Experimental Investigation on the influence of algorithm configuration towards Symbolic Regression Systems using irace

Suzanne Ong
University of Adelaide
Australia
suzanne.ong@student.adelaide.edu.au

Markus Wagner
University of Adelaide
Australia
markus.wagner@adelaide.edu.au

ABSTRACT

The performance of symbolic regression algorithms can be affected by the change in parameter setting due to algorithm configuration. It involves the tuning of parameters, such as population size and maximum generation value, before running the algorithm to observe the change in performance. In this paper, we review the effect of algorithm configurations on state-of-the-art symbolic regression systems available in terms of computational complexity, runtime, and memory. It explores the impact of the selected algorithm when the parameters are tuned.

1 INTRODUCTION AND MOTIVATION

Symbolic regression is the machine-learning task of finding a mathematical model that represents a closed-form expression that captures the dependencies of variables, either independent or dependent, from a dataset [2,5]. Traditionally introduced as an application of Genetic Programming, or GP [5], it is a popular approach where both the numerical coefficients and structure of the model that consists of symbolic mathematical expressions, are optimized [6,14].

Symbolic regression problems can be solved by tree-based Genetic Programming that evolves around individuals, which attempt to capture all characteristics of a solution, such as the appropriate subset of variables, model structure, and parameters. Individuals are in general manipulated by crossover [3] and mutation, which results in difficulties in generating accurate solutions. This is because evolutionary variations would partially change the model structure and variables, which requires all parameters to be refitted onto the model [4].

There are many ways of measuring an algorithm's performance. In this paper, we will focus on the two concepts, which are, minimizing computational resources consumed by the given algorithm like runtime and maximizing the quality of the algorithm. The behaviour of randomised algorithms can vary significantly between multiple runs based on whether the algorithm is well-tuned. Thus, the same algorithm will not always achieve the same performance, even when run repeatedly with fixed parameters on a single problem. Therefore, it is important to choose parameter settings that minimize some cost statistics of the algorithm's performance across the input data for the best effect [1].

For this project, the algorithm configuration problem we consider can be stated as follows: Given an existing algorithm, a set of input data and a set of parameters for the algorithm, find the optimal parameter values under which the algorithm achieves the best possible performance on the input data. This is because despite being built to solve symbolic regression problems, each algorithm is built differently in terms of its structure and ordering of operations, thus, results may vary in computational complexity depending on the range of the domain given [6]. Hence, to observe the optimal solutions, we reviewed some available, state-of-the-art software packages to search for the optimal algorithm configuration that yields the best performance overall across the benchmark problems. The configurations will be focusing on certain parameters, particularly in population size, the number of generations and tournament size in the same dataset at a constant mutation and crossover rate of symbolic regression problems.

The motivation for this paper is to extend the literature of available symbolic regression algorithms and determine a better optimization in algorithm configuration for each system.

Our research scope will be focusing primarily on testing up-to-date packages with algorithms that are preferably open-sourced, configurable and can be implemented in our research environment. Although there is a boost in research in implementing symbolic regression algorithms for practical applications in fields such as dynamical systems prediction [10], up to date evaluation and analysis on symbolic regression algorithms still holds theoretical significance in deciding the optimal algorithm to use due to the introduction of improved algorithm that helps mitigate the effects of existing limitations like bloating and overfitting [16]. The need to benchmark these algorithms is further emphasised with the recent development of physics-inspired divide and conquer strategy suggested by AI Feynman (2020) [6,14] and hash-based tree approach by Kammerer that aims to optimise search space through optimising coefficients by hashing subtrees with the parent node as the key [2]. The development of new algorithms and software packages that are optimized in efficiency, simplicity, and accuracy will be an interesting research aspect that may generate optimal solutions for symbolic regression problems.

Moreover, the idea of comparing different parameter settings, especially in population size and the number of generations among a variety of algorithms, would provide us with a better insight into the runtime performance, and the quality between different symbolic regressions systems. For instance, assuming we have a population size of 450, will there be any impact on the runtime performance and overall efficiency? Another extension of this idea will be the effect of qualitative parameters on the quality or prediction behaviour of the algorithm, which is useful in identifying the constraints of each selected model on the given problems.

2 LITERATURE REVIEW

A review of research papers was conducted to explore the recent development of automated algorithm configurations, the improvements and limitations of symbolic regression algorithms and identifying available software packages that allow us to test different algorithmic approaches on symbolic regression problems.

Algorithm configuration, also known as parameter tuning, is considered to be a challenging research task when conducted manually due to its expectation of insightful knowledge of the system from the research team and intensive workload. Initially, earlier studies suggested that mathematical approaches and search-based software engineering are used, which failed to identify deep parameters such as a mutation operator, which are unexposed parameters that would impact the variables and expressions in the system. Despite the setbacks, it is regarded as a significant procedure that determines the performance of the algorithms through the change in parameters in a controlled environment [3]. The concept of automating the procedure is emphasised by the introduction of automated algorithmic configuration by Wu via a mutation-based approach, which applies mutation analysis to locate and expose deep parameters based on the expression type. Using four independent systems with Abstract Syntax Tree inputs and reconfiguring the dlmalloc library written in C programming language for memory allocation, allows the identification of deep parameters location to be automated by creating non-identical versions of the program, or mutants, for regression testing [15]. To quote, two significant improvements to note in this research are the reduction of 12% in execution time and 21% in memory consumption [15], which would directly impact the cost statistics in runtime and computational efforts into the system, hence enhancing the system performance [1]. This is significantly higher than the shallow tuning approach, less intensive in labour due to the use of automation, and therefore, making deep parameter extraction a promising method to consider for automated algorithm configuration in systems with symbolic regression algorithms.

Softwares like Eureqa used to be one of the well-known implementations of symbolic regression that is based on coevolution-based fitness prediction. However, a recent study from La Cava suggests the possibility of some established symbolic regression tools like Eureqa being less preferred for benchmarking research due to its inflexibility in front-end API design, making it challenging to make comparisons in its overall computational performance or impact in parameter tuning [16]. Besides, after being acquired by DataRobot, Eureqa's commercial unavailability and the development of other well-performed algorithms have dissuaded researchers from testing further [6,16]. This became relevant in 2020 when the Al Feynman algorithm was introduced. Given the same dataset of mathematical equations, Eureqa had a success rate of 15%, which is 75% lower than Al Feynman's algorithm, as it runs the full equation and expanded its search space, therefore increasing the risk of getting stuck in the optima [14]. One way of analyzing the result is by using Al Feynman's algorithm's feature of simplifying dimensionless variables or constants automatically for each pair, thus reducing the search space and time taken to identify the next expression. This can also be explained due to Al Feynman's neural-network-based approach, which tests on symmetry and separability of the variables. If the function exhibits symmetry (translation, rotational or scaling), it would try replacing all identified pairs into a single variable based on the expression and symbol given. It would then check for separability and split the dataset into two with no common variables if the function can be written as a sum or product of the split datasets. This resulted in the elimination of redundant variables within the equation, which in turn provide a much faster and higher success rate than Eureqa at a controlled number of data points [14,16].

Besides Eureqa's and Al Feynman's methods, several packages allow a direct and indirect Genetic Programming approach for the regression analysis. The first approach involves using Koza's symbolic regression algorithm, also known as the Genetic Programming method provided by the GPLearn package from 2015. It implements the naïve approach of building a population of random formulas, which offers the most flexibility in algorithm configuration but also a higher risk of facing bloat in large program size, which leads to slow computational time. Another manual approach will be the genetic programming algorithm based on the DEAP package, which requires simplification commands from using the sympy package [2]. For a faster approach, the fast function extraction package (FFX) provides the ability to conduct symbolic regression using an elastic net method. It is a deterministic, random search [6] approach that outperformed Koza's Genetic Programming by pruning the basis functions to compact the models. However, its tightly coupled code implementation on its default parameter settings makes it challenging to configure.

Another approach in building predictive models from data sets in software other than symbolic regression will be the application of other regression techniques, such as random forest and support vector regression. To summarise, random forest (RF) is a regression approach that fits several classified regression trees on samples of the training dataset. It is an approach that involves averaging the dataset to mitigate its overfitting problem and its parameters (number of trees in the forest and possible number of dataset features that ensures the optimal splitting) can be tuned using grid search [16]. Other than the randomForest package in R, a well-known implementation of this approach is by importing

RandomForestRegressor class at sklearn.ensemble from the scikit-learn library, which enables the construction of the trees from the given dataset based on the maximum sample size provided [9]. In this method, the grid search would compute the cross-validation score for all the grid points as the program runs to select the best parameter settings in the maximum depth of the tree and the sample size. The package can also be used to construct the support vector machine for regression for analysis [9,11], which makes the package a versatile tool of choice in comparing regression algorithms in terms of its overall performance while the parameter is tuned. Similar to the random forest method, it used grid-search with the difference of using the parameter of the radial basis function kernel, instead of the tree as the affected parameter [16].

La Cava's research over the performance of 14 different symbolic methods and 7 machine learning methods as the baseline algorithm in July 2021 enhances the level of benchmarking standards on optimization algorithms, which makes comparisons on the overall performance of most of the available symbolic regression systems. With the default parameter setting being kept as the controlled variable of the research, the research specifies addressing the observed benefits and constraints of the selected algorithms [6]. This allows additional room for subsequent research projects, as it identifies the limitations of currently approved algorithms and offers a set of statistical measurements, datasets used and results to test for the level of accuracy.

3 METHODOLOGY

This paper proposes an approach to parameter tuning for algorithms used for symbolic regression problems. The motivation to provide additional research literature to both the usage of automated algorithm configuration tools like irace and the optimal parameters for these algorithms. The project should be completed once the following procedures below are accomplished in the given timeframe:

1. Compile datasets with symbolic regression problems for benchmarking
2. Build a prototype of different algorithmic methods in a symbolic regression framework that can successfully run and optimize the constructed dataset
3. Separate each dataset into training, testing and validation sets and determine the fitness of the program.
4. Measure and compare the performance for the three selected systems in a designated range of configurable parameters
5. Evaluate the algorithms runtime performance based on CPU computational time between the time taken to get the best parameter settings for irace and non-irace versions
6. Analyze the quality of the algorithm by assessing the mean absolute error, mean squared error and root mean squared errors
7. Record and present the overall finding, success or unsuccessful result in report and presentation

The final selection of symbolic regression algorithms used are the Symbolic Regression function from GPLearn, Random Forest Regressor and Support Vector Regressor from sklearn package. Additional methods such as TPOT Regressor, GPTIPS, Al Feynman's solution [14] , DEAP and FFX, would not be evaluated in this experiment due to time constraints. It is important to verify the some tools and programming languages, such as R and aclib package, is used and should be compatible for users while conducting the experiments, as aclib enables users to obtain the overall irace operational time in .json format, whereas R is required for the source code of irace package to run in the respective setup.

The project will be deemed successful if the selected method is implemented with results showing a high level of accuracy, as measured against existing tests conducted by La Cava's [6]. However, in situations where there may be indications that a negative result may be obtained regardless of the system's performance. Therefore, it is necessary to implement contingency planning, which is a good practice for any project design. This means that in a scenario where a negative result is obtained, it should not be disregarded and should be presented as a valid conclusion that provides room for additional research in the future. The practice of cherry-picking results in the project should not be practised as well to maintain the level of accuracy and consistency of the project.

The contingency plan of this experiment involves testing on different systems for symbolic regression and automated algorithm configuration. A few examples of the system will be Ellyn, also a Python genetic programming tool that is constructed using C++ underpinning, or a symbolic regression model called QLattice from Feyn to generate the best predictions. In the case of understanding how to automate the algorithms' configuration to best suit these algorithms, such as ParamILS will be the alternative solution. With its similarity in fundamental usage with irace, it could be a useful tool to identify the optimised parameter setting.

4 EXPERIMENTAL SETUP

The experiment uses the benchmark datasets from the Penn Machine Learning Benchmark (PMLB) [16], a repository that provides downloadable datasets for symbolic regression problems, and synthetic benchmarks generated from Keijzer's and Nguyen's benchmarking functions [7]. The repository serves as a centralised platform for convenient access with datasets taken from Al Feynman's and Kaggle, OpenML and UCI in a unified file format where the output to be predicted is renamed as the target for easier usage when we separate the datasets to X for features and Y. For this experiment, datasets are selected and categorized based on its number of features, which is the characteristics in the data that can be measured for the analysis of the actual value of the dependent variable, the number of instances, also known as the observations or rows of the dataset , mathematical implementations and the lack of noise. We chose 10 real-world datasets, 10 datasets by Al Feynman [14] and 10 datasets from Strogatz are taken from the repository,with the addition of the 30 synthetic datasets from Koza's that is reconstructed from the objective functions implemented in Luiz's approach [7] on symbolic regression to have a larger, more diverse range of dataset samples to test on for analysis. The compilation of datasets is dependent on the size and retrieval, as ideally by providing the required number of results in a feasible time, it would mitigate the risk of inaccessibility when tried on different operating systems and devices. This gave us a collection of 80 datasets in total, in which we categorized them into four groups, with one each for real-world dataset, Al Feynman's datasets, Strogatz and synthetic datasets to evaluate the performance of both irace and the algorithms in different dataset sizes. Moreover, using a diverse set of datasets allows us to test the robustness, suitability and potential constraints in memory or time of the tools or algorithms used. Additional information of the datasets selected are attached in Table 1.

Dataset	Number of features	Feature examples	Number of instances
Real-world	[3,100]	Age, Weight, Knee	<10,000
Al-Feynma n	[3,5]	Beta, alpha, theta	100,000
Strogatz	2	x, y	400
Synthetic	[2,4]	x_0, x_1, x_2, x_3	30,000

Table 1: The dataset characteristics based on the number of features, instances and format in each allocated category.

4.1 IRACE

In this experiment, the irace package from R is used to perform the algorithm configurations. Its setup consists of a script called target-runner to evaluate with the given configuration, the parameter text file that contains the definition of parameter ranges, and a scenario file that determines the files for the training and test instances, as well as the tuning budget. To perform the tuning, the package requires a target algorithm, a list of assigned parameters for tuning with their type and range and a set of instances to benchmark the configurations. irace runs in a routine that selects the best configuration among a set of configurations within the range of the defined parameters [11,13]. The chosen configurations are then evaluated using a statistical test of the users' choice iteratively to eliminate configurations that performed poorly based on the ranks. Using an early stopping approach, the irace process focused on the limited budget of evaluation on the most promising candidates, as thorough evaluations will be expensive to be used in poorly performing configurations. The race would continue the process of evaluation and elimination of configurations until the evaluation budget for the race is exhausted or a minimum number of configurations remain. The surviving configurations are then used as seeds to generate a new set of configurations, and these configurations are then served as a benchmark with a new race.

irace is considered for the experiment due to its ability to integrate onto the target algorithm without the need to understand the insights or operations within, as well as its user-friendly interface and well-documented manual [8] that are easily accessible to beginners or experienced users. Its black-box approach of optimizing configurations allows irace to work on the target algorithm without making changes in the source code for each algorithm. This is further supported by irace's recognition in numerous experiments, such as in identifying the best configurations for databases and rainfall predictions [11, 13]. Its advantage over model-based optimization is shown as it determines the impact of parameter configuration without assuming the statistical models [11]. Through its open-sourced examples, it is noted that irace has been used to configure optimization algorithms for NP problems [8]. Its process of considering most considerations based on the ranking is more efficient than searching all possible combinations exhaustively, which is expensive in computational cost and exponential time complexity if the defined parameter range is large. Since the optimal configuration varies in each problem, configurations may be affected by the number of features, the dataset size and overall values. For each iteration, irace will resample best performing configurations and by searching the space of the promising configurations. This is a very efficient process in a situation where a configuration is evaluated on an instance only once, as in practice searching through the entire range of parameters on any application would take up additional time for the evaluation of the configurations.

Algorithms	Parameters	Setup range
Genetic Programming (GPLearn)	Population size	[50,500]
	Generations	[1,100]
	Crossover rate	[0.01, 0.99]
	Parsimony coefficient	[0.01, 1.0]
Random Forest Regressor	Number of estimators	[1,200]
	Maximum depth	[1,20]
	Number of jobs to run	[1,20]
	Verbosity	[0,20]
Support Vector Regressor	Strength of regularization	[1,10,000]
	Maximum Epsilon	[0.1, 100]
	Tolerance of stopping criterion	[0.001,1.0]
	Maximum iterations	[1,1000]

Table 2: Parameter settings for all datasets in irace implementation of scenario.txt

We assigned the algorithm, set the parameters based on the parameters in Table 2, and used the sets of features of the defined datasets as the instances of the configurations. The parameter range values have been set in consideration of two factors, which are the value of the parameters defined by default from the documentations, and the wall time generated by irace on the systems when compiled, in which we selected those no more than 15 minutes as a rule of thumb due to constraints in time. Using the default value, such as 100 for the number of estimators in Random Forest Regressor, the parameter range is allocated between 1 to 200, with the default parameter value as the rounded-up median or to 20 if the value is not given in the documentation. This procedure detects the impact of both increasing and decreasing values of a certain parameter, while also ensuring that there is a sufficient range of combinations for the best parameter settings to be identified from. The remaining parameter range is allocated in the same method given, with the exception of population size due to the high wall time generated by irace. As the population size is defined as the number of programs to run, by implementing a range higher than 500, the wall time of the operation will exceed the time limit. Hence, the population size would remain as [50,500] and compared with a default value of 1,000 in Fig 1. We used the Friedman test of significance (F-test) at a 95% confidence interval as the statistical test to resample and discard the configurations. Using a budget of 1,000 experiments per tuning at maximum, we conducted tests on each dataset and ensured the reliability of the experiment by using the same dataset, hardware and parameter range to obtain the desired results. irace is used to identify configurations that would improve the default settings of the algorithms, and evaluate statistical comparisons on them.

To reduce issues of overfitting, for each dataset, we separate the datasets into training and test instances in irace to optimise the parameters for all algorithms. 7 datasets are chosen as the training data for the black-box and Strogatz's dataset category, whereas 21 from Al Feynman's and synthetic data sets are chosen as the training instances. The remaining data sets were then used for testing in the elitist race. Therefore, this ensures that different inputs are used for optimising as the training set for better prediction on the actual target outputs in the data sets.

4.2 DATA HANDLING

For each dataset assigned, it is split individually into X and Y, with X defined as all the input data of the given features, and Y as the actual value of the dependent variable the algorithm will be predicting. In order to ensure that the algorithms are benchmarked accurately, measures like using the same training, testing and validation set of datasets are implemented for each algorithm using identical operations while splitting the dataset to obtain the estimators. As each dataset consists of a different number of features, observations and variables that need to be predicted, for all compilations, during the pre-processing stage of the experiment, we used the train_test_split function from sklearn.model_selection library to split the datasets into X and Y. X and Y are then randomly shuffled and divided in such that 70% of samples in each of the datasets were allocated for training, leaving the remaining 30% to be used as a testing set. This would ensure that the algorithms compile in a similar experimental setup to mitigate any risk from bias or influences from external factors.

However, evaluating each model's performance using only the training set may lead to overfitting, which increases the possibility of generating the perfect score when compared with the actual value and thus to be rendered as an undesirable estimation when used in real-world applications [4]. To avoid this situation, which may influence the irace configuration based on ranks in this experiment, we divide the training set into ten subsets of validation data, or folds, using K-fold cross-validation. It started off by training the first nine folds and evaluating the tenth in the first iteration and repeating the process until the loop reached ten. At the end of this procedure, the performance on each fold was averaged to obtain the final validation metrics for the model. After the validation is completed, the training set is fitted into the model that is tuned in parameters by irace and evaluates the performance onto the test set. We retained the evaluation score and discarded the model, and allowed irace to iterate continuously until the elite configurations were tested and generated in the terminal.

4.3 PERFORMANCE EVALUATION

Four systems are chosen to evaluate their performance and quality while training datasets and generating the best configurations through automated algorithm configurations, which are Symbolic Regression from GPLearn package, Random Forest Regressor and Support Vector Regression (SVR) from sklearn package. These algorithms are specifically used due to their flexibility in parameter tuning, as these algorithms provide users with the ability to tune parameters such as the subtree mutation rate and stopping criteria without changing the source code implementation and API reference, unlike the algorithm like the FFX and TPOT Regressor packages, which requires users to hard code the implementation due to its fixed parameter settings [6]. Table 1 provides the parameters tuned, as well the setup range for each algorithm to identify the optimal configuration based on performance metrics.

All algorithms are generated and compiled through the Python interface to ensure that the experiment is not affected by external factors, such as the latency of the programming languages used and the wall time when the algorithm is compiled in the Terminal. Therefore, packages such as gptips2 from MATLAB and Operon, a genetic programming framework in C++ are not considered in this experiment.

Once the selected algorithms are configured by irace in Python files, they undergo post-processing in Jupyter Notebook. The results are analysed and deployed as static visualizations for easier evaluation of statistical trends. For the hardware setup, this experiment has been tested at a Mac OS environment, at a RAM of 8GB with M1 as the CPU processor. The experiment requires the installation of Python 3, irace 3.4 and R 4.1.1 to simulate the algorithms to implement.

In this experiment, the performance and quality of the algorithms are determined based on the fitness evaluations on each algorithm and total CPU computational time in seconds, scaled at a logarithmic base of 10.

Fitness evaluation in this experiment involves calculations of the mean absolute error, mean squared error and root mean squared error for each algorithm once the parameters are tuned in irace. The goal is to test the accuracy and fitness level of the predictors to the model among the three designated algorithms as the evaluation of their overall performance. To evaluate the performance of an algorithm on a problem instance, the mean absolute error (MAE), mean squared error (MSE) and root mean square error (RMSE) of the trained algorithms' model on the test set will be evaluated in this experiment.

To recall the concepts of performance metrics [12] , MAE represents the average of the difference between the predicted ($\hat{y}_i$) and original (y_i) values of the dataset. It measures the average of the residuals in the dataset, which is known as the magnitude of errors in a set of dataset. The advantages of using MAE are its robustness to outliers in the dataset and usage to take measurement on the accuracy of the model. However, it is not sensitive to outliers in the real-world datasets and treats both larger and smaller errors equally. Hence, the quality of the algorithm is further supported by the measurement of MSE and RMSE.

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}|$$

MSE is the average of the squared difference between the predicted and original value of the datasets. It provides the variance of the residuals and a good indication of the model performance. MSE is useful in this experiment due to its ability to determine the model performance in datasets which consist of outliers, because the difference between the estimates increases when the errors are squared, hence providing a higher mean in comparison to forecast the actual values based on the data given. Alternatively, the disadvantage of MSE in relation to the experiment is its usability in a scenario where bad predictions generated ruin the model's ability in making predictions. One example to quote would be the occurrence of noises in a dataset, it would be highly biased for higher values and requires the evaluation of RMSE for the model's accuracy.

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y})^2$$

RMSE measures the standard deviation of residuals and can be obtained from getting the squared MSE. Like MAE, it serves as an indicator of the accuracy of the model, but provides a better accuracy level than MAE and MSE in situations where outliers and large errors are present. Instead of averaging the weighted individuals equally, which may create absolute errors that disrupt the model performance, the errors are squared before averaged, therefore assigning higher weight on larger errors to reflect the performance.

$$RMSE = \sqrt{MSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y})^2}$$

A lower value of MAE, MSE and RMSE will be the desirable results in the experiment, as it indicates a better fit of the model and represents the accuracy of the model in the algorithms to predict the actual value. In this experiment, we calculated the MAE, MSE and RMSE to compare and check the results obtained for the model performance between each algorithm. RMSE is chosen as the main performance metric due to its higher sensitivity on large errors than MAE in the datasets and less biased for higher values compared to MSE.

The CPU computational time, which includes the runtime used to split, train and configure into the irace package, are compared between the time taken for configurations to be generated with and without irace. The parameters settings are determined by using RandomSearchCV, a scikit-learn package's implementation on randomized search. It applies the random search operation, which is an exhaustive algorithm for parameter tuning that involves randomly selecting a configuration from the domain of the parameters into a discrete grid. This is followed by the generation of every combination of the values of the grid and calculating performance metrics like root mean squared errors using cross-validation. As one of the evolutionary algorithms for hyperparameter tuning, it is used in all algorithms to compare with irace configurations on the difference in performance via the CPU computational time. By selecting random combinations to train the model and score, it then proceeds to try the selected combination of values of this grid and calculate the performance metrics. It determines which point of the grid maximizes the average value in cross-validation iteratively based on the limit imposed in the number of search iterations, and identifies the optimal combination of values for the hyperparameters. After generating the best parameters shown in Table 2 using its built-in function best_params, we compile both scenarios to calculate the total CPU computational time, which includes the training time, wall clock time, calculated irace operational time that is stored and retrieved using aclib package, and the time taken for the dataset to be validated, fit and computing the calculation of performance metrics in the program.

5 RESULTS

This section briefly depicts the performance metrics and influence in CPU computation time for each algorithm while configured using irace. The optimal parameters generated by irace are presented in tables below, with an example of Keijzer-4 dataset's parameter and overall performance for evaluation.

Algorithms	Parameters	irace	Random Search CV
Genetic Programming (GPLearn)	Population size	311	309
	Generations	46	46
	Crossover rate	0.8	0.77
	Parsimony coefficient	0.89	0.92
Random Forest	Number of estimators	462	497

Regressor	Maximum depth	15	11
	Number of jobs to run	1	3
	Verbosity	15	10
Support Vector Regressor	Strength of regularization	6247.89	9
	Maximum Epsilon	6.36	0.7
	Tolerance of stopping criterion	1.0	0.78673034
	Maximum iterations	239	35

Table 3: Best configuration evaluated by irace and without irace using Keijzer-4 synthetic dataset (Problem instance: $x^3 * e^{-x} * \cos(x) * \sin(x) * (\sin^2(x) * \cos(x) - 1)$) [7]

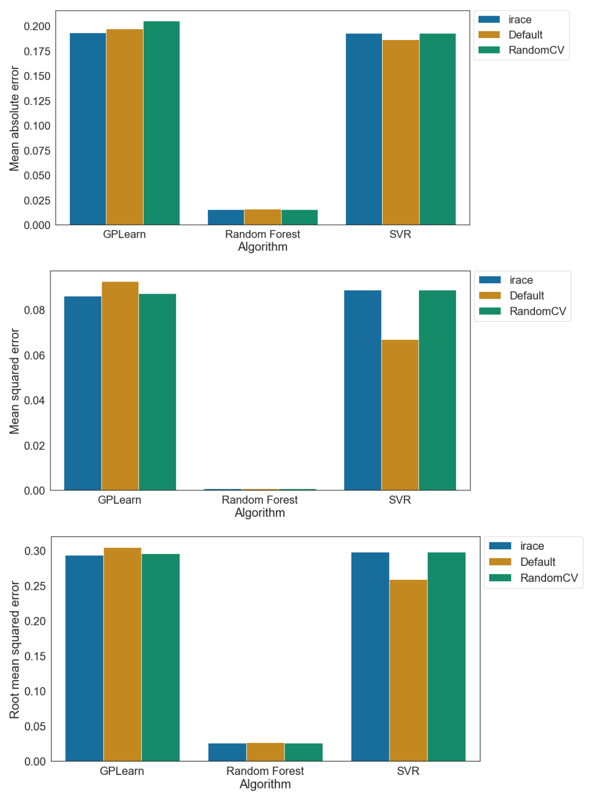

Fig 1: The MAE, MSE and RMSE for Keijzer-4 synthetic dataset as a benchmark to compare the difference in quality among the three algorithms.

Upon observation, Fig 1 indicates that out of the three tested algorithms, Random Forest Regressor consists of the lowest MAE, MSE and RMSE, which highlighted its high level of fitness to the model compared to GPLearn and Support Vector Regressor based on the parameter tuned using values in Table 3. It is also shown that other than the Support Vector Regressor, GPLearn and Random Forest Regressor perform best at irace configured parameters, as both present a lower value in MAE, MSE and RMSE.

Algorithms	Parameters	Real-world	AI Feynman	Strogatz	Benchmarks
Genetic Programming (GPLearn)	Population size	181	181	95	260
	Generation	33	15	47	13
	Crossover rate	0.12	0.07	0.03	0.01
	Parsimony coefficient	0. 80	0.53	0.71	0.61
Random Forest Regressor	Number of estimators	119	174	275	150
	Maximum depth	1	1	1	1
	Number of jobs to run	8	10	2	11
	Verbosity	8	8	8	4
Support Vector Regressor	Strength of regularization	9661.79	9801.99	8265.47	6223.7
	Maximum Epsilon	51.04	91.74	69.36	8.93
	Tolerance of stopping criterion	0.7	0.98	0.78	0.99
	Maximum iterations	21	435	276	236

Table 4: Best irace configuration evaluated for all datasets based on the problem categories

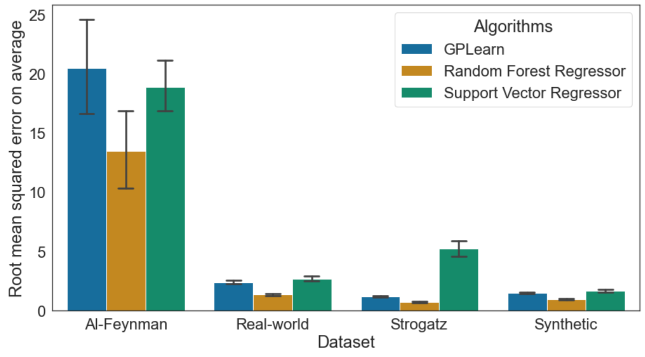

Fig 2: The average RMSE for each dataset category for all algorithms tested. The lines of the bars represented the 95% confidence interval of the mean RMSE generated.

On average, when compared together, the best parameter settings on each algorithm based on the dataset groups are generated in accordance with the irace's output in Table 4. One observation to note would be the difference in parameters in each dataset group in comparison, as well as the best parameter configured in Table 3. Due to the difference in number of features, instances and values on each dataset, the results suggested that optimal parameter settings may vary for each dataset. Thus, it further supported the usage of algorithm configurations, as the tuned parameters would influence the quality of the algorithm while solving symbolic regression problems.

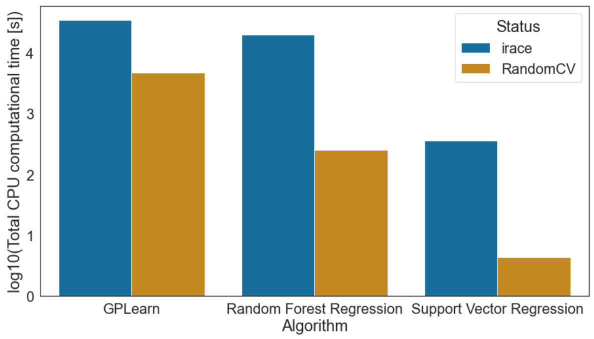

Fig 3: Scaled CPU computational time evaluation in assigned algorithms for both irace and non-irace configurations.

Fig 3 depicts that for all algorithms, irace seems to take up more CPU computational time compared to RandomSearchCV, and therefore, performed slower configurations than random search tuning when 4 selected parameters were tuned. The fast configuration of Random Search CV may also have originated by the limitation imposed in the search iterations, as it only iterates in the number of values in the range without additional configurations, which potentially makes it faster than irace, which iterates continuously on the best configurations until the elite configurations are generated. It may present differently if more than four parameters are tested in both cases.

5.1 STATISTICAL ANALYSIS

The probability that the observed difference of our results occurred by random chance is further evaluated using the measurement of the p-value. For this experiment, we used one-way ANOVA to evaluate the difference in mean between the RMSE of GPLearn, RandomForestRegressor and SVR, while implementing the two-sample t-test for calculating the p-value between the mean CPU computational time in irace and RandomSearchCV, in which both groups are independent to each other. Both operations are conducted by scipy.stats and researchpy packages, with results shown in Table 5.

Statistical tests	P-value at 95% confidence interval
One-way ANOVA to find the difference in mean RMSE for all three algorithms	1.226×10^{-8}
Two-sample t-test to find the difference between the mean CPU computational time for using irace and using RandomSearch CV	0.2215

Table 5: Conducted statistical tests to check if the algorithms statistically outperformed the others and to identify the discrepancy between the time taken to configure with and without irace

Upon observation in Table 5, it is observed that the p-value for the RMSE in the three algorithms is less than 0.01. This means that the null hypothesis, which defined that there is no difference in mean of RMSE between the three algorithms, is rejected. Hence it depicts that there is sufficient evidence to conclude that at least one algorithm performs differently compared to the others. On the other hand, the p-value of 0.2215 suggests that despite irace detrimental performance in computational time compared to RandomSearchCV, we retain the null hypothesis that indicates that there is insufficient evidence to conclude that RandomSearchCV computes faster than irace on average. Therefore, the observation in Table 5 concluded that there is no significant difference in the mean CPU computational time between configurations using irace and using RandomSearchCV.

5.2 GITHUB REPOSITORY

The source code and datasets used can be found at https://github.cs.adelaide.edu.au/a1809907/irace in the name "irace". Operations to run the following statistical tests and visualisations can be found at Post-Processing.ipynb.

6 CONCLUSION

Overall, we conclude that the quality of the algorithms based on the goodness of its fitness varies after its parameters are configured using irace. Random Forest Regressor statistically outperformed the other algorithms in quality for the given datasets, which is further supported by the p-value analysis in Table 5 for having a different level of performance based on RMSE. Our results also identify a slower configuration in irace compared to the Random Search approach. Having four parameters tuned on each algorithm, we observe that Support Vector Regression uses the least amount of CPU computational time compared to the others.

We highly encourage future work to test on other state-of-the-art regression algorithms, such as Ellyn and QLattice, and compare against the three algorithms. This will provide additional resources in determining the effectiveness of these algorithms in solving symbolic regression problems that consist of different numbers of features and observations. It will also be a good alternative to change the experiment setup, such as the hardware and processors, so that the configuration runs in parallel and memory intake is evaluated in larger samples. Furthermore, we will investigate whether the change in parameters after tuning will be statistically significant in enhancing the algorithms' performance. Since we have obtained promising results on how automated algorithm configuration affects the fitness evaluation and how the targeted variables are predicted, additional improvements can be conducted for this study by testing on different parameter ranges for each algorithm for optimal performance. By using additional literatures, datasets and benchmarks available [6, 16], we can enhance the research efforts on the symbolic regression algorithms, as well as existing automated algorithm configuration softwares through an extension of this research. Upon further review, there are some interesting concepts to note, such as the comparison of the best parameter settings for real-world or machine-learning related problems [13] in this experiment for irace in symbolic regression algorithms. This would potentially provide a research gap to benchmark the best parameter settings based on the scale, number of features and the range of the instances, which in turn would optimize the performance of the systems further in terms of its computational cost and quality of the fit. As the results in Fig 1 suggested, although the default settings may not achieve optimal results in some other datasets, it reinforces the need to configure the parameters based on its optimal rank to get the best performance. Therefore, further implementation in research via experimenting on different ranges of parameters will be ideal to get a wider perspective of the performance and efficiency of these algorithms.

REFERENCES

[1] Hutter, F, Hoos, H, Leyton-Brown, K & Stützle, T 2009, 'ParamILS: An automatic algorithm configuration framework', *Journal of Artificial Intelligence Research*, vol 36, pp. 267–306, DOI**:** 10.1613/jair.2861

[2] Kammerer, L, Kronberger, G, Burlacu, B, Winkler, S.M, Kommenda, M & Affenzeller M 2020, 'Symbolic Regression by Exhaustive Search: Reducing the Search Space Using Syntactic Constraints and Efficient Semantic Structure Deduplication', *Banzhaf W, Goodman E, Sheneman L, Trujillo L, Worzel B (eds) Genetic Programming Theory and Practice XVII. Genetic and Evolutionary Computation. Springer, Cham*, DOI:10.1007/978-3-030-39958-0_5.

[3] Karaseva, T, S & Mitrofanov, S A 2020, 'IOP Conference Series: Materials Science and Engineering', vol 862, no. 5, pp. 52-69 , DOI: 10.1088/1757-899X/862/5/052069.

[4] Kommenda, M, Burlacu, B, Kronberger, G. *et al* 2020, 'Parameter identification for symbolic regression using nonlinear least squares', *Genetic Programming and Evolvable Machine***,** pp. 471–501, DOI: 10.1007/s10710-019-09371-3

[5] Koza, J, R 1994, ' Genetic programming as a means for programming computers by natural selection', *Stat Comput* 4, pp. 87–112' , DOI: 10.1007/BF00175355.

[6] La Cava, W , Orzechowski, W, Burlacu, P, França, B, Virgolin, F,O, Jin, M, Kommenda, Y & Moore, M 2021, 'Contemporary Symbolic Regression Methods and their Relative Performance'.

[7] Luiz, V, B, Oliveira, Joao, B, S, M, Luis, F, M & Gisele, L, P, 2018, ‘Analysing symbolic regression benchmarks under a meta-learning approach’, *GECCO’18: Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pp. 1342-1349, DOI: 10.1145/3205651.3208293

[8] Manuel, L, Jérémie, D, Leslie, P, Thomas, S & Mauro, B, 2016, ‘The irace package: Iterated Racing for Automatic Algorithm Configuration’, *Operations Research Perspectives*, DOI:10.1016/j.orp.2016.09.002

[9] Pedregosa, F, Varoquaux, G, Gramfort, A, Michel, V, Thirion, B, Grisel, O, Blonde, M, Prettenhofer, P, Weiss,R, Dubourg, V, Vanderplas, J, Passos, A, Cournapeau, D, Brucher, M, Perrot, M & Duchesnay E 2011, 'Scikit-learn: machine learning in Python', *Journal of Machine Learning Research*, vol.12, pp. 2825–2830.

[10] Quade, M, Abel, M, Shafi, K, Niven, R, & Noack, B.R 2016, 'Prediction of dynamical systems by symbolic regression.' *Physical review,* vol. 94, pp. 1-1, DOI: 10.1103/PhysRevE.94.012214.

[11] Sam, C, Michael, K & Alex, A, F, 2018, ‘Decomposition genetic programming: An extensive evaluation on rainfall prediction in the context of weather derivatives’, *Applied Soft Computing*, Vol. 70, pp. 208-224, DOI: 10.1016/j.asoc.2018.05.016

[12] Shwetha, A, 2016, ‘What are RMSE and MAE? - A simple guide to evaluation metrics’, *Towards data science*, viewed 26 October 2021, <https://towardsdatascience.com/what-are-rmse-and-mae-e405ce230383>

[13] Silva-Muñoz, M, Franzin, A & Bersini, H 2021, ‘Automatic configuration of the Cassandra database using irace’, *PeerJ Computer Science*, DOI: 10.7717/peerj-cs.634

[14] Udrescu, S-M & Tegmark, M 2020, 'AI Feynman: A Physics-Inspired Method for Symbolic Regression', Vol. 6, no. 16, DOI: 10.1126/sciadv.aay2631.

[15] Wu, F, Weimer, W, Harman, M, Jia, Y & Krinke, J 2015, 'Deep Parameter Optimisation', pp. 1375-1382, DOI: 10.1145/2739480.2754648.

[16] Žegklitz, J & Pošík, P 2021, 'Benchmarking state-of-the-art symbolic regression algorithms', *Genetic Programming and Evolvable Machine* , vol. 22, pp. 5–33, DOI: 10.1007/s10710-020-09387-0.