Exploring Self-Adaptive Genetic Algorithms to Combine Compact Sets of Rules

1st Michael Heider
Organic Computing Group
University of Augsburg
Augsburg, Germany
0000-0003-3140-1993

2nd Maximilian Krischan
Organic Computing Group
University of Augsburg
Augsburg, Germany
maximiliankrischan@live.com

3rd Roman Sraj
Organic Computing Group
University of Augsburg
Augsburg, Germany
roman-alexander.sraj@uni-a.de

4th Jörg Hähner
Organic Computing Group
University of Augsburg
Augsburg, Germany
0000-0003-0107-264X

***Abstract*—Rule-based machine learning (RBML) models are often presumed to be very beneficial for tasks where explainability of machine learning models is considered essential. However, their models are only really explainable as long as their rule sets are compact. This leads to the need for an optimizer to take prediction error and rule count as objectives. Given the highly complex fitness landscape of rule set learning tasks, good hyperparameters of the optimizer as well as their robustness against local minima is detrimental.**

In this paper, we explore the use of four self-adaptive genetic algorithms (SAGAs) for the optimization of a recent evolutionary RBML system to reduce the number of hyperparameters to tune and hopefully find better minima. To evaluate the advantages, we benchmark against a non-adaptive genetic algorithm (GA) on five real-world data sets. We find—with the support of a rigorous statistical analysis—that some of the SAGAs deliver a suitable alternative, which is easier to handle for non-experts in GA configurations. This is crucial for a wider application of this RBML method.

***Index Terms*—evolutionary machine learning, explainability, self-adaptation, learning classifier systems**

I. Introduction

With the increasing use of machine learning for decision making, situations were stakeholders demand insights into why certain decisions are made become more common. While some stakeholders might be satisfied with exemplary explanations or more palatable statistical analyses, others will request the ability to look deeper into a machine learning models' core in order to be satisfied with its decisions [1].

Rule-based machine learning (RBML) models are generally considered to be on the side of easier to explain models. However, they are limited by key factors, such as the number of rules, their individual complexity and their interaction to make predictions, cf. [2]. One subset of RBML algorithms is called Learning Classifier Systems (LCSs) [3], [4]. This family of algorithms trains a set of rules that together approximate an unknown function. These rules are of an if-then structure, so that they partition the input (or feature) space and assign individual local models to each such partition. An important distinction to tree-based RBML models is that LCSs allow rules to overlap. If multiple rules are assigned to the subspace where a specific data point lies, i.e. they "match" this data point, a strategy to combine their individual predictions, called mixing, is used. An often used mixing strategy is to assign weights to individual rules, e.g. based on their fitness, and to build a weighted average (for regression) or perform majority voting (for classification or reinforcement learning) accordingly. Typically, LCSs prominently feature the use of metaheuristics (commonly, evolutionary algorithms) to optimize their model structure, i.e. to determine the bounds of the rules that make up the model and how many rules to use. They have been used for all major learning paradigms in the past and feature a long lasting [4] and still active [5]–[7] research community.

A recently proposed LCS is the Supervised Rule-based learning System (SupRB) [8]. The core feature of this LCS is the separation of *rule discovery* from the construction of the prediction model (Heider et al. [8] also refer to this as *solution composition*), which can aid exploration and allows it to construct substantially more compact models than established LCS such as the well-known XCSF classifier system [9] while maintaining competitive prediction errors [10]. For more details on SupRB, see Section III.

One drawback we found is that, due to the separation of rule discovery and solution composition into two independent metaheuristics, SupRB has a lot of parameters to tune. This is a difficult task for non-experts in GAs (and even for them it is non-trivial) or would require a considerable amount of compute for automatic tuning. Therefore, in this paper, we will investigate the adaptation of four approaches for self-adaptive genetic algorithms (SAGAs; cf. Section IV) from literature into the solution composition component of SupRB. We test the four new SupRB variants on five different real-world datasets for regression and compare them to the originally used genetic algorithm (GA). Details on the experiment setup can be found in Section V-A while Section V-B features the results of our benchmarking. We then discuss these results and perform a rigorous statistical analysis in Section V-C.

II. Related Work

The use of self-adaptive operators within the metaheuristic optimizer of LCSs has been an object of research for decades: Hurst and Bull [11] did build a fully self-adaptive ZCS [12] (the LCS on which the more well-known XCS [13] is based). They found that the self-adaptive parameters resulted in better performance over static ones even in stationary environments

but especially in dynamic settings. Hurst and Bull [14] also tested a self-adaptive mutation and learning rate in XCS and showed that it improves XCS's poor performance on long action chain environments. Bull and Hurst [15] furthermore used self-adaptive parameters for an LCS using small neural networks as the individual rules and showed that this is beneficial on different variants of maze running. Unold [16] investigated whether self-adaptive mutation benefits the rule discovery process of XCS and showed that it does for multiplexer tasks.

III. The Supervised Rule-based learning System

The Supervised Rule-based learning System (SupRB) [8] is a rather new LCS that involves two alternating phases of optimization. The training process initially starts with an empty population. Subsequently, in the first phase, called *rule discovery*, independent runs of $(1, \lambda)$ evolution strategies (ESs) each perform a search for a new rule, with the volume a rule encompasses and its in-sample error serving as the fitness of individuals. The initial candidate of this search is placed at one data point (matching only this point) from the training data for which the current model had a high prediction error. It is selected randomly based on an error-weighted roulette wheel selection. Mutation only allows the outward shifting of rule bounds according to a half-norm distribution. This seems to make for a more stable search [10]. After termination of the ESs, these rules are added to a pool of rules, concluding the first phase.

Then, in the second phase, called *solution composition*, the current pool is used to compose a compact yet accurate model based on a subset of the rules within the pool. The fitness is based on a combination of in-sample prediction error and the number of rules. If multiple rules out of a subset match an example/data point, the prediction of this model is based on a weighted average of the individual predictions. Making predictions this way is called mixing. The weights are determined by the error of a rule and how many training examples it was given to fit its local model. The assumption is that rules that saw very few data points should be deemed unreliable and that rules with bad predictions should also not partake in a model's predictions.

In its original version, a GA is used for solution composition and its fittest individual (one of the elitists) is returned to the rule discovery component or the user if training is completed. Wurth et al. [17] proposed to replace the GA with other common metaheuristics and found that there is some problem dependence but the GA is generally a good choice. In this paper, we partially replace the GA with four self-adaptive GAs but retain the remaining mechanisms of SupRB, including the fitness during solution composition, the described mixing approach, and the ES for rule discovery.

As we plan to benchmark the adapted SupRB on real-valued regression tasks (cf. Section V-A), rules need to allow real-valued inputs and outputs. Rule conditions (the if-part which assigns responsibilities to rules) are based on hyperrectangles using upper and lower bounds [18], as these are among the easiest to explain options for such rules and we also deem the explainability of LCS models as their core feature over possibly more-potent machine learning approaches. Local models of rules (the then-part which actually makes the prediction) use linear models. Constant models are often too inaccurate in their predictions and therefore inappropriate for complex real-world datasets and higher-order models are, again, difficult to interpret or even explain to non-experts. To fit the linear local models we use linear least squares with L2-normalization.

IV. Self-adaptive GAs

In this section, we discuss the four SAGAs we adapted to be fit for the solution composition task in SupRB. To make this paper easier to read, we decided to number them, even if in their original paper, they were given a name. This also reflects the fact that we had to do some changes to make them suitable.

a) SAGA1: This GA [19] dynamically adapts crossover and mutation rates to keep the genetic diversity in the population high. Diversity is determined by the coefficient of average solution fitness to the maximal fitness within this population. Crossover and mutation rates are adapted inversely. The more diverse a population is, the higher crossover and the lower mutation will be.

b) SAGA2: Sun and Lu [20] adjust crossover and mutation rates according to the diversity of the current population and the similarity between the current and the preceding generation. First, SAGA2 adapts the bounds available for both rates on a global level and then individual rates are set dynamically between these bounds. Diversity is measures based on minimal, maximal, and mean fitness. Similarity operates directly on the sets of fitness values. Based on a combination of diversity and similarity, bounds for mutation and crossover rates are increased or decreased, similar to SAGA1. Individual crossover rates are calculated for each operation based on the overall bounds and the relative fitness of two selected parents (using the mean of both) in comparison to the mean, min and max fitnesses within the current population. A similar process is used for mutation.

c) SAGA3: Kivijärvi et al. [21] assign each individual new parameters for their respective mutation rates, crossover operators, and a noise factor. We make a small adjustment for SAGA3, in that we do not need the noise factor and instead introduce a crossover rate behaving similarly to the mutation rate. Parameters are generally propagated via crossover and mutated along with the other parameters of an individual. After selecting the parents, their bookkeeping parameters are recombined by randomly selecting one of the parents' parameters. Then, the new child's parameters are mutated according to a fixed probability of 0.05 as suggested by Kivijärvi et al. [21]. Afterwards, crossover (of both parents) and mutation are applied to the chromosome as per the current parameters of the new child.

d) SAGA4: In this approach, inspired by [22], we adapt only the population size rather than other parameters. They

also adapted mutation and crossover rates, but their experiments were not as promising as the population adaptivity. Instead of performing a generational replacement in the GA, each individual receives an "age" attribute, which gets reduced by one in each generation until the individual is eliminated by dropping to zero. The age attributed is altered based on fitness, i.e. individuals with a fitness better than the median receive an extra generation to live, and overall size, i.e. if the population becomes very large (10 times the original size) all individuals below a dynamic fitness threshold loose two generations. All individuals start with an age of three.

V. Evaluation

In this section, we first present the experimental setup including the tested dataset (cf. Section V-A), then display the results (Section V-B) and, finally, perform an extensive statistical analysis (Section V-C).

A. Experiment Setup

The code of our implementation of the self-adaptive GAs as well as our experiments will be submitted for merge into the respective official SupRB repositories[1]. In general, our setup follows the general outline as described in [10].

The (non-adaptive) hyperparameters of our algorithms or the GA, respectively, and SupRB's remaining configuration options were tuned for each learning task using a Tree-structured Parzen Estimator in the Optuna framework [23] that optimizes average solution error[2] on 4-fold cross-validation of training data. Our final evaluation uses eight different splits of training and test data, where training data was always one fourth of the data set. We evaluate each GA with eight different random seeds for each train-test-split, resulting in a total of 64 runs.

We perform 32 iterations of SupRB, where each rule discovery phase produces four rules with an $(1, 20)$-ES. This ES is allowed to run until it did not find a new better rule for 146 generations (which is the default of SupRB). We automatically tuned the σ of the ES's mutation and the fitness weight between the objectives of large and accurate individual rules. The remaining options for the ES are statically configured as described above. Each SAGA has slightly different parameters which need to be configured but, in general, we tune the number of generations, the selection operator and (for all but SAGA3) the crossover operator to use. We crucially also allowed the tuner to increase the number of generations in the expectation that self-adaptivity can sometimes take longer to arrive at a point of balance as extensive prior tuning. The GA was tuned according to the guidelines given by Heider et al. in the official repository.

All algorithms received equal tuning budgets in terms of CPU hours and memory.

[1]https://github.com/heidmic/suprb for the algorithm itself and https://github.com/heidmic/suprb-experimentation for the experimental setup.

[2]Heider et al. [10] did tune for solution fitness but on the Pareto-front of error and rule set size we deem error vastly more important for hyperparameter selection.

Furthermore, SupRB expects inputs in an $[-1, 1]$ interval and its targets to be standardized (zero mean and unit variance).

Table I: Overview of the datasets we benchmark our proposed SAGAs on.

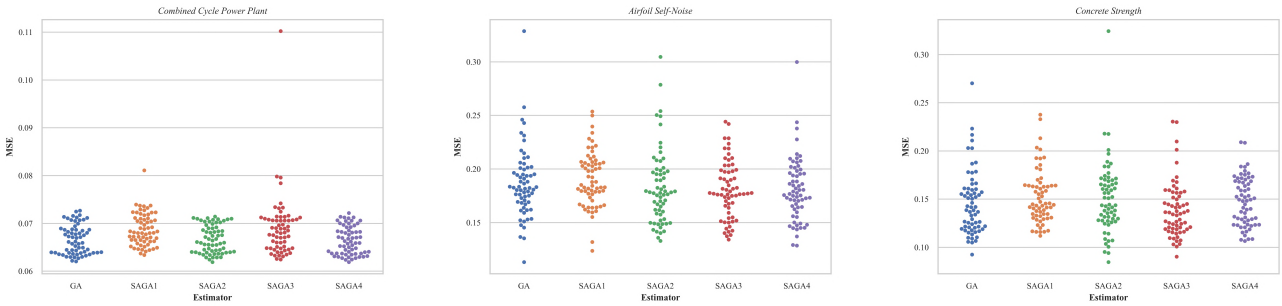
Name	Abbreviation	n_{dim}	n_{sample}
Combined Cycle Power Plant	CCPP	4	9568
Airfoil Self-Noise	ASN	5	1503
Concrete Strength	CS	8	1030
Physicochemical Properties of Protein Tertiary Structure	PPPTS	9	45739
Parkinson's Telemonitoring	PT	18	5875

We test on five datasets part of the UCI Machine Learning Repository [24]. An overview of dimensionalities and sample sizes is given in Table I. The Combined Cycle Power Plant (CCPP) [25], [26] dataset has an almost linear relation between features and targets. Airfoil Self-Noise (ASN) [27] and Concrete Strength (CS) [28] are both highly non-linear and we expect larger solutions being necessary to solve them than for CCPP. ASN has less input features than CS but should be the more difficult task. Physicochemical Properties of Protein Tertiary Structure (PPPTS) [29] is even less linear in its feature-target relations and considerably larger than the others. Parkinson's Telemonitoring (PT) [30] is the dataset with the highest number of features (doubling those of PPPTS) and also non-linear. The first three datasets were also used in [10] and we extend this towards larger (in both number of features and data set size) tasks to better test SupRB's ability on realistic real-world tasks and determine whether our SAGA approaches find good solutions here.

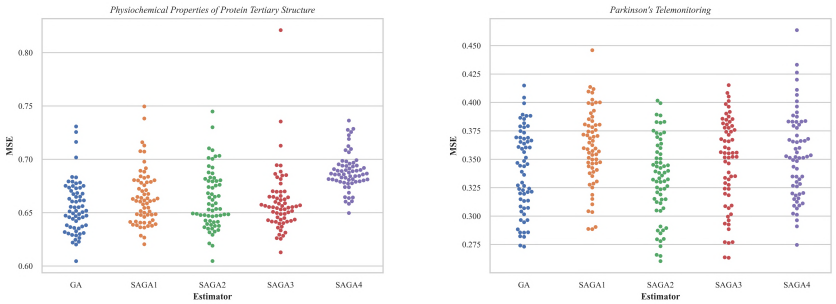
B. Results

From the experiment results, we find that the original GA performs relatively similarly to the four SAGAs. On one hand, this is of course a good result, as it tells us that self-adaptation is not needed (or at least not highly beneficial) to navigate the optimization landscape relatively well and the GA does not fall into the very first (and bad) local optimum. On the other hand, the benefits of using self-adaptivity become less clear at the very first glance. But of course, we still trade in the need to configure a lot of hyperparameters for no (or at least very little) performance loss by making some of the previously static parameters self-adaptive, which can be a great win. However, training got minimally slower with the SAGAs (a summary of a detailed runtime analysis is found at the end of this subsection).

When analysing the distributions of errors the 64 runs produced per dataset, we find that, albeit marginally, the GA, SAGA2, and SAGA3 outperform SAGA1 and SAGA4. Figure 1 displays these distributions in detail, where each dot represents one run's elitist's test scores. All 64 runs per algorithm and dataset were performed with the parameters determined by their individual tuning processes. Optically, it is not easy to make definitive distinctions on performance due to the shapes of the distributions. Other common forms of



(a) Distribution of errors on CCPP (b) Distribution of errors on ASN (c) Distribution of errors on CS



(d) Distribution of errors on PPPTS (e) Distribution of errors on PT

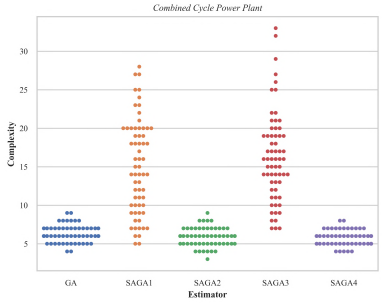
Figure 1: Distribution of runs' errors on the test data. All datasets are standardized with unit variance, therefore, a trivial model should at least achieve an error of 1.0. Note that the plots are on different scales, reflecting the varying difficulty of the learning tasks.

displaying results such as violin or box plots obscure this fact more but we find that this is an important results in itself. The performance of the algorithms is not only based on the data split but also the random seed and—even with self-adaptivity—the evolutionary search is unable to converge on the same value each time (despite the models themselves being deterministic). From exemplary analyses of the pools of rules after training, we assume that this has more to do with the fact that some beneficial rules are missing (or others ill-placed). However, the current solution composition approaches are not free from a small responsibility for the performance measured as they did not always converge optimally, especially during the earlier training stages. An interesting observation is the number of sometimes even relatively strong outliers in performance, where individual runs underperformed vastly compared to their peers.

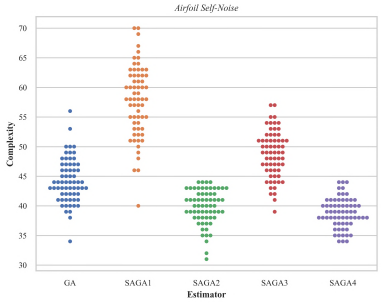
During training, the (SA)GAs had a second objective to optimize besides errors: the number of rules (*complexity*) of their individuals. When it comes to explainability, this is a crucial parameter that determines the usability of a specific model. Arguably, error is more relevant for many applications, but in physical processes we are often confronted with situations where tolerances for prediction errors exist and it is reasonable to assume stakeholders would prefer models they can actually analyse if all models in question perform similarly. In Figure 2, we show complexities per algorithm and dataset in a similar manner to the errors in Figure 1. Note that, in contrast to prediction errors, all complexities have to be integers, which makes these plots more neat looking but has no other direct implications. We can clearly see that SAGA4 is the best overall algorithm for complexity on this selection of datasets. Second best seems to be SAGA2, followed by the GA. SAGA1 and SAGA3 are clearly worse. Especially on PPPTS, SAGA4 seemingly shifted strongly toward substantially more compact solutions instead of more accurate ones. On the other sets, SAGA4 showed vaguely similar errors while still maintaining at least a small lead in terms of model complexity.

We did also measure runtimes for individual training and evaluation runs in isolated environments[3]. We found that, as expected, the GA is the fastest of the five algorithms under investigation, albeit marginally (reducing the runtime by less than 10% when compared to the others in most cases). However, it has more parameters to tune, so as soon as tuning comes into play, the algorithms become much more equal. While specifics were highly dataset dependent, SAGA2 is roughly equal to the others on all but PPPTS, where it took almost 55% longer than the GA. However, on CCPP, where GA and SAGA2 were almost identical in runtime, SAGA3 ran 30% longer than both. On the other datasets, the algorithms were rather close all the time with very low standard deviations (typically between 0.5% and 1.5% of the total runtime). Low variance leads us to assume that runtime

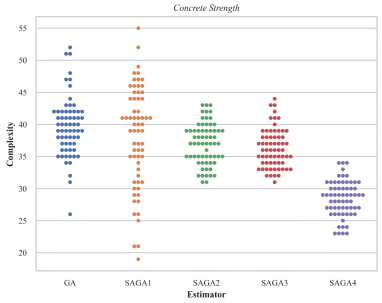
[3]On a Ryzen 5 5600X desktop computer with 32GB RAM and Windows 11 that was not running any other tasks during the experiments.



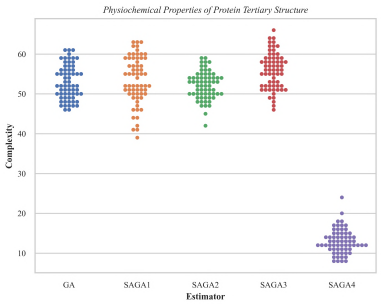
(a) Distribution of complexities on CCPP



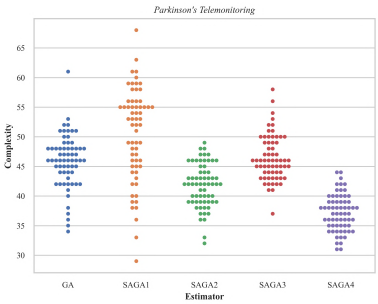
(b) Distribution of complexities on ASN



(c) Distribution of complexities on CS



(d) Distribution of complexities on PPPTS



(e) Distribution of complexities on PT

Figure 2: Distribution of runs' complexities (number of rules in the final elitist). Note that the plots are on different scales, reflecting the varying difficulty of the learning tasks.

is not really dependent on the data split, the random seed of the optimizer or even on the performance of the algorithm, but purely a question of hyperparameters which are the result of task difficulty.

C. Analysis

As the results of our algorithms are not allowing a conclusive decision based on visualization alone, we perform a rigorous statistical analysis to aid us in the decision which algorithm we should select or propose to others to try first when using SupRB on their data. We begin with the application of the model proposed by Calvo et al. [31], [32] to the gathered data. This model gives us the *posterior distribution over the probability of that solution composition method performing best*. Figure 3 displays the results of this analysis. Note that typical minimal thresholds for automated decision making are 80% (or usually even more) probability assigned to a single algorithm [33]. Based on that we can clearly not make a decision regarding which algorithm to choose. However, if we were to pick a top 3, it would be GA, SAGA2, and SAGA3, confirming our results from the visual inspection of results.

When performing the same analysis for complexity (cf. Figure 4), we find that it is much clearer than on MSE, confirming the results of our visual inspection. When choosing one of these algorithms on any dataset similar to those tested, we can expect SAGA4 to produce the smallest model in about 62% of runs. Second best is SAGA2 with about 17% and third is the GA with marginally more than 10% of cases.

Overall, when small models are the goal, we should probably run SupRB with SAGA4 in most cases, however, recall that we are still below the threshold typically used for automated decisions and it seems to underperform in terms of predictive performance (especially on PPPTS, cf. Figure 1d). If we wanted to make a general recommendation based on these results, SAGA2 seems like a plausible top candidate. Even the original GA is not far off and, depending on tuning needs, might be an okay middle ground approach.

To get closer to a decision, we now apply Corani and Benavoli's Bayesian correlated t-test [34] between the algorithm combinations in question. This test compares two solution composition approaches directly and takes into account the probability distribution of the difference in performance, i.e. it tells us how much better we can expect an algorithm to perform on a certain dataset. Note that while we did perform this test for all relevant combinations, we will only show the most interesting ones. Figure 5 shows the results for this test between SAGA2 and the GA on errors. We can clearly see that if one of these algorithms is better on one of the datasets, it is most likely (with 99% probability) better for all runs. However, when accounting for the effect size, this difference is most likely *not practically significant*. Figure 6 shows the results for this test between SAGA4 and the GA on errors. Expected effect sizes, i.e. the expected difference in MSE when comparing both algorithms, are considerably larger than for SAGA2 and GA. However, they seem only practically significant for PPPTS and PT, although, this of course depends

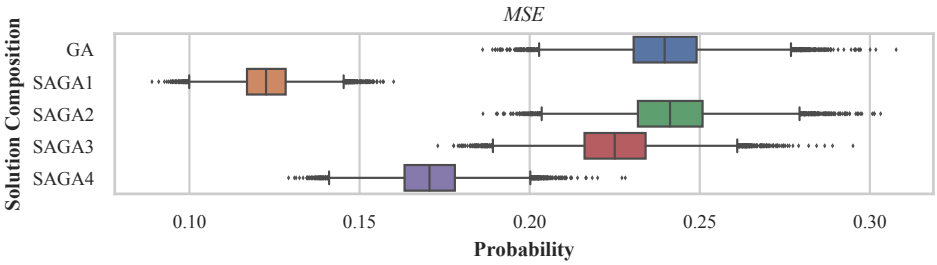


Figure 3: Box plot (with standard 1.5 IQR whiskers and outliers) of the posterior distribution returned by the model of Calvo et al. [31], [32] when applied to the MSE data. A probability value of p% denotes the probability of that solution composition approach performing best with respect to MSE.

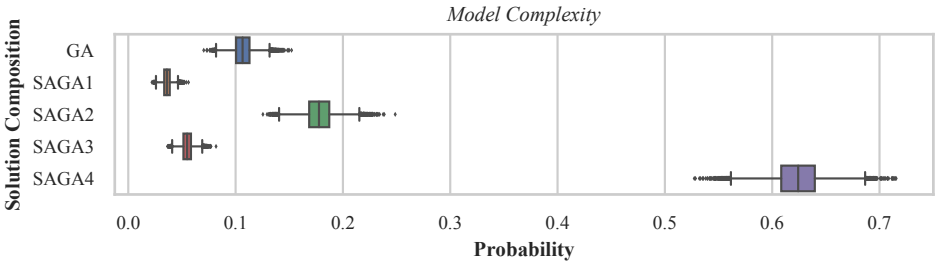


Figure 4: Box plot (with standard 1.5 IQR whiskers and outliers) of the posterior distribution returned by the model of Calvo et al. [31], [32], like Figure 3 but applied to the complexity data. Again, a probability value of p% denotes the probability of that solution composition approach performing best with respect to the number of rules of the returned model.

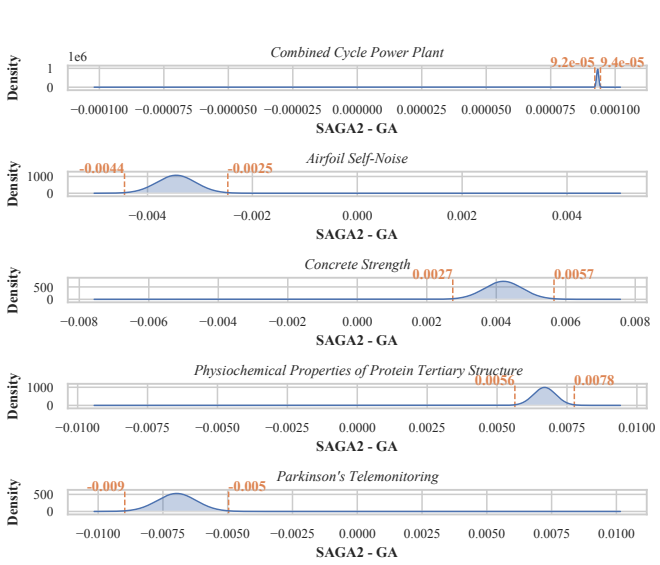


Figure 5: Density plot of the posterior distributions returned by Corani and Benavoli's Bayesian correlated t-test [34] when investigating the difference in MSE between SAGA2 and GA. Probability density to the right equates to the GA having an expected lower error than SAGA2 and vice versa. Orange dashed lines and numbers indicate the 99% HPDI (i.e. 99% of probability mass lies within these bounds). HPDI bounds rounded to two significant figures.

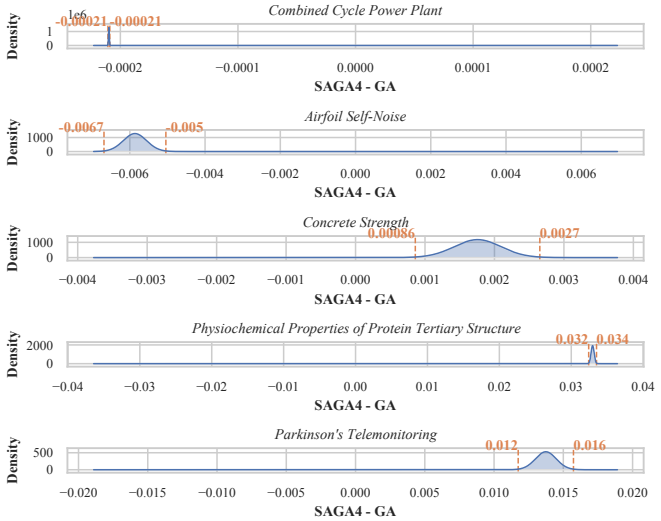


Figure 6: Density plot of the posterior distributions like Figure 5 but for the difference in MSE between SAGA4 and GA. Probability density to the right equates to the GA having an expected lower error than SAGA4 and vice versa.

on the actual application.

Finally, we show an example t-test for the difference in model complexity in Figure 7. We chose SAGA4 as the most compact algorithm and the GA. We add an additional element

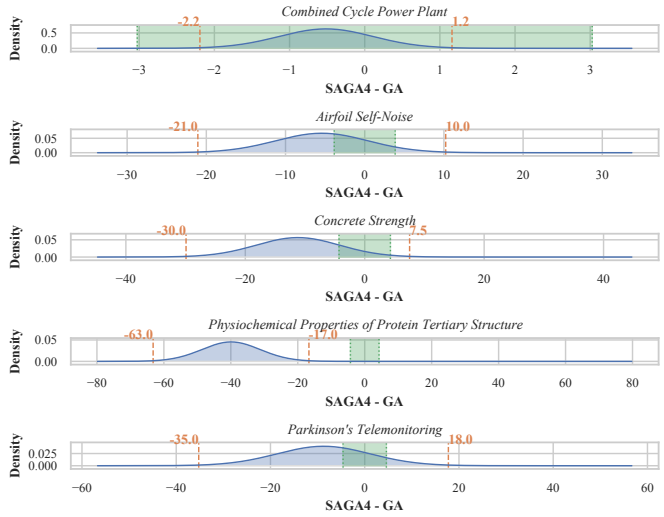


Figure 7: Density plot of the posterior distributions like Figure 5 on the difference in model complexity between SAGA4 and GA. Probability density to the right equates to the GA having an expected lower complexity than SAGA4 and vice versa. The added green area is ± 3 which is the threshold we assume to be when difference in complexity has no meaningful practical implications.

to highlight the area we deem practically insignificant, i.e. if two models are only expected to diverge by 3 in any direction in terms of complexity, they are equally good when it comes to the task of performing manual inspection or similar explainability analyses. The plots show us that we can expect SAGA4's models on PPPTS to be about 40 rules smaller than the GA's, to expect no relevant difference on CCPP, and probably but not definitively smaller models for ASN, CS, and PT.

VI. Future Work

An obvious next step to make SupRB easier to configure would be to introduce self-adaptivity into the rule discovery component. Possible options would be to make the mutation rate or the λ parameter self-adaptive. Additionally, the number of iterations (alternating phases of rule discovery and solution composition) should probably be more dynamic to both prevent stopping too early and also to avoid unnecessary runtime if convergence of the overall system already occurred. Similarly, the solution composition's optimizer should be guided more directly by its state of convergence rather than fixed computation steps. Especially in the last iteration, we want to increase the likelihood of selecting the optimal subset from our pool of rules, but also in previous iterations, better models could help the rule discovery to operate in more relevant regions. While convergence does not give any guarantee that we find a global optimum rather than a local one, we can be relatively sure that an unconverged optimization process should not yet be stopped if compute is available. For the last step, this is especially critical as we might choose that model for deployment in a productive environment. Therefore, it seems plausible to slightly adjust SupRB's flow of things to give increased computation budget (or even use other optimizers) for a final optimization after training is otherwise completed. Here, it might also be beneficial not to use the last elitist again to make it more likely to not run into local minima (should a worse solution be found, we can always take any previous elitist). Another interesting avenue could be to make this final optimization process multi-objective and to let relevant stakeholders decide which of the Pareto-optimal models to deploy.

VII. Conclusion

Rule-base learning algorithms like Learning Classifier Systems (LCSs) are a promising candidate for all applications where explainability is essential. However, due to their evolutionary optimization heuristics, they often feature large numbers of tunable hyperparameters and have little convergence guarantees. Setting hyperparameters that are critical to achieve optimal results is a difficult task, especially for the many data scientists working on everyday applications who are not well-versed when it comes to genetic algorithms (GAs) and their configuration.

In this paper, we chose to introduce different strategies for self-adaptivity into the model construction method of a recent LCS, called SupRB. We adapted four self-adaptive GAs (SAGAs) from literature to be fit for this task and benchmarked them against the GA originally proposed for SupRB on five real-world regression tasks. We chose strategies that adapt mutations and crossovers based on fitness metrics and one strategy to adapt the population size based on relative fitness.

We found that the new strategies are competitive and do outperform the GA in some cases. The GAs runtime is slightly lower but that comes with the tradeoff of more parameters to tune despite receiving largely similar results with tuned parameters. We confirmed our visually-obtained results with an extensive statistical analysis based on Bayesian model analysis techniques and Bayesian correlated t-tests.

References

[1] M. Heider, H. Stegherr, R. Nordsieck, and J. Hähner, "Assessing model requirements for explainable AI: A template and exemplary case study," *Journal of Artificial Life*, 2023, in press.

[2] A. Barredo Arrieta, N. Díaz-Rodríguez, J. Del Ser, A. Bennetot, S. Tabik, A. Barbado, S. Garcia, S. Gil-Lopez, D. Molina, R. Benjamins, R. Chatila, and F. Herrera, "Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI," *Information Fusion*, vol. 58, pp. 82–115, 2020.

[3] M. Heider, D. Pätzel, H. Stegherr, and J. Hähner, *A Metaheuristic Perspective on Learning Classifier Systems*. Singapore: Springer Nature Singapore, 2023, pp. 73–98.

[4] R. J. Urbanowicz and J. H. Moore, "Learning Classifier Systems: A Complete Introduction, Review, and Roadmap," *Journal of Artificial Evolution and Applications*, 2009.

[5] D. Pätzel, A. Stein, and M. Nakata, "An overview of LCS research from IWLCS 2019 to 2020," in *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion (GECCO '20), July 2020, Cancun, Mexico*, C. A. C. Coello, Ed., 2020.

[6] D. Pätzel, M. Heider, and A. R. M. Wagner, "An overview of LCS research from 2020 to 2021," in *GECCO '21: Genetic and Evolutionary Computation Conference, Companion Volume, Lille, France, July 10-14, 2021*, K. Krawiec, Ed. ACM, 2021, pp. 1648–1656. [Online]. Available: https://doi.org/10.1145/3449726.3463173

[7] M. Heider, D. Pätzel, and A. R. M. Wagner, "An overview of LCS research from 2021 to 2022," in *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, 2022, pp. 2086–2094.

[8] M. Heider, H. Stegherr, J. Wurth, R. Sraj, and J. Hähner, "Separating Rule Discovery and Global Solution Composition in a Learning Classifier System," in *Genetic and Evolutionary Computation Conference Companion (GECCO '22 Companion)*, 2022.

[9] S. W. Wilson, "Classifiers that approximate functions," *Natural Computing*, vol. 1, no. 2/3, pp. 211–234, 2002.

[10] M. Heider, H. Stegherr, J. Wurth, R. Sraj, and J. Hähner, "Investigating the Impact of Independent Rule Fitnesses in a Learning Classifier System," in *Bioinspired Optimization Methods and Their Applications*, M. Mernik, T. Eftimov, and M. Črepinšek, Eds. Cham: Springer International Publishing, 2022, pp. 142–156.

[11] J. Hurst and L. Bull, "A self-adaptive classifier system," in *Advances in Learning Classifier Systems*, P. Luca Lanzi, W. Stolzmann, and S. W. Wilson, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 70–79.

[12] S. W. Wilson, "ZCS: A zeroth level classifier system," *Evolutionary Computation*, vol. 2, no. 1, pp. 1–18, 1994. [Online]. Available: https://doi.org/10.1162/evco.1994.2.1.1

[13] ——, "Classifier fitness based on accuracy," *Evolutionary Computation*, vol. 3, no. 2, pp. 149–175, 1995.

[14] J. Hurst and L. Bull, "A self-adaptive XCS," in *Advances in Learning Classifier Systems*, P. L. Lanzi, W. Stolzmann, and S. W. Wilson, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 57–73.

[15] L. Bull and J. Hurst, "A neural learning classifier system with self-adaptive constructivism," in *The 2003 Congress on Evolutionary Computation, 2003. CEC '03.*, vol. 2, 2003, pp. 991–997 Vol. 2.

[16] O. Unold, "Self-adaptive learning classifier system," *Journal of Circuits, Systems, and Computers*, vol. 19, no. 01, pp. 275–296, 2010.

[17] J. Wurth, M. Heider, H. Stegherr, R. Sraj, and J. Hähner, "Comparing different metaheuristics for model selection in a supervised learning classifier system," in *Genetic and Evolutionary Computation Conference Companion (GECCO '22 Companion)*, 2022.

[18] S. W. Wilson, "Mining oblique data with XCS," in *Advances in Learning Classifier Systems*, P. Luca Lanzi, W. Stolzmann, and S. W. Wilson, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 158–174.

[19] J. Vasconcelos, J. Ramirez, R. Takahashi, and R. Saldanha, "Improvements in genetic algorithms," *IEEE Transactions on Magnetics*, vol. 37, no. 5, pp. 3414–3417, 2001.

[20] N. Sun and Y. Lu, "A self-adaptive genetic algorithm with improved mutation mode based on measurement of population diversity," *Neural Computing and Applications*, vol. 31, no. 5, pp. 1435–1443, 2019. [Online]. Available: https://doi.org/10.1007/s00521-018-3438-9

[21] J. Kivijärvi, P. Fränti, and O. Nevalainen, "Self-adaptive genetic algorithm for clustering," vol. 9, no. 2, pp. 113–129, 2003. [Online]. Available: https://doi.org/10.1023/A:1022521428870

[22] M. Smetek and B. Trawiński, "Investigation of genetic algorithms with self-adaptive crossover, mutation, and selection," in *Hybrid Artificial Intelligent Systems*, E. Corchado, M. Kurzyński, and M. Woźniak, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 116–123.

[23] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A Next-generation Hyperparameter Optimization Framework," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, ser. KDD '19. New York, NY, USA: Association for Computing Machinery, 2019, pp. 2623–2631.

[24] D. Dua and C. Graff, "UCI machine learning repository," 2017. [Online]. Available: http://archive.ics.uci.edu/ml

[25] H. Kaya and P. Tüfekci, "Local and Global Learning Methods for Predicting Power of a Combined Gas & Steam Turbine," 2012.

[26] P. Tüfekci, "Prediction of full load electrical power output of a base load operated combined cycle power plant using machine learning methods," *International Journal of Electrical Power & Energy Systems*, vol. 60, pp. 126–140, 2014.

[27] T. Brooks, D. Pope, and M. Marcolini, "Airfoil Self-Noise and Prediction," 1989.

[28] I.-C. Yeh, "Modeling of Strength of High-Performance Concrete Using Artificial Neural Networks," *Cement and Concrete Research*, vol. 28, no. 12, pp. 1797–1808, 1998.

[29] P. Rana, "Physicochemical Properties of Protein Tertiary Structure," UCI Machine Learning Repository, 2013, DOI: https://doi.org/10.24432/C5QW3H.

[30] A. Tsanas, M. Little, P. McSharry, and L. Ramig, "Accurate telemonitoring of parkinson's disease progression by non-invasive speech tests," *Nature Precedings*, pp. 1–1, 2009.

[31] B. Calvo, J. Ceberio, and J. A. Lozano, "Bayesian inference for algorithm ranking analysis," in *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, ser. GECCO '18. New York, NY, USA: Association for Computing Machinery, 2018, pp. 324–325.

[32] B. Calvo, O. M. Shir, J. Ceberio, C. Doerr, H. Wang, T. Bäck, and J. A. Lozano, "Bayesian performance analysis for black-box optimization benchmarking," in *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, ser. GECCO '19. New York, NY, USA: Association for Computing Machinery, 2019, pp. 1789–1797.

[33] A. Benavoli, G. Corani, J. Demšar, and M. Zaffalon, "Time for a change: A tutorial for comparing multiple classifiers through bayesian analysis," *J. Mach. Learn. Res.*, vol. 18, no. 1, pp. 2653–2688, 2017.

[34] G. Corani and A. Benavoli, "A bayesian approach for comparing cross-validated algorithms on multiple data sets," *Machine Learning*, vol. 100, no. 2, pp. 285–304, Sep 2015.