dataset. Additionally, we have detailed the number of numerical and categorical features, as well as the total number of features after fuzzifying the data.

We used the schemes detailed in Figure 2 to fuzzify features. However, for categorical features with too many different categories, that scheme would create an impractical number of fuzzy features, so we propose in this work using target encoding followed by the scheme for numerical features. All fuzzification

⁴ <https://github.com/bdsul/grape/tree/main/GP/ClassificationFPT>

Table 1: Characteristics of datasets, taken from the UCI repository [7], the CMU repository [17] and the ProPublica data store [1].

	Classes	Training	Testing	Numerical features	Categorical features	Features after Fuzzification
australian	2	688		8	6	32
adult	2	32561	16281	5	9	40
bankMarketing	2	41188		9	11	58
credit	2	690		6	9	37
germanCredit	2	1000		7	13	73
haberman	2	306		3	-	9
heartDisease	2	297		5	8	35
horse	3	300	68	2	18	58
iris	3	150		4	-	12
lawsuit	2	264		3	1	10
lupus	2	87		3	-	9
pima	2	768		8	-	24
recidivism	2	6172		4	5	25
satellite	6	4435	2000	36	-	108
segment	7	210	2100	16	2	54
spect	2	80	187	-	22	22
transfusion	2	748		4	-	12
vehicle	3	846		18	-	54
violentRecidivism	2	4743		5	4	23
wine	3	178		13	-	39

thresholds were defined based on the training set to avoid bias towards the test set. For example, in Figure 2a, X_{\min} and X_{\max} are set according to the training set. If the respective feature in the test set presents a value smaller than X_{\min} or greater than X_{\max} , its fuzzy pertinence value will be 0 or 1, respectively. Similarly, target encoding followed only the values from the training set to encode both the training and test sets.

Table 2: Experimental setup.

(a) Model parameters.		(b) Fuzzy operators used in the function set. WA means Weighted Average, while OWA means Ordered Weighted Average.	
Parameter type	Parameter value	Operator	Equivalent expression
Number of runs	30	$\max(a, b)$	-
Number of generations	50	$\min(a, b)$	-
Population size	500	$WA(a, b, r)$	$r \times a + (1 - r) \times b$
Maximum depth	17	$OWA(a, b, r)$	$r \times \max(a, b) + (1 - r) \times \min(a, b)$
Mutation probability	0.05	dilator(a)	\sqrt{a}
Crossover probability	0.8	concentrator(a)	a^2
Initialisation method	Ramped half-and-half	complement(a)	$1 - a$
Minimum initial depth	2		
Maximum initial depth	6		
Tournament size	7		
epsilon	MAD		
Lexi ² criterion	Number of nodes		

Table 2a summarises the parameters used in our experiments, while Table 2b describes the meaning of the fuzzy operators used in our function set. Regarding the operators WA and OWA, the operand r is constrained to use a constant from the set $\{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9\}$.

5 Results and Discussion

Table 3 presents the mean accuracy obtained on the test set across 30 trials of our work employing tournament selection, alongside results from other studies using CGP [6] and GE [25,27] for the works we have problems in common, where we used the same population size and number of generations. The results, with superior accuracies denoted in bold, indicate that the GP approach yielded the highest accuracy in 10 out of 14 problems.

Table 4 provides a detailed summary of our study’s outcomes, displaying for each dataset the mean accuracy on the test set alongside the average node count of the optimal individual.

We conducted a statistical analysis to evaluate the significance of these findings. The Shapiro-Wilk test was applied to both the accuracy and the number of nodes to test for normality, with a p-value threshold set at 0.05. To compare the performance of various approaches against the baseline employing tournament selection, we utilised the Wilcoxon signed-rank test for the non-Gaussian results, and the Student’s t-test for the Gaussian results. However, to account for the multiple comparisons being made, we applied a Bonferroni correction, adjusting the significance level by a factor of 6. Consequently, we adopted a more stringent p-value threshold of 0.0083, resulting from the division of 0.05 by 6, to reject the null hypothesis (no difference between the metrics).

To conserve space, we have omitted the explicit reporting of calculated p-values. Instead, we employed the symbols +, -, = to denote whether the results, in comparison to tournament selection, are significantly better, worse, or not significantly different, respectively. Consider that for accuracy, being better means it increased, while for size, being better means it decreased. We noted that, with the exception of the satellite dataset, every problem exhibited at least one approach in which accuracy was maintained without significant reduction, while the application of parsimony pressure methods resulted in a substantial decrease in size.

Table 3: Mean of the accuracy computed on the test set over 30 runs using plain tournament with GP compared to the results using CGP [6] and GE [25,27], where the best results are in bold.

	GE	CGP	GP
adult	0.81	-	0.83
australian	0.86	0.85	0.85
bankMarketing	0.89	-	0.90
germanCredit	0.71	-	0.72
haberman	0.74	0.73	0.74
heartDisease	0.79	-	0.83
iris	0.96	0.95	0.96
lawsuit	0.96	0.93	0.95
lupus	0.73	0.74	0.75
pima	0.74	0.72	0.76
recidivism	0.72	-	0.68
transfusion	0.77	0.76	0.78
violentRecidivism	0.83	-	0.81
wine	0.83	0.9	0.91

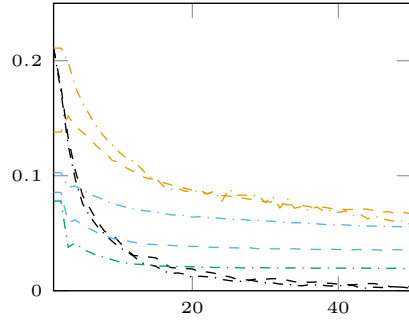
Table 4: Mean (standard deviation) of the accuracy computed on the test set and the number of nodes of the best individual. Both were calculated over 30 runs with the respective method. TPP refers to tournament with parsimony pressure, while B- ϵ -L² refers to batch-epsilon-lexi², where the number between parenthesis is the batch size. The first row for each dataset refers to the accuracy, and the second row refers to the size. The symbols +, -, = indicate, for each method other than tournament, whether the results in comparison to tournament are, respectively, significantly better, worse, or not significantly different. The last row indicates how many datasets the respective method presented results, which are not significantly worse than tournament in accuracy and significantly better in size.

	tournament	TPP	ϵ -lexi ²	B- ϵ -L ² (2)	B- ϵ -L ² (5)	B- ϵ -L ² (10)	B- ϵ -L ² (20)
adult	0.83(0.01)	0.82(0.01)-	0.82(0.00)-	0.82(0.00)-	0.83(0.00)=	0.83(0.00)-	0.83(0.00)=
	100.8(44.3)	55.0(32.9)+	39.2(28.8)+	35.2(12.7)+	51.0(25.8)+	61.6(33.3)+	87.1(33.9)=
australian	0.85(0.02)	0.85(0.02)=	0.85(0.02)=	0.86(0.02)=	0.86(0.02)=	0.86(0.02)=	0.86(0.02)=
	120.0(67.1)	59.4(32.5)+	72.3(34.1)+	53.0(22.8)+	76.2(24.3)+	86.3(23.9)+	108.9(38.3)=
bankMarketing	0.90(0.00)	0.90(0.00)-	0.89(0.00)-	0.89(0.00)-	0.90(0.00)=	0.90(0.00)=	0.90(0.00)=
	72.2(36.4)	39.2(21.5)+	18.8(11.0)+	17.6(9.9)+	28.2(12.9)+	48.3(18.5)+	58.3(22.0)=
credit	0.86(0.03)	0.86(0.03)=	0.86(0.03)=	0.85(0.03)=	0.86(0.03)=	0.85(0.03)=	0.86(0.02)=
	96.5(48.1)	67.9(35.3)=	56.5(24.8)+	53.0(23.8)+	72.6(22.3)=	79.4(30.4)=	95.2(36.0)=
germanCredit	0.72(0.03)	0.70(0.03)=	0.70(0.03)=	0.70(0.03)=	0.71(0.03)=	0.72(0.03)=	0.72(0.03)=
	131.4(46.4)	97.8(34.0)+	45.5(22.8)+	42.3(17.1)+	58.9(35.2)+	85.1(39.8)+	113.6(38.6)=
haberman	0.74(0.05)	0.74(0.05)=	0.74(0.05)=	0.74(0.05)=	0.74(0.06)=	0.73(0.06)=	0.74(0.05)=
	139.8(49.1)	73.6(40.6)+	36.3(18.0)+	36.9(22.9)+	51.0(20.2)+	74.7(27.3)+	107.0(39.0)+
heartDisease	0.83(0.04)	0.81(0.04)=	0.79(0.05)-	0.81(0.04)=	0.81(0.04)=	0.82(0.03)=	0.82(0.04)=
	129.4(42.7)	88.8(36.7)+	48.8(23.6)+	72.5(30.1)+	99.8(35.8)+	102.4(34.9)=	125.0(46.6)=
horse	0.73(0.04)	0.72(0.03)=	0.70(0.03)=	0.72(0.04)=	0.72(0.02)=	0.74(0.04)=	0.73(0.05)=
	107.0(51.7)	80.3(42.5)=	35.0(14.5)+	40.3(16.7)+	45.7(17.2)+	74.9(33.5)+	93.4(34.1)=
iris	0.96(0.03)	0.95(0.03)=	0.95(0.03)=	0.96(0.03)=	0.95(0.03)=	0.95(0.03)=	0.95(0.03)=
	60.8(67.5)	14.1(5.4)+	21.0(16.3)+	26.9(19.3)+	33.4(19.4)=	27.0(17.8)=	23.7(24.3)+
lawsuit	0.95(0.02)	0.96(0.02)=	0.95(0.03)=	0.95(0.02)=	0.95(0.02)=	0.95(0.02)=	0.96(0.03)=
	109.1(56.1)	48.8(31.1)+	21.5(16.3)+	22.5(12.4)+	36.3(26.5)+	53.8(31.7)+	77.4(26.4)+
lupus	0.75(0.07)	0.74(0.07)=	0.74(0.06)=	0.74(0.07)=	0.73(0.07)=	0.74(0.08)=	0.74(0.08)=
	79.2(60.3)	47.0(30.4)+	39.8(24.4)+	38.8(27.8)+	59.4(26.0)=	72.1(24.5)=	94.6(43.4)=
pima	0.76(0.02)	0.75(0.02)=	0.76(0.03)=	0.75(0.02)=	0.76(0.03)=	0.76(0.02)=	0.76(0.03)=
	99.7(60.2)	49.7(32.0)+	26.6(10.9)+	37.2(22.3)+	41.2(17.2)+	57.5(22.0)+	74.3(18.2)=
recidivism	0.68(0.01)	0.67(0.01)=	0.66(0.01)-	0.67(0.01)-	0.67(0.01)-	0.68(0.01)=	0.68(0.01)=
	102.7(48.5)	90.6(38.7)=	43.1(24.9)+	33.1(16.3)+	48.9(26.2)+	57.9(23.6)+	72.1(27.3)+
satellite	0.69(0.03)	0.66(0.03)-	0.57(0.06)-	0.69(0.03)=	0.70(0.02)=	0.72(0.02)+	0.72(0.04)+
	48.3(26.3)	33.7(15.1)=	18.9(7.8)+	36.8(14.1)=	45.0(16.9)=	52.9(15.7)=	59.8(31.7)=
segment	0.73(0.05)	0.61(0.08)-	0.48(0.08)-	0.73(0.05)=	0.77(0.04)+	0.76(0.08)=	0.76(0.09)=
	44.3(16.6)	34.1(14.4)=	23.2(6.7)+	34.3(9.6)+	39.6(13.7)=	49.0(17.7)=	59.5(22.8)-
spect	0.77(0.04)	0.75(0.05)=	0.78(0.04)=	0.78(0.03)=	0.78(0.03)=	0.77(0.03)=	0.76(0.04)=
	165.3(46.5)	125.6(45.8)+	75.4(25.1)+	84.1(28.2)+	114.8(40.7)+	120.5(29.7)+	160.0(54.5)=
transfusion	0.78(0.03)	0.77(0.03)=	0.77(0.03)=	0.77(0.03)=	0.78(0.03)=	0.78(0.03)=	0.78(0.03)=
	121.3(58.6)	73.5(44.4)+	40.3(17.1)+	40.0(18.8)+	64.4(27.0)+	77.4(35.9)+	96.3(40.5)=
vehicle	0.69(0.05)	0.66(0.04)-	0.64(0.04)-	0.69(0.04)=	0.72(0.04)+	0.72(0.04)=	0.73(0.04)+
	59.3(30.1)	37.8(31.1)+	13.0(8.9)+	34.3(13.6)+	52.4(24.2)=	64.0(22.7)=	83.1(32.8)-
violentRecidivism	0.81(0.01)	0.80(0.01)=	0.80(0.01)=	0.80(0.01)=	0.80(0.01)=	0.80(0.01)=	0.81(0.01)=
	121.9(57.9)	76.3(46.6)+	31.6(26.9)+	25.2(14.7)+	46.1(22.1)+	53.7(27.3)+	69.2(28.4)+
wine	0.91(0.04)	0.91(0.05)=	0.90(0.05)=	0.91(0.05)=	0.93(0.04)=	0.92(0.05)=	0.92(0.04)=
	60.9(35.2)	24.7(15.3)+	40.8(16.2)+	55.0(18.9)=	60.1(24.6)=	64.8(31.8)=	57.5(37.0)=
Total		12	13	15	12	11	5

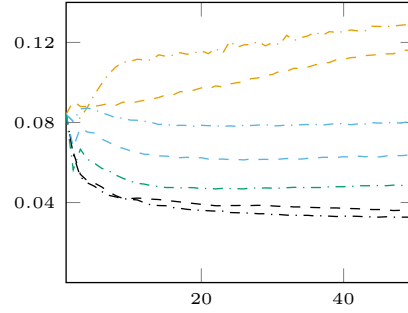
In addition, the final row of the table indicates the number of datasets for which each method yielded results that were not significantly inferior to tournament selection in terms of accuracy yet were significantly more efficient in terms of size reduction. We observed the most efficient method for reducing size without compromising accuracy was batch-epsilon-lexi² using a small batch size (2). Furthermore, this same method, utilising larger batch sizes (5, 10 and 20), was able to significantly enhance accuracy compared to tournament selection on a total of five occasions. Nonetheless, ultimately, the approach using tournament still provided the best accuracy for most of the problems. As discussed in Section 3, defining the aggregated fitness as the RMSE between the values from the ensemble of FPTs and the one-hot representation of the target outputs brings much information to the aggregated score, making tournament selection a highly competitive selection method.

Figure 3 presents some generational results for other measurements using the heart disease dataset, but similar observations can be made for other problems. All graphs are plotted over an average of 30 runs. Figure 3a shows the average MAD across generations. Firstly, MAD is taken for each sample across all individuals, and then this graph illustrates the average MAD over all samples. We also report these values for tournament-based approaches, even though it is not being used for those methods, in order to clarify what is happening during the evolution for all scenarios. Figure 3b shows the average variance of the fitness case values across generations. For this graph, we calculate the variance of the MSE values for every sample for all individuals. This measurement is not considered in the evolutionary process but is good for analysing the spread of fitness case values. Figure 3c depicts the average number of filtering steps with the Lexicase-based methods across generations. This measurement refers to the number of fitness case values used by the Lexicase loop to filter the pool of candidates, and it is used to confirm an assumption we made in Section 3, where we stated the bigger the batch size, the smaller the number of filtering steps. In Figure 3d, we can see the average behavioural diversity of the predictions across generations. After predicting classes with an individual, we define its behaviour as a vector with the classes predicted for every sample. In this way, behavioural diversity is the number of unique behaviours divided by the population [16]. Finally, in Figure 3e, we can see the average percentage of distinct individuals being selected across generations. For this measurement, we count how many unique individuals were selected to create the offspring for the next generation.

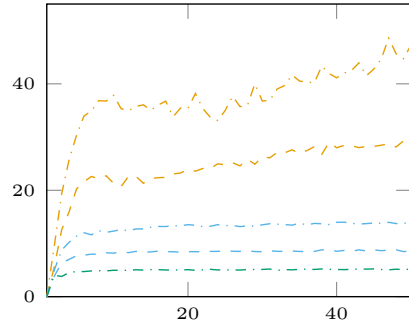
Assuming epsilon-lexi² as a batch-epsilon-lexi² approach with a batch of size 1, we highlight from Figure 3a that for Lexicase-based approaches, the format of the curves is similar. Other than batches of size 1 and 2, the larger the batch size the lower the convergence value of MAD. On the other hand, the tournament-based approaches presented a sharp decrease in the average MAD since early generations, converging to the smallest values in later generations. Considering tournament presented the best performance for most of the problems, one might intuitively think that a smaller epsilon would be helpful for the Lexicase-based approaches. We tested this with some preliminary experiments, which reduced



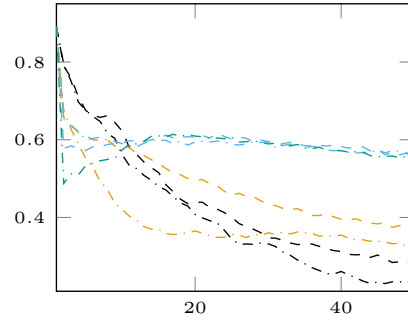
(a) Average MAD across generations.



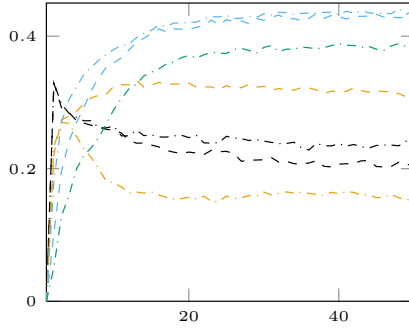
(b) Average variance across generations.



(c) Average number of filtering steps with the Lexicase-based methods across generations.



(d) Average behavioural diversity of the predictions across generations.



(e) Average percentage of distinct individuals being selected across generations.

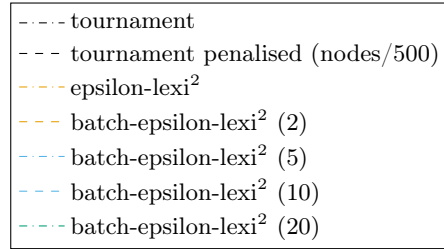


Fig. 3: Generational results for some measurements with the heart disease dataset.

the epsilon by a factor, but the results were relatively poor. In this way, we can assume the small MAD is a consequence of how the tournament approach guides the evolution using the given fitness function, and it is not necessarily the cause of its good performance. These observations are complemented by Figure 3b, where the tournament approaches presented the smallest variance, and by Figure 3d, where we can see that the tournament approaches converged to the lowest diversity.

Considering the low values for diversity and MAD using tournament in late generations, we can assume the individuals in these populations are more similar to each other than in the Lexicase-based approaches. In this way, since tournament exhibits the best performance, we can assume it is exploiting very well at this stage. Moreover, it may have explored the search space very well in early generations, considering that its decrease in diversity was delayed compared to Lexicase-based approaches. This observation is enforced by the fact that the peak of distinct individuals being selected by tournament is in early generations, as we can see in Figure 3e.

6 Conclusion

In this work, we evolved FPTs using GP following successful strategies developed in works with other evolutionary algorithms. The accuracy of the solutions improved for most of the datasets in common with previous works using similar computational costs (population size and number of generations). In addition, we proposed using Lexicase Selection to evolve FPTs, as well as a new fuzzification scheme for categorical features with too many categories by using target encoding followed by a scheme for numerical features.

Moreover, we evaluated the performance of the system across a broad range of datasets and proposed changes in well-known selection methods to include parsimony pressure strategies. For the majority of the problems investigated, we identified at least one method capable of significantly diminishing the complexity of the models without compromising their performance, which remained comparable to that achieved by tournament selection.

In addition, we showed that in terms of accuracy, plain tournament and batch-epsilon-lexi² resulted in the best performance, highlighting that for FPTs, selection methods based on aggregated fitness scores worked best on these problems.

Although the bottom-up approach to induce FPTs was already applied for regression problems, as we highlighted in Section 2, there is still a lack in the literature about using evolutionary algorithms for that. In future work, we plan to extend the strategies presented here to apply GP to synthesising FPTs for regression problems.

Representing input data using fuzzy logic usually brings more interpretability to the features since it separates the data into specific, often meaningful, parts of their domain, usually associated with a descriptive term. In this way, we also intend, in future work, to deeply analyse FPTs from the perspective of interpretability, since it has become an essential concern for Machine Learning (ML) models in critical areas such as healthcare, law and manufacturing industries.

References

1. Propublica data store, <https://www.propublica.org/datastore/dataset/compas-recidivism-risk-score-data-and-analysis>
2. Aenugu, S., Spector, L.: Lexicase selection in Learning Classifier Systems. In: Proceedings of the Genetic and Evolutionary Computation Conference. pp. 356–364 (Jul 2019). <https://doi.org/10.1145/3321707.3321828>
3. Cordon, O.: A historical review of evolutionary learning methods for Mamdani-type fuzzy rule-based systems: Designing interpretable genetic fuzzy systems. *International Journal of Approximate Reasoning* **52**(6), 894–913 (Sep 2011). <https://doi.org/10.1016/j.ijar.2011.03.004>
4. de Lima, A., Carvalho, S., Dias, D.M., Naredo, E., Sullivan, J.P., Ryan, C.: Lexi2: Lexicase selection with lexicographic parsimony pressure. In: Proceedings of the Genetic and Evolutionary Computation Conference. pp. 929–937. GECCO '22, Association for Computing Machinery, New York, NY, USA (Jul 2022). <https://doi.org/10.1145/3512290.3528803>
5. de Lima, A.D., Lopes, A.J., do Amaral, J.L.M., de Melo, P.L.: Explainable machine learning methods and respiratory oscillometry for the diagnosis of respiratory abnormalities in sarcoidosis. *BMC Medical Informatics and Decision Making* **22**(1), 274 (Oct 2022). <https://doi.org/10.1186/s12911-022-02021-2>
6. Dos Santos, A.R., Amaral, J.L.M.: Synthesis of Fuzzy Pattern Trees by Cartesian Genetic Programming. *Mathware & soft computing: The Magazine of the European Society for Fuzzy Logic and Technology* **22**(1), 52–56 (2015)
7. Dua, D., Graff, C.: UCI machine learning repository (2017), <http://archive.ics.uci.edu/ml>
8. Fortin, F.A., De Rainville, F.M., Gardner, M.A.G., Parizeau, M., Gagné, C.: DEAP: Evolutionary algorithms made easy. *The Journal of Machine Learning Research* **13**(1), 2171–2175 (Jul 2012)
9. Gandomi, A.H., Alavi, A.H.: A new multi-gene genetic programming approach to nonlinear system modeling. Part I: Materials and structural engineering problems. *Neural Computing and Applications* **21**(1), 171–187 (Feb 2012). <https://doi.org/10.1007/s00521-011-0734-z>
10. Helmuth, T., Lengler, J., La Cava, W.: Population Diversity Leads to Short Running Times of Lexicase Selection. In: Rudolph, G., Kononova, A.V., Aguirre, H., Kerschke, P., Ochoa, G., Tušar, T. (eds.) *Parallel Problem Solving from Nature – PPSN XVII*. pp. 485–498. *Lecture Notes in Computer Science*, Springer International Publishing, Cham (2022). https://doi.org/10.1007/978-3-031-14721-0_34
11. Helmuth, T., McPhee, N., Spector, L.: Lexicase Selection for Program Synthesis: A Diversity Analysis. In: *Genetic Programming Theory and Practice XIII*, pp. 151–167. Springer (Dec 2016). https://doi.org/10.1007/978-3-319-34223-8_9
12. Helmuth, T., Pantridge, E., Spector, L.: On the importance of specialists for lexicase selection. *Genetic Programming and Evolvable Machines* **21**(3), 349–373 (Sep 2020). <https://doi.org/10.1007/s10710-020-09377-2>
13. Helmuth, T., Spector, L., Matheson, J.: Solving Uncompromising Problems With Lexicase Selection. *IEEE Transactions on Evolutionary Computation* **19**(5), 630–643 (Oct 2015). <https://doi.org/10.1109/TEVC.2014.2362729>
14. Hernandez, J.G., Lalejini, A., Ofria, C.: An Exploration of Exploration: Measuring the Ability of Lexicase Selection to Find Obscure Pathways to Optimality. In: Banzhaf, W., Trujillo, L., Winkler, S., Worzel, B. (eds.) *Genetic*

- Programming Theory and Practice XVIII, pp. 83–107. Genetic and Evolutionary Computation, Springer Nature, Singapore (2022). https://doi.org/10.1007/978-981-16-8113-4_5
15. Huang, Z., Gedeon, T.D., Nikraves, M.: Pattern Trees Induction: A New Machine Learning Method. *IEEE Transactions on Fuzzy Systems* **16**(4), 958–970 (Aug 2008). <https://doi.org/10.1109/TFUZZ.2008.924348>
 16. Jackson, D.: Promoting Phenotypic Diversity in Genetic Programming. In: Schaefer, R., Cotta, C., Kołodziej, J., Rudolph, G. (eds.) *Parallel Problem Solving from Nature, PPSN XI*. pp. 472–481. *Lecture Notes in Computer Science*, Springer, Berlin, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15871-1_48
 17. Kooperberg, C.: StatLib: An archive for statistical software, datasets, and information. *The American Statistician* **51**(1), 98 (Feb 1997)
 18. Koshiyama, A.S., Vellasco, M.M.B.R., Tanscheit, R.: GPFIS-CLASS: A Genetic Fuzzy System based on Genetic Programming for classification problems. *Applied Soft Computing* **37**, 561–571 (Dec 2015). <https://doi.org/10.1016/j.asoc.2015.08.055>
 19. La Cava, W., Spector, L., Danai, K.: Epsilon-Lexicase Selection for Regression. In: *Proceedings of the Genetic and Evolutionary Computation Conference 2016*. pp. 741–748. *GECCO '16*, Association for Computing Machinery, New York, NY, USA (Jul 2016). <https://doi.org/10.1145/2908812.2908898>
 20. Luke, S., Panait, L.: Lexicographic parsimony pressure. In: *Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation*. pp. 829–836. *GECCO'02*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (Jul 2002)
 21. Mei, Y., Chen, Q., Lensen, A., Xue, B., Zhang, M.: Explainable Artificial Intelligence by Genetic Programming: A Survey. *IEEE Transactions on Evolutionary Computation* **27**(3), 621–641 (Jun 2023). <https://doi.org/10.1109/TEVC.2022.3225509>
 22. Miller, J.F.: Cartesian Genetic Programming. In: Miller, J.F. (ed.) *Cartesian Genetic Programming*, pp. 17–34. *Natural Computing Series*, Springer, Berlin, Heidelberg (2011). https://doi.org/10.1007/978-3-642-17310-3_2
 23. Montana, D.J.: Strongly typed genetic programming. *Evolutionary Computation* **3**(2), 199–230 (Jun 1995). <https://doi.org/10.1162/evco.1995.3.2.199>
 24. Moore, J.M., Stanton, A.: Lexicase selection outperforms previous strategies for incremental evolution of virtual creature controllers. In: *ECAL 2017, the Fourteenth European Conference on Artificial Life*. pp. 290–297. MIT Press (Sep 2017). https://doi.org/10.1162/isal_a_050
 25. Murphy, A., Ali, M., Dias, D., Amaral, J., Naredo, E., Ryan, C.: Grammar-based Fuzzy Pattern Trees for Classification Problems. In: *Proceedings of the 12th International Joint Conference on Computational Intelligence*. pp. 71–80. *SCITEPRESS - Science and Technology Publications*, Budapest, Hungary (2020). <https://doi.org/10.5220/0010111900710080>
 26. Murphy, A., Ali, M.S., Mota Dias, D., Amaral, J., Naredo, E., Ryan, C.: Fuzzy Pattern Tree Evolution Using Grammatical Evolution. *SN Computer Science* **3**(6), 426 (Aug 2022). <https://doi.org/10.1007/s42979-022-01258-y>
 27. Murphy, A., Murphy, G., Amaral, J., MotaDias, D., Naredo, E., Ryan, C.: Towards Incorporating Human Knowledge in Fuzzy Pattern Tree Evolution. In: Hu, T., Lourenço, N., Medvet, E. (eds.) *Genetic Programming*. pp. 66–81. *Lecture Notes in Computer Science*, Springer International Publishing, Cham (2021). https://doi.org/10.1007/978-3-030-72812-0_5

28. Murphy, A., Murphy, G., Dias, D.M., Amaral, J., Naredo, E., Ryan, C.: Human in the Loop Fuzzy Pattern Tree Evolution. *SN Computer Science* **3**(2), 163 (Feb 2022). <https://doi.org/10.1007/s42979-022-01044-w>
29. Rodrigues dos Santos, A., Machado do Amaral, J.L., Ribeiro Soares, C.A., Valladão de Barros, A.: Multi-objective Fuzzy Pattern Trees. In: 2018 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE). pp. 1–6 (Jul 2018). <https://doi.org/10.1109/FUZZ-IEEE.2018.8491689>
30. Ryan, C., Collins, J.J., Neill, M.O.: Grammatical evolution: Evolving programs for an arbitrary language. In: Banzhaf, W., Poli, R., Schoenauer, M., Fogarty, T.C. (eds.) *Genetic Programming*. pp. 83–96. *Lecture Notes in Computer Science*, Springer, Berlin, Heidelberg (1998). <https://doi.org/10.1007/BFb0055930>
31. Senge, R., Hüllermeier, E.: Pattern trees for regression and fuzzy systems modeling. In: *International Conference on Fuzzy Systems*. pp. 1–7 (Jul 2010). <https://doi.org/10.1109/FUZZY.2010.5584231>
32. Senge, R., Hüllermeier, E.: Top-Down Induction of Fuzzy Pattern Trees. *IEEE Transactions on Fuzzy Systems* **19**(2), 241–252 (Apr 2011). <https://doi.org/10.1109/TFUZZ.2010.2093532>
33. Shaker, A., Senge, R., Hüllermeier, E.: Evolving fuzzy pattern trees for binary classification on data streams. *Information Sciences* **220**, 34–45 (Jan 2013). <https://doi.org/10.1016/j.ins.2012.02.034>
34. Sobania, D., Rothlauf, F.: Program Synthesis with Genetic Programming: The Influence of Batch Sizes. In: Medvet, E., Pappa, G., Xue, B. (eds.) *Genetic Programming*. pp. 118–129. *Lecture Notes in Computer Science*, Springer International Publishing, Cham (2022). https://doi.org/10.1007/978-3-031-02056-8_8
35. Spector, L.: Assessment of problem modality by differential performance of lexibase selection in genetic programming: A preliminary report. In: *Proceedings of the 14th Annual Conference Companion on Genetic and Evolutionary Computation*. pp. 401–408. *GECCO '12*, Association for Computing Machinery, New York, NY, USA (Jul 2012). <https://doi.org/10.1145/2330784.2330846>
36. White, D.R., McDermott, J., Castelli, M., Manzoni, L., Goldman, B.W., Kronberger, G., Jaskowski, W., O'Reilly, U.M., Luke, S.: Better GP benchmarks: Community survey results and proposals. *Genetic Programming and Evolvable Machines* **14**(1), 3–29 (Mar 2013). <https://doi.org/10.1007/s10710-012-9177-2>
37. Wilkinson, L., Anand, A., Tuan, D.N.: CHIRP: A new classifier based on composite hypercubes on iterated random projections. In: *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. pp. 6–14. *KDD '11*, Association for Computing Machinery, New York, NY, USA (Aug 2011). <https://doi.org/10.1145/2020408.2020418>
38. Yi, Y., Fober, T., Hüllermeier, E.: Fuzzy operator trees for modeling rating functions. *International Journal of Computational Intelligence and Applications* **08**(04), 413–428 (Dec 2009). <https://doi.org/10.1142/S1469026809002679>