A Closer Look at Length-niching Selection and Spatial Crossover in Variable-length Evolutionary Rule Set Learning

David Pätzel
University of Augsburg
Germany
david.paetzel@uni-a.de

Richard Nordsieck
XITASO GmbH
Germany
richard.nordsieck@xitaso.de

Jörg Hähner
University of Augsburg
Germany
joerg.haehner@uni-a.de

ABSTRACT

We explore variable-length metaheuristics for optimizing sets of rules for regression tasks by extending an earlier short paper that performed a preliminary analysis of several variants of a single-objective Genetic Algorithm. We describe more in depth the algorithm and operator variants used and document design decisions as well as the rationale behind them. The earlier work identified crossover as being detrimental for solution compactness; we take a closer look by analysing convergence behaviour of the variants tested. We are able to conclude that using one of the investigated crossover operators trades prediction error outliers for more smaller errors at the expense of solution compactness. The positive effects of length-niching selection (holding off premature convergence to a certain solution length) are undetectable in fitness values in the settings considered. We further perform comparisons with already-known rule-based algorithms XCSF and CART Decision Trees and conclude that, even without parameter tuning, the best-performing of the variants of the GA outperforms XCSF on the tasks considered, comes close to being competitive with respect to test Mean Absolute Error and creates similarly compact solutions as the Decision Tree algorithm. The 54 learning tasks considered are synthetic and in the limit learnable by rule-based algorithms.

CCS CONCEPTS

• **Computing methodologies → Rule learning**; *Supervised learning*; *Genetic algorithms*.

KEYWORDS

Genetic Algorithm, Variable-length Representation, Metaheuristic Rule Set Learning, Learning Classifier Systems

ACM Reference Format:
David Pätzel, Richard Nordsieck, and Jörg Hähner. 2024. A Closer Look at Length-niching Selection and Spatial Crossover in Variable-length Evolutionary Rule Set Learning. In *Genetic and Evolutionary Computation Conference (GECCO '24 Companion), July 14–18, 2024, Melbourne, VIC, Australia.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3638530.3664178

This work is licensed under a Creative Commons Attribution International 4.0 License.

GECCO '24 Companion, July 14–18, 2024, Melbourne, VIC, Australia
© 2024 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0495-6/24/07
https://doi.org/10.1145/3638530.3664178

1 INTRODUCTION

Rule Set Learners (RSLs) are algorithms that build models consisting of sets of rules. They are intrinsically interpretable [7] (as long as the form of rules used is not too complicated and the number of rules is not too high) and therefore an interesting family to consider when explainability is one of the requirements for a model. While there are approaches such as *Decision Tree* (DT) algorithms (e. g. CART [8]) which use local heuristics to build a set of rules, there are also algorithms that utilize metaheuristics to directly search rule set space. Such a direct search often has advantages such as being able to define user preferences more straightforwardly via fitness functions, allowing rules to overlap and generally having a higher algorithmic flexibility. Within this paper, we call these algorithms *Metaheuristic Rule Set Learners* (MRSLs). Examples for families of algorithms that fall into this category are Learning Classifier Systems (LCSs) [15, 39], Evolutionary Fuzzy Rule-based Systems [11] and Ant-Miner [23]. We are particularly interested in systems that optimize at the level of rule sets instead of at the level of individual rules since they by design yield much more compact solutions (i. e. fewer rules and therefore an increased interpretability) which has also been observed in experiments [e. g. 18]. The present paper is further restricted to regression tasks.

Recent work by Ryerkerk et al. [31, 32] stresses the importance of using variable-length encodings and appropriate metaheuristic operators if the problems being solved are variable-length—even showing that variable-length algorithms can perform better than fixed-length algorithms on tasks where optimal solution length is known [33]. The task of finding well-performing rule sets is inherently variable-length since not only rule parameters but also the *number* of rules has to be optimized. Variable-length approaches therefore seem to be a natural fit for MRSLs and while there have been proposed MRSLs that do use variable-length approaches in the past [e. g. 4, 6, 12], research attention for these has waned over the years.

This motivated us to explore the idea of using variable-length encodings in MRSL algorithms. A first short study within this direction was [26] where a simplistic single-objective *Genetic Algorithm* (GA) based on the *corrected Akaike Information Criterion* (AICc) [1, 9, 20, 38] was used to perform a preliminary comparison of two variable-length operators (*spatial crossover* [10] and *length-niching selection* [31]) which had not been used in the context of rule learning before but look promising on paper: Spatial crossover is meant to promote building blocks in continuous parameter spaces just like the ones defined by rule conditions whereas length-niching selection tries to prevent premature convergence to a certain solution length and consequent loss of diversity. However, due to spatial constraints that earlier work was not able to fully describe the algorithm used nor in depth document the design decisions and rationale behind them properly. The present paper provides such a description and closer look and further extends the analysis of the earlier work by 1) performing an analysis of convergence behaviour of the variants tested, 2) giving more insights into the effects of crossover which earlier results indicated hurt solution compactness and 3) baselining the best variant against two already known algorithms for building rule sets, CART [8] and XCSF [42]. We publish our code as a Julia library [28, 29]. The synthetic learning tasks we use are generated such that they are, in theory (i. e. in the limit of infinite training data), solveable by the systems we consider and their progressive difficulty can be gauged.

2 RELATED WORK

To our knowledge, neither spatial crossover nor length-niching selection have been used in MRSL algorithms before.

In the LCS community, the algorithm we use would be called a Pittsburgh-style system [15] as its metaheuristic operates at the level of rule sets. While up to the mid 2000s, there have been designed quite a few Pittsburgh-style systems, interest in these algorithms has waned over the years. Urbanowicz and Moore [40] present an extensive review of systems (both Pittsburgh- and other styles) up to 2009 and show that at the time all known Pittsburgh-style systems (and actually most of LCS research) specialize on classification and similar discrete tasks whereas our work deals with regression.

A variable-length encoding which is similar to the one used in the present study is used by the GABIL system [12] and systems using its representation [e. g. 4, 6]. However, these approaches focus on concept learning or classification whereas we investigate regression tasks.

The XCS classifier system [41] and its descendants [e. g. 42] optimize a set of rules by applying local operators to each rule. While they attempt to steer the system towards smaller rule sets using implicit evolutionary pressures, in most cases, the resulting rule sets are comparably large. The GA we use for our study and the operators we consider are meant to be applied to entire rule sets and not to individual rules which enables the designer to explicitly model the optimization target at the rule set level (including a preference for small solutions).

Another line of research that goes into a similar direction as the one presented here is Drugowitsch's work on a probabilistic framework for LCSs [13] which has not seen any further development since 2008 other than a replicability study a few years ago [24] and is in its current state computationally too expensive to be meaningfully applied to problems with as few as two- or three-dimensional input spaces.

We further should mention Heider et al.'s recent work on the SupRB algorithm for regression tasks [e. g. 17, 19]. Other than our algorithm, SupRB uses a fixed-length representation for the set of conditions and uses two metaheuristics: One to build a pool of well-performing rules and another one to optimize bit strings that represent subset selections from that pool. In Ryerkerk et al.'s terminology [32], this constitutes a *hidden-metavariable representation* and an extensive comparison with that system is definitely warranted in the future. Note that we compared our system to XCSF instead of SupRB in the present paper because, being a more established algorithm, it lends itself better as a first baseline.

Fan and Gray [14] construct regression trees using metaheuristics and compare the results with several other tree learning algorithms. They use the *Bayesian Information Criterion* as their fitness measure which is in form similar to the AICc we use but has a different motivation (and different strengths and weaknesses [9]). Tree-based rule sets are non-overlapping whereas our MRSL algorithm explicitly allow rules to overlap which can result in more compact models.

3 RULE SET MODELS

We start by defining the model family optimized by our MRSL algorithm which we will propose in the next section. The learning tasks we consider are supervised regression tasks: We're looking for an optimal model $\hat{f} : \mathcal{X} \to \mathcal{Y}$ that maps inputs $x \in \mathcal{X} = \mathbb{R}^{\mathcal{D}_\mathcal{X}}$ to outputs $y \in \mathcal{Y} = \mathbb{R}$. The sole resource for identifying this optimal model is a *training set*, comprising N inputs $(x_n)_{n=1}^N = X \in \mathcal{X}^N$ and outputs $(y_n)_{n=1}^N = y \in \mathcal{Y}^N$.

Within this paper, a *rule set model* corresponds to a *set of K rules*. Each rule k in the model consists of a *condition* $m(\psi_k; \cdot) : \mathcal{X} \to \{0, 1\}$ with parameters ψ_k and a *local model* which is in general modelled as an input-dependent random variable $Y_k(x)$. The overall output of a model for a fixed input $x \in \mathcal{X}$ is a random variable $Y(x)$ which is a normalized weighted sum ($\gamma_k \in \mathbb{R}$ is rule k's *mixing coefficient*) of the local model outputs of all the rules matching[1] x:

$$Y(x) = \sum_{k=1}^{K} \frac{m_k(\psi_k; x)\gamma_k}{\sum_{j=1}^{K} m_j(\psi_j; x)\gamma_j} Y_k(x) \tag{1}$$

The role of the mixing coefficients γ_k is that of weighing rules against each other in areas of overlap (more on that in Section 4.2). Note that an expression similar to Equation (1) can be found in the literature [25] but we explicitly normalize the mixing weights.[2]

We choose conditions to correspond to *intervals* in $\mathcal{X}$. This means that

$$m(\psi_k; x) = m((l_k, u_k); x) = \begin{cases} 1, & x \in [l_k, u_k) \\ 0, & \text{otherwise.} \end{cases} \tag{2}$$

Interval-based conditions are a common choice in MRSL algorithms with real-valued input [e. g. 19, 42] because they are human-comprehensible greater-/less-than statements and therefore often yield models that are preferred due to their higher interpretability [16] over other common choices such as ellipsoid-based conditions [36].

Since the focus of the present work is analysing metaheuristic operators in the variable-length setting, we chose the simplest possible model family for the local models, namely, *constant models*. In the context of the present paper, this has several advantages over other common choices like linear regression local models (e. g. used by some XCS variants [42] or SupRB [19]): The computational cost

[1] A rule k with condition parameters ψ_k is said to *match* input x if $m(\psi_k; x) = 1$.

[2] Mixing weight normalization makes it necessary for every point in the input space to be matched by at least one rule (otherwise Equation (1) is not defined due to division by zero). This can be easily solved by adding to any model a *default rule* which matches every possible input, predicts the training data mean and has a mixing weight very close to zero (so that it only significantly influences model output in areas where no other rule matches).

of fitting them is lower and a direct fair comparison with CART DTs is possible (these DTs use constant local models as well [8] and while there exist extensions to linear local models [e. g. 35], they are not well-established so far). Further, constant models allow us to put more focus on the metaheuristic aspects of the MRSL algorithm: We argue that if an MRSL metaheuristic isn't able to optimize constant model placement for approximating noisy step functions, it will probably not be able to optimize linear or even higher-order models for more difficult tasks. We also expect using constant local models to facilitate analysis of the metaheuristic's behaviour.

Of course, predictive performance of constant local models on most real-world data sets is naturally lower than of more sophisticated models with a higher parameter count. We account for this by selecting the learning tasks considered in this paper to be noisy step functions; these are (in the limit of infinite training data) learnable optimally by models consisting of constant local models.

We define the output of rule k's local model as being normally distributed around a mean μ_k with variance σ_k^2—the latter being the local model's *noise estimate*. This can be written as $Y_k(x) \sim \mathcal{N}(\mu_k, \sigma_k^2)$. Since normally distributed variables are closed under addition and multiplication with scalars, this yields an overall model output distribution which is normal as well:

$$Y \mid x \sim \mathcal{N}\left(y \mid \sum_{k=1}^{K} \frac{m_k(x)\gamma_k}{\sum_{j=1}^{K} m_j(x)\gamma_j}\mu_k, \sum_{k=1}^{K}\left(\frac{m_k(x)\gamma_k}{\sum_{j=1}^{K} m_j(x)\gamma_j}\right)^2 \sigma_k^2\right) \tag{3}$$

We can make predictions for a given $x \in \mathcal{X}$ by evaluating the output distribution's mean. Additionally, the distribution's variance can be interpreted as being a measure of confidence in the prediction.

4 SEARCHING FOR VARIABLE-LENGTH RULE SETS

It is generally understood that MRSL algorithms perform both *model structure optimization* and *parameter fitting* [13, 15]. A rule set's model structure is the set of its conditions (to be more precise, the number of rules in the set K and the vector of all the rule's condition parameters $(\psi_k)_{k=1}^K$). Each model structure $\mathcal{M} = \left(K, (\psi_k)_{k=1}^K\right)$ defines a parametric model family whose parameters can be fitted rather straightforwardly using common techniques since each rule's local model is assumed to be *simple*[3] (e. g. constant or linear) and can be fitted independently of the other rules' local models. Therefore, most of an MRSL algorithm's computational effort goes into model structure optimization which is the MRSL algorithm's metaheuristic's task. Pätzel et al. [25] discuss these general concepts more in-depth.

In the present work, the metaheuristic chosen is a *Genetic Algorithm* (GA) similar to the one used by Ryerkerk et al. in their paper on the length-niching selection operator [31] which will be discussed in Section 4.3.5. Our GA operates on model structures,[4] that is, *genotypes* are structured variable-length vectors of condition parameters that correspond to the set of conditions of a set of rules. Ryerkerk et al. [32] calls such a structured representation a *metameric* representation: In their terminology, each group of condition parameters belonging to a single condition is one *metavariable* whereas each single parameter (i. e. each real-valued bound of one of the intervals) is a *design variable*. For each genotype, the corresponding *phenotype* is then a rule set model of the form given in the previous section. Computing the phenotype involves fitting local model parameters and finding mixing weights and is discussed in Section 4.2.

4.1 Defining rule set fitness

In order to define the GA, we have to define a fitness function, that is, a way to measure optimality of a set of rules given training data. What constitutes an optimal set of rules is, however, not agreed upon in the MRSL literature and especially unclear for regression tasks [cf. e. g. 13]. A formal approach to this problem is made by Drugowitsch [13], but the probabilistic model he uses is computationally infeasible even for low input space dimensionalities.

A central part of the optimality of a rule set is of course the induced model's predictive performance on the training data. This can be measured in many ways and is typically done based on *predictive error* (e. g. Mean Absolute Error). However, optimizing for low predictive error alone optimizes towards having one rule per training data point since that way the training error is minimal. Such a solution is seldomly acceptable as it can not be expected to generalize well to unseen data. Fitness measures therefore also have to consider *model complexity*. A typical way to measure this is to count the number of model parameters which is in the case of rule sets proportional to the number of rules in the set. We therefore have two optimization targets: Predictive error can be expected to decrease when the number of (sensibly placed) rules increases. There are several approaches to balance these targets against each other.

One direction pursued by several approaches to build MRSL algorithms [e. g. 2, 3, 22] is to perform multi-objective optimization. A strength of these schemes is that they allow the system's user to choose a model from a set of pareto-optimal models. However, this can at the same time be seen as a weakness: We argue that most users already have some sort of preference with respect to the tradeoff between solution size and predictive performance (e. g. due to explainability requirements) and that this preference should be used to direct the search process to preferred regions of solution space. Exploring regions that are definitely not preferred by the user during search is a waste of energy and time. Another approach in multi-objective optimization is the practice of computing a mixture model from the models on the pareto front (e g. to increase predictive performance [21]); however, if interpretability is preferred, this isn't an option either. We therefore opt for a single-objective approach for our GA.

Another common choice to consolidate predictive error and model complexity is to use a weighted sum or other form of combination of the targets as the fitness measure [e. g. 19]. This can, in discrete or classification settings, also take on information-theoretic forms such as the Minimum Description Length [e. g. 5]. In a similar spirit, and since we have an expression for the likelihood due to knowing the output distribution (Equation (3)) we are able to use

[3] If local models were not simple, then the overall model is not interpretable anymore and other (non-RSL) learning algorithms are probably a better choice anyways.

[4] This differs, for example, from the GA in systems like XCS which operates at the level of individual rules [41].

the *negative Akaike Information Criterion* [1] with the generally recommended *small sample correction* [9, 20, 38] (AICc) as the fitness measure for our system and therefore define,

$$\text{fitness}(\mathcal{M}) = 2 \log \mathcal{L}(\mathcal{M} \mid X, y) - 2\nu_{\mathcal{M}} - \frac{2\nu_{\mathcal{M}}(\nu_{\mathcal{M}} + 1)}{n - \nu_{\mathcal{M}} - 1} \quad (4)$$

where $\mathcal{L}(\mathcal{M} \mid X, y)$ is the likelihood of the model structure $\mathcal{M}$ given the training data, n is the number of training data points and $\nu_{\mathcal{M}} = 2K\mathcal{D}_{\mathcal{X}} + 2K$ is the number of parameters in the model structure $\mathcal{M}$. The AICc is well-understood and formally grounded and allows for a principled model comparison; at the same time, it can be interpreted as a weighted sum of a model complexity and a predictive error term.

4.2 Fitting local models and mixing weights

Given a K-vector of condition parameters $(\psi_k)_{k=1}^{K}$, a set of rules is built by fitting local models and computing mixing coefficients. For rule k, we choose local model parameters based on the training data that k matches: We simply fit the local model's output distribution parameters (μ_k, σ_k^2) to the matched outputs $\{y_j \mid m(\psi_k, x_j) = 1\}$ using the well-known *Maximum Likelihood Estimation* (MLE) formulae for normal distributions without the Bessel correction. The mixing coefficient of rule k is chosen as the inverse of the expected error, that is, $\gamma_k = \sigma_k^{-2}$. This is known to be a well-performing heuristic [13] with the intuition being that lower-error rules contribute more to the overall prediction than higher-error rules.

Given a model structure $\mathcal{M}$, we can now compute parameters for the full model given in Equation (3). The next section deals with the operators used to search for well-performing $\mathcal{M}$.

4.3 Searching for condition parameters

A high-level overview of the GA we use is given in Algorithm 1. We will now discuss each of the operators used by it more in-depth.

4.3.1 *Initialization.*

We want to initialize the rule set population similarly to the initialization performed by Ryerkerk et al. [31]: They enable the user to provide a range of solution (rule set) lengths from which rule set lengths for an initial population are then drawn at random. After that, rule sets of these lengths are drawn at random.

While we could draw a rule set of a given length K by drawing fully random rules (e. g. uniformly distributed interval bounds that are then reordered correctly), we think that we can actually help the metaheuristic by drawing at least *somewhat sensible* initial rule sets. At that, we define *somewhat sensible* as *the combined rule conditions in the set should cover at least 90 % of the training data and each condition should further cover some training data points that other rules do not cover.* This encodes two beliefs: We think it is less likely to perform well for rule sets that cover only small parts of the training data than for rule sets that cover most of the training data. Further, in most cases, rules that only cover training data points already covered by other rules add little value.

In order to fulfill this sensibility condition, we simply repeatedly draw a single rule which matches at least one, as of yet unmatched, training data point and stop as soon as at least 90 % of training data points are covered by rules. In order to fulfil the first condition of the rule set having length K, the distribution parameters θ for drawing a single rule have to be chosen properly. Since this proved to be difficult when done manually, we simply Monte-Carlo sampled the process of drawing rule sets very often for randomly drawn distribution parameters θ and took note of which parameters θ led to which K's given which input space dimensionality. We then derived for all required Ks and input space dimensionalities the parameters θ by roughly approximating the posterior density mode using a histogram.[5] In the process of doing these steps, we found that this kind of analysis to find optimal parameters could be beneficial for many other metaheuristic operators as well.

```
function ga()
  elitist = nothing
  pop = init()
  evaluate!(pop)
  update!(elitist, pop)
  for iter in 1:n_iter
    offspring = []
    for (i1, i2) in randompairs(pop)
        o1, o2 = recomb(pop[i1], pop[i2])
        mutate!(o1); mutate!(o2)
        repair!(o1); repair!(o2)
        evaluate!(o1); evaluate!(o2)
        append!(offspring, [o1, o2])
      end
    end
    update!(elitist, offspring)
    pop = select(pop ∪ offspring)
  end
  return elitist
end
```

Algorithm 1: The used GA. To keep notation concise, in-place operations are marked by a trailing "!".

4.3.2 *Recombination: Spatial and cut-and-splice crossover.*

We compare two of the crossover operators considered by Ryerkerk et al. [32] in their survey of variable-length metaheuristics: Cut-and-splice and spatial crossover.

Cut-and-splice is the variable-length version of the well-known n-point crossover. In the case of the genotypes we consider, cut-and-splice works by redistributing the conditions of the two possibly different-length parents $(\psi_k)_{k=1}^{K_1}$ and $(\psi_k)_{k=1}^{K_2}$ between two children at random. At that, each condition from either parent is given to exactly one child and each child receives at least one condition and at most $K_1 + K_2 - 1$. While cut-and-splice crossover seems to be a common choice in variable-length metaheuristics [32], it is considered very disruptive as it does not consider any kind of dependency between the variables.

Spatial crossover [10] differs from cut-and-splice in that it *does* consider certain dependencies between variables. In the case of the genotypes we consider, we choose a random condition from either of the parents and compute the corresponding interval's center. We then draw a plane through that point (with a random angle) which partitions input space in two. This allows us to assign conditions to the two children based on which side of the plane their center

[5] Out of brevity, we have to refer to our code [28] for more details.

lies on, similarly to how 1-point crossover generates offspring in binary settings. In the present case, spatial crossover can be seen as an attempt to identify building blocks that are spatially close groups of conditions which can be separated from the rest of the conditions by a plane. See [32] for a more in-depth description and an illustration of this concept. It should be noted that, to our knowledge, spatial crossover has not been used before in MRSL algorithms while cut-and-splice is a common method.

4.3.3 Mutation. Mutation of a rule set is performed just like in the GA defined by Ryerkerk et al. [31]: We first go over each rule's condition and with a probability of p_{mut} independently apply to each interval bound a Gaussian mutation with variance σ^2_{mut}. Then, independently of that, with a probability of p_{add}, a new rule (i. e. new condition parameters) is added to the set, thus increasing its length by one. The new rule is chosen such that it matches a certain minimum number[6] of training data points with data points not matched by any other rule being preferred. Afterwards, again independently and with a probability of p_{rm}, an existing rule is deleted from the rule set (decreasing its length by one)—but only if this does not leave an empty set.

4.3.4 Repair. After crossover and mutation, there could be rules in any of the rule sets that match too few training data points to be considered sensible. We repair all rule sets by simply deleting such rules. Within the present paper, we expect each rule to match at least two training data points to not be deleted. Note that this is a hyperparameter that could be tuned or be set according to user preferences.

4.3.5 Selection: Length-niching and tournament selection. Finally, the genetic algorithm performs selection. At that, we compare two selection regimes: Ryerkerk et al.'s length-niching selection using a biased selection window [31] and the well-known tournament selection.

Length-niching selection [31] attempts to keep population diversity higher than other selection mechanisms by defining a range of solution lengths (niches) for each of which an independent local selection is carried out. This way, solutions of a certain length compete less directly with solutions of different lengths during selection and population diversity with respect to solution length is maintained. This is meant to help prevent premature convergence. Ryerkerk et al. [31] proposed several methods to define the *length window* which is the range of solution lengths that are propagated to the next generation. Of the ones discussed, the biased window approach was chosen for the present paper because it does not require a priori knowledge of the optimal solution length but instead updates the window during search based on search behaviour. Biased-window length-niching selection has two hyperparameters, the window length w and the bias decay λ.

5 EXPERIMENTAL SETUP

This section describes how we performed experiments whose results we will then discuss in the next section.[7]

As was already mentioned above, the learning tasks we consider are regression tasks and, more precisely, noisy step functions. In order to obtain these step functions, we follow the procedure proposed by Pätzel et al. [25] but use constant local models: For each of three input dimensionalities considered ($\mathcal{D}_\mathcal{X} \in \{3, 5, 8\}$), for each of three model sizes ($K \in \{4, 8, 12\}$), we repeatedly draw random data-generating processes (i. e. random parameters for Equation (1)) until we have 6 such models whose respective conditions jointly cover 90 % of the input space (Monte-Carlo estimated). This yields a total of 54 learning tasks[8] from each of which we then generate train and test data sets of different sizes (Table 1) to account for increasing dimensionality.[9] Test data sets were ten times as large as training data sets and drawn independently.

Note that in the limit of infinite training data, the number of rules in the data-generating process corresponds to the number of rules of an optimal model for the corresponding learning task. However, within the present paper, we try to use more realistic training data set sizes and therefore this correspondence is likely not fulfilled. Nevertheless, for a fixed input space dimensionality, the number of rules in the data-generating process can be seen as a direct indicator for expected task difficulty.

Table 1: Learning tasks used. $\mathcal{D}_\mathcal{X}$ is input dimensionality, K is number of rules in the data-generating process, N is size of training data set.

$\mathcal{D}_\mathcal{X}$	K	$N = 200 \cdot 10^{\frac{\mathcal{D}_\mathcal{X}}{5}}$	Number of learning tasks
3	4	796	6
3	8	796	6
3	12	796	6
5	4	2000	6
5	8	2000	6
5	12	2000	6
8	4	7962	6
8	8	7962	6
8	12	7962	6

We investigate the performance of a total of 7 variants of our MRSL algorithm on these tasks. In doing so, we vary

- the crossover operator between spatial crossover, cut-and-splice crossover and no crossover at all (if crossover is enabled, crossover probability is 0.3),
- the selection operator between length-niching (window width $w = 7$; bias factor decay like in the original paper [31], i. e. $\lambda = 0.004$) and tournament selection (tournament size 4),
- the probability of adding and/or removing rules during mutation between being low ($p_{\text{add}} = p_{\text{rm}} = 0.05$) or high ($p_{\text{add}} = p_{\text{rm}} = 0.4$).

The exact combinations of these options that we test are given in Table 2.

In all experiments, we further

- set population size to 32,

[6] We currently hardcode this hyperparameter to four since the repair operator removes rules that match less than two training data points.

[7] The code for our experiments can be found on Zenodo [28, 29].

[8] We publish learning tasks and experiment data on Zenodo as well [30].

[9] Note that we do not increase data set size directly proportional to dimensionality but instead try to mimic a typical data set size for each dimensionality instead.

Table 2: MRSL algorithm variants tested.

crossover	spatial		cut-and-splice		off	
$p_{\text{add/rm}}$	low	high	low	high	low	high
selection						
length-niching	×	×	×	×		×
tournament	×	×				

- keep the GA running for 2000 generations with early stopping if no changes in fitness were detected for 500 generations,
- initialize populations with genotypes having lengths drawn from the range $\{3, \ldots, 50\}$ and
- use a standard deviation of $\sigma^2_{\text{mut}} = 0.05$ for the Gaussian mutation and apply said mutation to each real-valued interval bound with probability $p_{\text{mut}} = \frac{1}{2K\mathcal{D}_{\mathcal{X}}}$ (such that on average as many bounds are changed as there are in a single condition).

Note that, aside from initialization where our configuration generates an expected number of 848 rules (divided over 32 rule sets), mutation is the only way that entirely new rules can enter the system. A rate of $p_{\text{add}} = p_{\text{rm}} = 0.05$ means that over the running time 3200 new rules can be expected to be created whereas a rate of $p_{\text{add}} = p_{\text{rm}} = 0.4$ implies creation of 25600 new rules.

Since our MRSL algorithm is non-deterministic, we repeat runs for each of the data sets. However, we noticed that variance on each of the data sets is rather low and therefore opted for only performing five repetitions on each data set for the benefit of being able to include more data sets per $(\mathcal{D}_{\mathcal{X}}, K)$ combination in the analysis.

In order to gauge overall performance of our MRSL algorithm, we also perform comparisons with CART DTs as implemented by the *DecisionTrees.jl* Julia library [34] and XCSF as implemented by Preen’s Python library [27]. We allow the DTs to have between 1 and 70 rules and XCSF to have between 1 and 1000 rules (XCSF is notorious for being rule-hungry). Each of these algorithms is also run 5 times on each of the tasks.

Hyperparameter tuning. We perform hyperparameter tuning for both the DTs and XCSF independently *on each of the data sets* in order to achieve a comparison at least not unfair with respect to these baseline algorithms. For the DTs, we tune the *maximum depth* and the *minimum samples split* parameters whereas for XCSF we tune ϵ_0, β, ν and the *minimum condition spread* parameters. For the remaining parametrization we have to refer the reader to our code [28, 29].

Hyperparameter tuning of our MRSL algorithm is out of the scope of this paper; we first want to perform studies with respect to the hyperparameter’s sensitivities and reduce hyperparameter space this way instead of performing many more computations than necessary. Therefore, the results reported constitute an estimated lower bound for the performance of the presented MRSL algorithm which we expect to improve with proper hyperparameter tuning in the future.

6 EVALUATION AND DISCUSSION

The aforementioned earlier work [26] has already compared the distributions of test Mean Absolute Error (MAE) when pooling over all repetitions of all data sets of each combination of input space dimensionality and data-generating process size. It came to the conclusion that after a fixed number of 2000 generations there was no practically significant—if any at all—difference in test MAE. The number of rules used in the final solutions gave a slightly different picture: 6 of the 7 variants performed equally (i. e. had a visually indistinguishable empirical cumulative distribution) while one variant used significantly fewer rules. That variant was the only variant tested that did *not* use crossover, which is somewhat surprising on first glance.

We therefore take a closer look at this by considering convergence behaviour. Figure 1 shows, for each combination of input space dimensionality and data-generating process size, three *empirical distribution statistics* (empirical median and empirical central 80 % density interval) of the *history of elitist fitness values*. Since different tasks may have different scales for fitness (AICc is not a normalized measure), we normalize for each task with the minimum and maximum elitist fitness values observed over all algorithm variants considered. We first notice that the lower bound of the empirical central intervals for input space dimensionality 3 and data-generating process size 4 for several variants extends down to a relative fitness of 0. This is likely a fragment caused by fitness normalization on this set of simplest tasks considered. Further, there is a tendency for all variants that, for higher dimensionalities and higher data-generating process sizes, convergence takes longer (and may actually have barely been achieved within 2000 generations for some runs). This is expected since 1) for higher input space dimensionalities, search space is exponentially larger and 2) higher data-generating process sizes somewhat correspond to higher learning task difficulties [25].

The ordering of the variants with respect to fitness is quite consistent for all the settings with the variant with spatial crossover and tournament selection achieving the highest fitness values and the variant with spatial crossover and length-niching selection being the runner-up. This may indicate that the benefits of length-niching selection are not as relevant for this setting or that more iterations are needed for them to be detectable in fitness values (remember that length-niching selection mainly tries to prevent premature convergence to a certain solution length); so far, length-niching selection seems to decelerate fitness convergence without any benefit.

Figure 1 further shows that, in every setting, elitist fitness of the variant without crossover (we call this *NOX* for brevity in the following paragraphs) is consistently smaller than the one of the other variants. Given our previous findings [26] of NOX having a lower parameter count (which by itself should yield a higher AICc-based fitness), this can only be explained by a worse fit. While NOX was indistinguishable from the other variants with respect to test and train MAE, there *is* a slight difference to the disadvantage of NOX in the distribution of train (and test) *Mean Squared Error* (MSE) which might explain the worse fitness (Table 3 gives a summary of the data involved—NOX has the highest mean and median MSEs on more than half of the tasks): AICc-based fitness estimates goodness

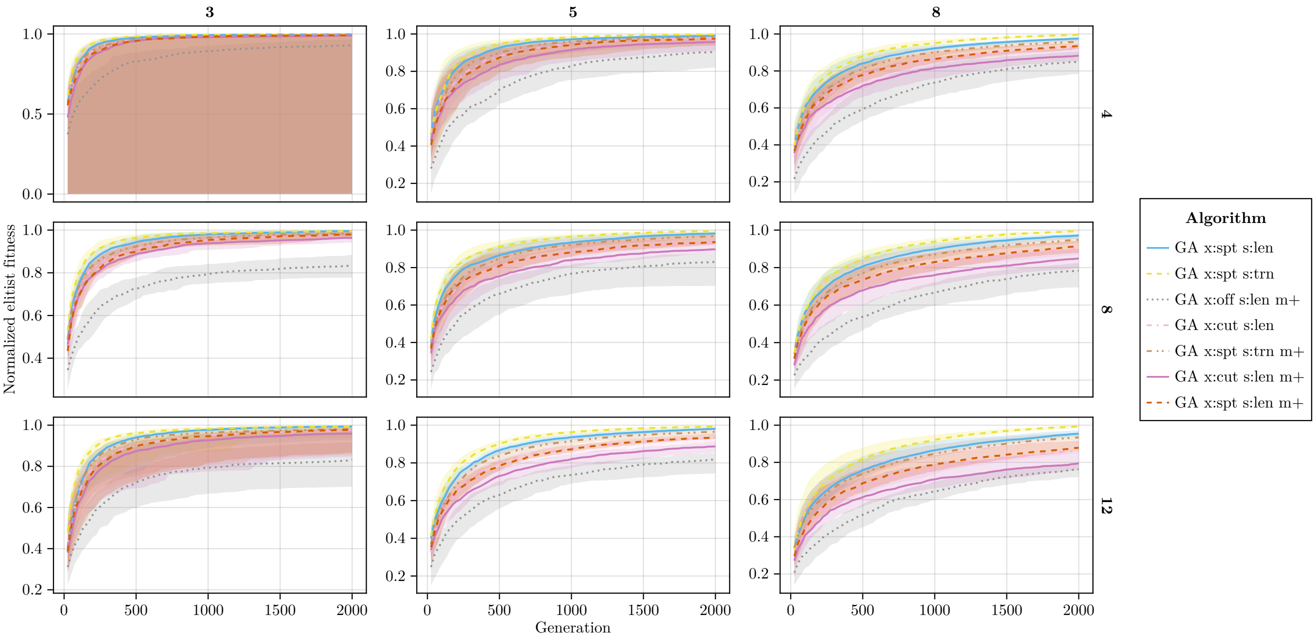

Figure 1: Comparison of the tested variants of our MRSL algorithm. In the algorithm labels, "x:*" denote the crossover operator used (spatial, cut-and-splice, off), "s:*" the selection operator (length-niching, tournament) and "m+" states that the higher mutation rate was in place. Normalized (per task) elitist fitness history when pooling all the runs of each algorithm variant (i. e. pool over 30 runs—5 repetitions and 6 data sets) for each combination of dimensionality $\mathcal{D}_\mathcal{X}$ (columns) and number of rules in the data-generating model (rows). Lines are empirical medians while shaded areas denote the empirical central 80 % interval.

of fit by the training data log likelihood. In the present case the underlying distribution is a normal distribution (Equation (3)) and the data log likelihood of such a distribution entails a quadratic term that directly corresponds to MSE.

Table 3: For each of the 54 tasks considered, the number of times that each variant had the highest mean or median MSE (statistics pooled over the 5 repetitions performed).

	Number of tasks with highest ___ MSE	
Variant	Mean	Median
GA x:off s:len m+	32	29
GA x:cut s:len m+	15	16
GA x:spt s:trn	3	3
GA x:spt s:len m+	2	2
GA x:spt s:trn m+	2	3
GA x:cut s:len	0	1
GA x:spt s:len	0	0

Note that similar MAE but a slightly higher MSE means that NOX's prediction error has more outliers than the prediction errors of the other variants; this can likely be explained with the lower rule count. In summary, using any of the investigated crossover operators *trades a few larger prediction errors for many smaller ones at the expense of solution compactness*. Finding a formal argument explaining this effect is out of the scope of this paper and left for future work.

In order to assess our MRSL algorithm's performance with respect to state-of-the-art RSL algorithms, we performed the same experiments with both XCSF and DT. The results are shown once more as empirical cumulative distributions in Figure 2 (same setup with respect to pooling etc. as in Figure 1) which also includes NOX since it is the variant of our MRSL best performing with respect to MAE. The first thing we notice is that XCSF performs very badly on all the tasks considered both in terms of MAE and number of rules used. While the latter is hardly surprising since we did not use any form of compaction[10] (i. e. pruning techniques to reduce the number of final rules), the former is quite unexpected since XCSF has shown competive performance on problems of similar or higher difficulty in the past [e. g. 37]. The best-performing of the three model families in terms of MAE are DTs but our MRSL is a rather close runner-up. With respect to the number of rules, there can be made out no clear-cut winner.

7 FUTURE WORK

There are many opportunities for future work. The one most relevant to the present paper is a deeper investigation into why crossover yields higher rule counts. Furthermore, the presented algorithm needs to be benchmarked against more algorithms such

[10] Note that compaction would very likely further degrade MAE performance.

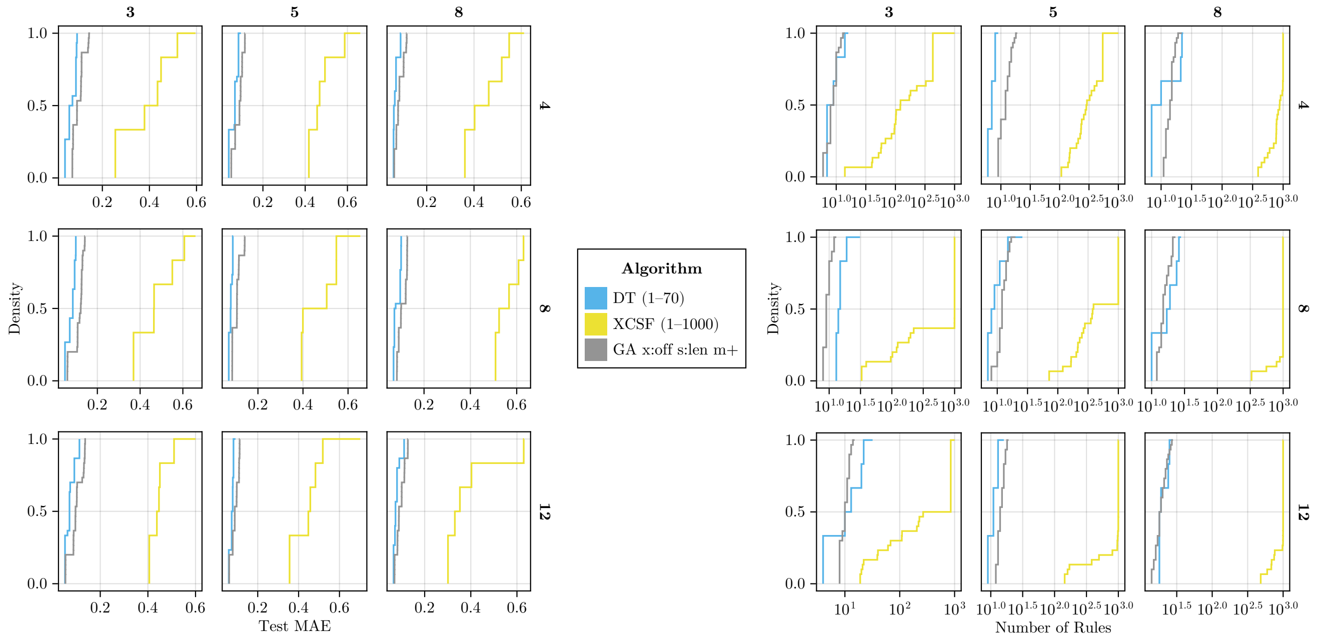

Figure 2: Comparison of DT, XCSF and our MRSL algorithm. For DT and XCSF, the number ranges in the legend indicate the allowed range of rule set sizes. Left: Empirical cumulative distributions of test Mean Absolute Error (MAE) when pooling all the runs of each algorithm variant (i. e. pool over 30 runs—5 repetitions and 6 data sets) for each combination of dimensionality $\mathcal{D}_\mathcal{X}$ (columns) and number of rules K (rows). Right: Empirical cumulative distributions of number of rules in the final solution when pooling the same way; note the log axis.

as SupRB [19]. We also indend to extend our Monte-Carlo sampling–based analysis of our initialization operator to other operators in order to understand better their hyperparametrizations and improve overall performance of this and similar systems. While we do have preliminary results, a dedicated analysis of the merits of our initialization scheme (for example, a proper study comparing it with completely random initialization and an investigation of the effects of initial coverage rates different from 90 %) is also pending.

8 SUMMARY AND CONCLUSION

Extending an earlier preliminary study, we provided a more detailed description of a variable-length *Metaheuristic Rule Set Learning* (MRSL) algorithm for regression tasks which is based on a Genetic Algorithm (GA) and uses interval-based conditions, constant local models and inverse variance–based mixing. Fitness is based on the *corrected Akaike Information Criterion*. Experiments were performed on a set of increasingly difficult synthetic learning tasks of different dimensionalities which are, in theory (given enough time, resources and training data), learnable by the algorithms considered.

Previous work established that in the context of the used MRSL algorithm, *not* using crossover performs better in terms of the final model's complexity (as measured by the number of rules) than using any of the two crossover operators tested while all tested variants performed indistinguishably with respect to test Mean Absolute Error. We took a closer look by analysing convergence behaviour and were able to identify a tradeoff induced by the crossover operators: *Without crossover*, models created are more compact and make fewer but larger prediction errors whereas *when using crossover*, models are less compact and make more but smaller prediction errors. With respect to the benefits of length-niching selection, results were unconclusive; its effect of holding off premature convergence to a certain solution length may be too small in the setting considered (premature convergence is maybe not actually a problem in our MRSL algorithm) or the effect only shows itself in fitness values after more iterations than the ones performed for the study. Within the 2000 iterations performed, length-niching selection seems to decelerate fitness convergence and is outperformed in terms of fitness by the variant with tournament selection.

We further compared performance of our MRSL algorithm with two already-known Rule Set algorithms, namely CART Decision Trees and XCSF. Of these two, only Decision Trees performed well on the learning tasks considered whereas XCSF was not able to learn the tasks despite being known to solve tasks of similar difficulty. While Decision Trees outperformed our algorithm in terms of test Mean Absolute error by a close margin, they perform equally in terms of model complexity. Overall, despite using mostly existing metaheuristic operators that were agnostic to the fact that the domain was Rule Set learning, the variable-length GA performed well. We expect that operators that use more information (e. g. amount of rule overlap and local rule error) perform even better.

REFERENCES

[1] Hirotsugu Akaike. 1974. A new look at the statistical model identification. *IEEE Trans. Automat. Control* 19, 6 (1974), 716–723. https://doi.org/10.1109/TAC.1974.1100705

[2] Rafael Alcala, María José Gacto, and Francisco Herrera. 2011. A Fast and Scalable Multiobjective Genetic Fuzzy System for Linguistic Fuzzy Modeling in High-Dimensional Regression Problems. *IEEE Transactions on Fuzzy Systems* 19, 4 (2011), 666–681. https://doi.org/10.1109/TFUZZ.2011.2131657

[3] Michela Antonelli, Pietro Ducange, and Francesco Marcelloni. 2013. An efficient multi-objective evolutionary fuzzy system for regression problems. *International Journal of Approximate Reasoning* 54, 9 (2013), 1434–1451. https://doi.org/10.1016/j.ijar.2013.06.005

[4] Jaume Bacardit. 2004. *Pittsburgh genetics-based machine learning in the data mining era: representations, generalization, and run-time.* Ph. D. Dissertation. PhD thesis, Ramon Llull University, Barcelona.

[5] Jaume Bacardit and Josep Maria Garrell. 2007. Bloat Control and Generalization Pressure Using the Minimum Description Length Principle for a Pittsburgh Approach Learning Classifier System. In *Learning Classifier Systems*, Tim Kovacs, Xavier Llorà, Keiki Takadama, Pier Luca Lanzi, Wolfgang Stolzmann, and Stewart W. Wilson (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 59–79.

[6] Jaume Bacardit and Natalio Krasnogor. 2006. *BioHEL: Bioinformatics-oriented Hierarchical Evolutionary Learning.* http://eprints.nottingham.ac.uk/id/eprint/482

[7] Alejandro Barredo Arrieta, Natalia Díaz-Rodríguez, Javier Del Ser, Adrien Bennetot, Siham Tabik, Alberto Barbado, Salvador Garcia, Sergio Gil-Lopez, Daniel Molina, Richard Benjamins, Raja Chatila, and Francisco Herrera. 2020. Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI. *Information Fusion* 58 (2020), 82–115. https://doi.org/10.1016/j.inffus.2019.12.012

[8] Leo Breiman, J. H. Friedman, Richard A. Olshen, and C. J. Stone. 1984. *Classification and Regression Trees.* Wadsworth.

[9] Kenneth P. Burnham and David R. Anderson. 2004. Multimodel Inference: Understanding AIC and BIC in Model Selection. *Sociological Methods & Research* 33, 2 (2004), 261–304. https://doi.org/10.1177/0049124104268644

[10] David M. Cherba and William Punch. 2006. Crossover gene selection by spatial location. In *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation* (Seattle, Washington, USA) *(GECCO '06)*. Association for Computing Machinery, New York, NY, USA, 1111–1116. https://doi.org/10.1145/1143997.1144175

[11] Oscar Cordón. 2011. 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 (2011), 894–913. https://doi.org/10.1016/j.ijar.2011.03.004

[12] Kenneth A. de Jong, William M. Spears, and Diana F. Gordon. 1994. *Using Genetic Algorithms for Concept Learning.* Springer US, Boston, MA, 5–32. https://doi.org/10.1007/978-1-4615-2740-4_2

[13] Jan Drugowitsch. 2008. *Design and Analysis of Learning Classifier Systems - A Probabilistic Approach.* Studies in Computational Intelligence, Vol. 139. Springer Berlin Heidelberg, Berlin, Heidelberg.

[14] Guangzhe Fan and J. Brian Gray. 2005. Regression Tree Analysis Using TARGET. *Journal of Computational and Graphical Statistics* 14, 1 (2005), 206–218. https://doi.org/10.1198/106186005X37210

[15] Michael Heider, David Pätzel, Helena Stegherr, and Jörg Hähner. 2023. *A Metaheuristic Perspective on Learning Classifier Systems.* Springer Nature Singapore, Singapore, 73–98. https://doi.org/10.1007/978-981-19-3888-7_3

[16] Michael Heider, Helena Stegherr, Richard Nordsieck, and Jörg Hähner. 2023. Assessing Model Requirements for Explainable AI: A Template and Exemplary Case Study. *Artificial Life* 29, 4 (2023), 468–486. https://doi.org/10.1162/artl_a_00414

[17] Michael Heider, Helena Stegherr, David Pätzel, Roman Sraj, Jonathan Wurth, Benedikt Volger, and Jörg Hähner. 2023. Discovering Rules for Rule-Based Machine Learning with the Help of Novelty Search. *SN Computer Science* 4, 6 (12 Oct 2023), 778. https://doi.org/10.1007/s42979-023-02198-x

[18] Michael Heider, Helena Stegherr, Roman Sraj, David Pätzel, Jonathan Wurth, and Jörg Hähner. 2023. SupRB in the context of rule-based machine learning methods: A comparative study. *Applied Soft Computing* 147 (2023), 110706. https://doi.org/10.1016/j.asoc.2023.110706

[19] Michael Heider, Helena Stegherr, Jonathan Wurth, Roman Sraj, and Jörg Hähner. 2022. Separating Rule Discovery and Global Solution Composition in a Learning Classifier System. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion* (Boston, Massachusetts) *(GECCO '22)*. Association for Computing Machinery, New York, NY, USA, 248–251. https://doi.org/10.1145/3520304.3529014

[20] Clifford M. Hurvich and Chih-Ling Tsai. 1989. Regression and time series model selection in small samples. *Biometrika* 76, 2 (06 1989), 297–307. https://doi.org/10.1093/biomet/76.2.297

[21] Yaochu Jin and Bernhard Sendhoff. 2008. Pareto-Based Multiobjective Machine Learning: An Overview and Case Studies. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 38, 3 (2008), 397–415. https://doi.org/10.1109/TSMCC.2008.919172

[22] Amiram Moshaiov, Yosef Breslav, and Eliran Farhi. 2021. Multi-Modal Multi-Objective Evolutionary Optimization for Problems with Solutions of Variable-Length. In *2021 IEEE Congress on Evolutionary Computation (CEC)*. 1193–1200. https://doi.org/10.1109/CEC45853.2021.9504705

[23] Rafael S. Parpinelli, Heitor S. Lopes, and Alex A. Freitas. 2002. An Ant Colony Algorithm for Classification Rule Discovery. In *Data Mining*. IGI Global, 191–208. https://doi.org/10.4018/978-1-930708-25-9.ch010

[24] David Pätzel and Jörg Hähner. 2022. The Bayesian learning classifier system: implementation, replicability, comparison with XCSF. In *Proceedings of the Genetic and Evolutionary Computation Conference* (Boston, Massachusetts) *(GECCO '22)*. Association for Computing Machinery, New York, NY, USA, 413–421. https://doi.org/10.1145/3512290.3528736

[25] David Pätzel, Michael Heider, and Jörg Hähner. 2023. Towards Principled Synthetic Benchmarks for Explainable Rule Set Learning Algorithms. In *Proceedings of the Companion Conference on Genetic and Evolutionary Computation* (Lisbon, Portugal) *(GECCO '23 Companion)*. Association for Computing Machinery, New York, NY, USA, 1657–1662. https://doi.org/10.1145/3583133.3596416

[26] David Pätzel, Richard Nordsieck, and Jörg Hähner. 2024. Length-niching Selection and Spatial Crossover in Variable-length Evolutionary Rule Set Learning. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion* (Melbourne, VIC, Australia) *(GECCO '24 Companion)*. Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/3638530.3654308

[27] Richard John Preen and David Pätzel. 2023. *XCSF.* https://doi.org/10.5281/zenodo.8193688

[28] David Pätzel. 2024. *dpaetzel/RSLModels.jl: v0.2.0.* https://doi.org/10.5281/zenodo.10955477

[29] David Pätzel. 2024. *dpaetzel/RunRSLBench.jl: v0.1.1.* https://doi.org/10.5281/zenodo.11143388

[30] David Pätzel. 2024. *Learning tasks and result data for the 2024 IWERL@GECCO paper A Closer Look at Length-niching Selection and Spatial Crossover in Variable-length Evolutionary Rule Set Learning.* https://doi.org/10.5281/zenodo.11143818

[31] Matt Ryerkerk, Ron Averill, Kalyanmoy Deb, and Erik Goodman. 2020. A novel selection mechanism for evolutionary algorithms with metameric variable-length representations. *Soft Computing* 24, 21 (01 11 2020), 16439–16452. https://doi.org/10.1007/s00500-020-04953-1

[32] Matt Ryerkerk, Ronald C. Averill, Kalyanmoy Deb, and Erik D. Goodman. 2019. A survey of evolutionary algorithms using metameric representations. *Genet. Program. Evolvable Mach.* 20, 4 (2019), 441–478. https://doi.org/10.1007/s10710-019-09356-2

[33] Matthew L. Ryerkerk, Ronald C. Averill, Kalyanmoy Deb, and Erik D. Goodman. 2017. Solving metameric variable-length optimization problems using genetic algorithms. *Genetic Programming and Evolvable Machines* 18, 2 (01 Jun 2017), 247–277. https://doi.org/10.1007/s10710-016-9282-8

[34] Ben Sadeghi, Poom Chiarawongse, Kevin Squire, Daniel C. Jones, Andreas Noack, Cédric St-Jean, Rik Huijzer, Roland Schätzle, Ian Butterworth, Yu-Fong Peng, and Anthony Blaom. 2022. *DecisionTree.jl - A Julia implementation of the CART Decision Tree and Random Forest algorithms.* https://doi.org/10.5281/zenodo.7359268

[35] Yu Shi, Jian Li, and Zhize Li. 2019. Gradient Boosting With Piece-Wise Linear Regression Trees. arXiv:1802.05640 [cs.LG]

[36] Patrick O. Stalph and Martin V. Butz. 2012. Guided Evolution in XCSF. In *Proceedings of the 14th Annual Conference on Genetic and Evolutionary Computation* (Philadelphia, Pennsylvania, USA) *(GECCO '12)*. Association for Computing Machinery, New York, NY, USA, 911––918. https://doi.org/10.1145/2330163.2330289

[37] Anthony Stein. 2019. *Interpolation-Assisted Evolutionary Rule-Based Machine Learning - Strategies to Counter Knowledge Gaps in XCS-Based Self-Learning Adaptive Systems.* Doctoral Thesis. Universität Augsburg.

[38] Nariaki Sugiura. 1978. Further analysis of the data by Akaike's information criterion and the finite corrections: further analysis of the data by Akaike's. *Communications in Statistics-theory and Methods* 7, 1 (1978), 13–26.

[39] Ryan J. Urbanowicz and Will N. Browne. 2017. *Introduction to Learning Classifier Systems.* Springer. https://doi.org/10.1007/978-3-662-55007-6

[40] Ryan J. Urbanowicz and Jason H. Moore. 2009. Learning Classifier Systems: A Complete Introduction, Review, and Roadmap. *Journal of Artificial Evolution and Applications* 2009 (2009).

[41] Stewart W. Wilson. 1995. Classifier Fitness Based on Accuracy. *Evolutionary Computation* 3, 2 (1995), 149–175.

[42] Stewart W. Wilson. 2002. Classifiers that approximate functions. *Natural Computing* 1, 2 (01 6 2002), 211–234. https://doi.org/10.1023/A:1016535925043