

**Analysis of the use of repulsors to improve
generalization ability in genetic programming:**

An application to symbolic regression problems

Jorge Miguel Silvestre Canelhas

Dissertation presented as a partial requirement for the
degree of Master of Information Management,
specialization in Information Systems and Technologies
Management

NOVA Information Management School
Instituto Superior de Estatística e Gestão de Informação
Universidade Nova de Lisboa

**ANALYSIS OF THE USE OF REPULSORS TO IMPROVE
GENERALIZATION ABILITY IN GENETIC PROGRAMMING,
AN APPLICATION TO SYMBOLIC REGRESSION**

by

Jorge Miguel Silvestre Canelhas

Dissertation presented as partial requirement for obtaining the Master's degree in Information Management, with a specialization in Information Systems and Technologies Management

Advisor: Prof. Leonardo Vanneschi

September 2017

DEDICATION

To Angélica, may you achieve whatever you aspire.

ACKNOWLEDGEMENTS

It would be impossible for me to finish my dissertation without the guidance of my professors, help from colleagues, friends, and support from my family.

I would like to express my deepest gratitude to my advisor, Prof. Leonardo Vanneschi for his guidance and for his classes that proved to be inspirational.

I'd also like to thank my colleagues with whom I've studied and worked, and my friends that supported me in my studies.

Finally, a big thank you to my family for enduring my absence and for supporting me while pursuing this degree.

This page was intentionally left blank.

ABSTRACT

Genetic Algorithms are bio-inspired metaheuristics that solve optimization problems; they are evolutionary algorithms that mimic the biological processes of evolution and natural selection to evolve solutions to a given problem. Genetic programming consists of the creation of programs employing GAs to evolve them. In both GA and GP, the algorithm starts with a random solution to a problem that is improved generation after generation building it on the positive traits of the previous generation and discarding traits that do not improve the solution.

Repulsors consist of giving the learning algorithm some prior knowledge on the outcome of previous generations on a test set, to try to replace solutions that performed poorly on the data set with better ones.

This thesis aims to test and document if the use of repulsors can change the behavior of GP, improve its learning rate and reduce overfitting thus also improving the generalization abilities?

Overfitting is a problem in many machine learning algorithms, genetic programming (GP) is also affected by it, one of the objectives of this dissertation is to assess if overfitting can be reduced by using knowledge on the prior behavior of programs generated by GP on a validation data set, and, applying this knowledge to change the selection phase penalizing solutions similar to those that generalized poorly before. These poorly performing solutions will be called repulsors and are the main topic of this dissertation.

We developed a program that implemented standard and repulsor based genetic programming. The program was then executed several times over some datasets and collect the results. Finally, the results were compared, and conclusions were taken.

The results indicate that the use of repulsors produces better results on the training set and in the test set, this leads us to conclude that the use of repulsors has a positive effect on the performance of GP.

The results indicate that the use of repulsors does indeed produce better results. On the training phase, seven out of the nine datasets showed improved algorithm performance when learning. In the test sets, the algorithm presented better generalization ability on five out of nine datasets.

Studies could be extended to the use of multi-objective optimization when selecting individuals, and the extension of the repulsor list to other (independent) runs with the same parameters and dataset.

KEYWORDS

genetic algorithm; genetic programming; machine learning; repulsor; overfitting; symbolic regression

This page was intentionally left blank.

INDEX

1. Introduction.....	1
1.1. Objective.....	2
1.2. Method.....	2
1.3. Document Structure.....	3
2. Theoretical framework.....	4
2.1. Base Concepts.....	4
2.1.1. Machine Learning.....	4
2.1.2. Optimization Problems.....	4
2.1.3. Symbolic Regression.....	4
2.1.4. Learning Process and datasets.....	5
2.1.5. Overfitting.....	6
2.2. Evolution and Natural Selection.....	7
2.3. Genotype and phenotype (Encoding).....	7
2.4. Genetic Algorithms.....	8
2.4.1. Standard Genetic Algorithm.....	9
2.4.2. Common Genetic Algorithm Parameters.....	9
2.4.1. Common Genetic Algorithm Termination Criteria.....	9
2.4.2. Fitness Evaluation.....	10
2.4.3. Phases of Genetic Algorithms.....	10
2.4.4. Selection phase and selection methods.....	11
2.4.5. Variation phase and variation methods.....	14
2.5. Classic Genetic Programming.....	16
2.5.1. Classic Genetic Programming Algorithm.....	17
2.5.2. Program Representation.....	17
2.5.3. Common GP Parameters.....	19
2.5.4. GP Fitness evaluation.....	19
2.5.5. Population Initialization and initialization methods.....	20
2.5.6. Selection Phase.....	21
2.5.7. Variation Phase (Crossover and Mutation).....	21
2.6. Repulsor Based Genetic Programming.....	23
2.6.1. Repulsors.....	23
2.6.2. Overfit detection.....	23
2.6.3. GP Repulsors Algorithm.....	23

3. Problem Justification and Literature review.	24
4. Objectives and Research Questions.	26
4.1. Objectives.....	26
4.1.1. Main objectives.	26
4.1.2. Additional Objectives.	26
4.2. Research Questions.....	26
5. Methodology.....	27
5.1. Software development.....	27
5.2. Procedure.....	27
5.3. Data collection.....	27
5.4. Run analysis.....	27
6. GENETIC PROGRAMMING Software.....	28
6.1. Main Genetic Programming Software.....	28
6.1.1. Algorithm validation and tests.	28
6.2. Inputs and Outputs.....	29
6.2.1. Inputs.....	29
6.2.2. Outputs.....	29
6.2.1. Data processing and analytics software.....	30
6.3. Implementation and operation.....	31
6.3.1. Primitives/Operators.....	31
6.3.2. Variables.....	31
6.3.3. Constants.....	31
6.3.4. Termination Criteria.....	31
6.3.5. Individual and Population Structure.....	31
6.4. Algorithms.....	32
6.4.1. Standard GP Algorithm Implementation.....	32
6.4.2. Repulsor Based GP Implementation.....	33
6.5. Fitness Evaluation.....	35
6.5.1. Individual fitness on the training set.....	35
6.5.2. Repulsor distance evaluation.....	35
6.6. Overfitting detection method.....	36
6.7. Parameters.....	37
6.8. Dataset preparation and use.....	38
6.8.1. Dataset Descriptions.....	38
6.8.2. Dataset Sizes.....	39

7. Results and discussion	40
7.1. Global Result Analysis.....	40
7.1.1. Training fitness Centrality Measures.....	40
7.1.2. Test fitness Centrality Measures.....	41
7.1.3. Global Centrality Measures comparison.....	42
7.1.4. Best element.	42
7.1.5. Number of Repulsors Found.	43
7.2. Detailed analysis of individual dataset results.	44
7.2.1. Detailed Analysis of the Fittable_sin dataset.....	45
7.2.2. Detailed Analysis of the Fittable_fn05 dataset	49
7.2.3. Detailed Analysis of the Fittable_fn07 dataset	53
7.2.4. Detailed Analysis of the Fittable_fn08 dataset	57
7.2.5. Detailed Analysis of the CCPP Folds dataset.	61
7.2.6. Detailed Analysis of the Concrete Slump Folds dataset.....	65
7.2.7. Detailed Analysis of the Concrete compressive strength dataset.	69
7.2.8. Detailed Analysis of the Wine Quality red dataset.	73
7.2.9. Detailed Analysis of the LD50 Toxicity dataset.	77
8. Conclusions.....	81
8.1. Answers to the Research Questions.....	82
9. Limitations and recommendations for future works	83
10. Bibliography.....	84
11. Appendix.....	86
11.1. Program Source Code	86
11.2. Dataset Descriptions.....	86
11.2.1. Combined Cycle Power Plant data set (CCPP).....	86
11.2.2. Wine Quality Red.....	87
11.2.3. Concrete Slump Test.....	88
11.2.4. Concrete Compressive Strength dataset.....	89
11.2.5. Toxicity dataset.....	89
11.3. Run information	90
11.3.1. Number of runs per dataset table.....	90
11.3.2. Run data.....	90
11.4. Kruskal-Wallis Test detail	91
11.5. Correlation details.....	92
11.5.1. Correlation matrix for run summary.	92

11.5.2.	Correlation Table by dataset.	93
11.6.	Detailed Analysis Additional Information.	94
11.6.1.	Fittable_sin dataset.	94
11.6.2.	Fittable_fn05 dataset.	94
11.6.3.	Fittable_fn07 dataset.	95
11.6.4.	Fittable_fn08 dataset.	95
11.6.5.	CCPP folds dataset.	96
11.6.6.	Concrete slump dataset.	96
11.6.7.	Concrete compressive strength dataset.	97
11.6.8.	Wine quality red dataset.	97
11.6.9.	LD50 Toxicity dataset.	98

LIST OF FIGURES

Figure 1: Equation found by symbolic regression.	4
Figure 2: Concrete slump dataset to be modeled by a symbolic regression produced by GP ..	5
Figure 3: Learning Process.....	5
Figure 4: Dots represent real data, Blue Line overfitting as an over adjustment to the learning data.....	6
Figure 5: Text encoding of a traveling salesman problem solution.	7
Figure 6: Binary representation of a solution.	7
Figure 7: Text representation of RPN individuals	7
Figure 8: Main phases of Genetic Algorithms.	10
Figure 9: Fitness proportional selection probabilities.	11
Figure 10: Example of ranking selection probabilities.	12
Figure 11: Steady state selection(Sivanandam & Deepa, 2008)	13
Figure 12: Variation phase.	14
Figure 13 : Binary Crossover example.....	14
Figure 14: Cycle crossover.....	15
Figure 15: Binary mutation.....	15
Figure 16: Inversion mutation	15
Figure 17: RPN representation of a solution.....	16
Figure 18: Plot of a solution found to approximate the sin function.	16
Figure 19: Representation of an expression	18
Figure 20: Representation of a solution to the sin function in different generations, left: early generation, right: late generation.....	19
Figure 21: On the Left, a tree created with the full algorithm, and on the right, a tree initialized with the growth algorithm.	20
Figure 22: Mutation Example, left: the original tree, middle: point mutation and right: sub-tree mutation.....	21
Figure 23: Crossover Example, above two parents, the selected part is in red and green, below, the siblings with the swapped parts still in their parent's color.....	22
Figure 24: Repulsor based algorithm example.	23
Figure 25:GP Repulsors 08 algorithm simplified diagram.....	33
Figure 26:GP Repulsors 09 algorithm simplified diagram.....	34
Figure 27: Overfit detection at 1.5 times the fitness difference.....	36
Figure 28: Overfit Detection System schematic.....	36
Figure 29: Data set partitioning.	39

Figure 30: Repulsor count correlation Matrix.....	43
Figure 31:fittable_sin, Best Individual Generation.	46
Figure 32:fittable_sin, training set graphical centrality comparison.	47
Figure 33: fittable_sin, test set graphical centrality comparison.....	47
Figure 34: Boxplot of the best fitness on the fittable_sin test set.....	48
Figure 35: Correlation matrix for fittable_sin dataset.	48
Figure 36:fittable_fn05, Best Individual Generation.....	50
Figure 37: fittable_fn05, training set graphical centrality comparison.	51
Figure 38: fittable_fn05, test set graphical centrality comparison.....	51
Figure 39: Boxplot of the best fitness on the fittable_fn05 test set.....	52
Figure 40:Correlation matrix for fittable_fn05 dataset.	52
Figure 41:fittable_fn07, Best Individual Generation.....	54
Figure 42: fittable_fn07, training set graphical centrality comparison.	55
Figure 43: fittable_fn07, test set graphical centrality comparison.....	55
Figure 44: Boxplot of the best fitness on the fittable_fn07 test set.....	56
Figure 45:Correlation matrix for fittable_fn07 dataset.	56
Figure 46:fittable_fn08, Best Individual Generation.....	58
Figure 47: fittable_fn08, training set graphical centrality comparison.	59
Figure 48: fittable_fn08, test set graphical centrality comparison.....	59
Figure 49: Boxplot of the best fitness on the fittable_fn08 test set.....	60
Figure 50: Correlation matrix for fittable_fn08 dataset.	60
Figure 51: CCPP folds, Best Individual Generation.....	62
Figure 52: CCPP Folds, training set graphical centrality comparison.....	63
Figure 53: CCPP Folds, test set graphical centrality comparison.....	63
Figure 54: Boxplot of the best fitness on the CCPP folds test set.....	64
Figure 55: Correlation matrix for CCPP folds dataset.	64
Figure 56: Concrete Slump, Best Individual Generation.	66
Figure 57: concrete slump, training set graphical centrality comparison.	67
Figure 58: concrete slump, test set graphical centrality comparison.....	67
Figure 59: Boxplot of the best fitness on the Concrete Slump test set.	68
Figure 60: Correlation matrix for the concrete slump dataset.	68
Figure 61: Concrete CS, Best Individual Generation.	70
Figure 62:fittable_sin, training set graphical centrality comparison.	71
Figure 63: fittable_sin, test set graphical centrality comparison.....	71
Figure 64: Boxplot of the best fitness on the Concrete CS, test set.	72
Figure 65: Correlation matrix for the Concrete CS dataset.....	72

Figure 66: WQR, Best Individual Generation.	74
Figure 67: WQR, training set graphical centrality comparison.	75
Figure 68: WQR, test set graphical centrality comparison.	75
Figure 69: Boxplot of the best fitness on the wine quality red test set.....	76
Figure 70: Correlation matrix for wine quality red dataset.	76
Figure 71:LD50 toxicity, Best Individual Generation.....	78
Figure 72: LD50, training set graphical centrality comparison.	79
Figure 73: LD50, test set graphical centrality comparison.....	79
Figure 74: Boxplot of the best fitness on the LD 50 Toxicity test set.	80
Figure 75: Correlation matrix for LD50 Toxicity dataset.	80

LIST OF TABLES

Table 1 – Nature/computer terminology mapping.....	8
Table 2 – Fitness proportional selection probabilities.....	11
Table 3 – Rank Selection order example.....	12
Table 4 – RPN processing example.	18
Table 5 – Search matches on Repulsors.....	24
Table 6 – File List.	29
Table 7 –Directory List.....	30
Table 8 –MySQL Table List.....	30
Table 9 – Genetic Program Parameters.	37
Table 10 – Mathematical Function datasets description.....	38
Table 11 –UCI real-life regression datasets description.	38
Table 12 – Mathematical Function datasets sizes.	39
Table 13 – UCI real-life regression datasets sizes.	39
Table 14 – Mean Training Set Fitness at iteration 200.	40
Table 15 –Median Training Set Fitness at iteration 200.	40
Table 16 – Mean Test Set Fitness at iteration 200.....	41
Table 17 – Median Test Set Fitness at iteration 200.....	41
Table 18 – Algorithm where the Best Median was found.	42
Table 19 – Generation where the best solution was found (training set).....	42
Table 20 – Mean number of Repulsors found.	43
Table 21 – Median number of Repulsors found.	43
Table 22 – Correlation analysis field descriptions.....	44
Table 23 – fittable_sin summary statistics for the Training set.....	45
Table 24 – fittable_sin summary statistics for the Test set.	45
Table 25 – Median number of repulsors found, fittable_sin.	46
Table 26 – fittable_sin, Individual depth and size median.	46
Table 27 – Kruskal-Wallis Test for fittable_sin.....	48
Table 28 – fittable_fn05 summary statistics for the Training set.	49
Table 29 – fittable_fn05 summary statistics for the Test set.	49
Table 30 – fittable_fn05, median number of repulsors found.	50
Table 31 – fittable_fn05, individual size and depth Median.....	50
Table 32 – Kruskal-Wallis Test for fittable_fn05.	52
Table 33 – fittable_fn07 Summary statistics for the Training set.....	53
Table 34 – fittable_fn07 Summary statistics for the Test set.	53

Table 35 – fittable_fn07, median number of repulsors found.	54
Table 36 – fittable_fn07, Individual size and depth Median.	54
Table 37 – Kruskal-Wallis Test for fittable_fn07.	56
Table 38 – fittable_fn08 summary statistics for the Training set.	57
Table 39 – fittable_fn08 summary statistics for the Test set.	57
Table 40 – fittable_fn08, median number of repulsors found.	58
Table 41 – fittable_fn08, Individual size and depth Median.	58
Table 42 – Kruskal-Wallis Test for fittable_fn08.	60
Table 43 – CCPP folds, Summary statistics for the Training set.	61
Table 44 – CCPP folds, Summary statistics for the Test set.	61
Table 45 – CCPP folds, Median number of repulsors found.	62
Table 46 – CCPP folds, Individual size and depth Median.	62
Table 47 – Kruskal-Wallis Test on ccpp folds test data set.	64
Table 48 – concrete slump, Summary statistics for the Training set.	65
Table 49 – concrete slump, Summary statistics for the Test set.	65
Table 50 – concrete slump, Median number of repulsors found.	66
Table 51 – concrete slump, Individual size and depth Median.	66
Table 52 – Kruskal-Wallis Test for the concrete slump dataset.	68
Table 53 – Concrete CS, Summary statistics for the Training set.	69
Table 54 – Concrete CS, Summary statistics for the Test set.	69
Table 55 – Concrete CS, Median number of repulsors found.	70
Table 56 – Individual size and depth Median.	70
Table 57 – Kruskal-Wallis Test for the Concrete CS dataset.	72
Table 58 – WQR, Summary statistics for the Training set.	73
Table 59 – WQR, Summary statistics for the Test set.	73
Table 60 – WQR, Median number of repulsors found.	74
Table 61 – Individual size and depth Median.	74
Table 62 – Kruskal-Wallis Test for the WQR dataset	76
Table 63 – LD50, Summary statistics for the Training set.	77
Table 64 – LD50, Summary statistics for the Test set.	77
Table 65 – LD50, Median number of repulsors found.	78
Table 66 – Individual size and depth Median.	78
Table 67 – Kruskal-Wallis Test for the LD50 dataset.	80
Table 68 – Mean number of repulsors found fittable_sin.	94
Table 69 – Individual size fittable_sin.	94
Table 70 – Individual tree depth fittable_sin.	94

Table 71 – Mean number of repulsors found Fittable_fn05.....	94
Table 72 – Individual size Fittable_fn05.....	94
Table 73 – Individual tree depth Fittable_fn05.....	95
Table 74 – Mean number of repulsors found Fittable_fn07.....	95
Table 75 – Individual size Fittable_fn07.....	95
Table 76 – Individual tree depth Fittable_fn07.....	95
Table 77 – Mean number of repulsors found Fittable_fn08.....	95
Table 78 – Individual size Fittable_fn08.....	95
Table 79 – Individual tree depth Fittable_fn08.....	96
Table 80 – Mean number of repulsors found CCPP folds.	96
Table 81 – Individual size CCPP folds.	96
Table 82 – Individual tree depth CCPP folds.	96
Table 83 – Mean number of repulsors found Concrete slump.....	96
Table 84 – Individual size Concrete slump.	96
Table 85 – Individual tree depth Concrete slump.....	97
Table 86 – Mean number of repulsors found Concrete compressive strength.....	97
Table 87 – Individual size Concrete compressive strength.	97
Table 88 – Individual tree depth Concrete compressive strength.....	97
Table 89 – Mean number of repulsors found Wine quality red.	97
Table 90 – Individual size Wine quality red.....	97
Table 91 – Individual tree depth Wine quality red.	98
Table 92 – Mean number of repulsors found LD50 Toxicity.....	98
Table 93 – Individual size LD50 Toxicity.	98
Table 94 – Individual tree depth LD50 Toxicity.....	98

LIST OF ABBREVIATIONS AND ACRONYMS

GPU	Central Processing Unit
GA	Genetic Algorithm
GP	Genetic Programming
GPU	Graphical Processing Unit
ML	Machine Learning
OFT	Overfit Threshold
ODS	Overfit Detection System
RAM	Random Access Memory
RPN	Reverse Polish Notation
RMSE	Root Mean Squared Error

This page was intentionally left blank.

1. INTRODUCTION

Machine learning gives "computers the ability to learn without being explicitly programmed"(Munoz, 2014). Genetic algorithms are a heuristic ML approach based on evolution and natural selection.

Genetic programming (GP) is the use of genetic algorithms (GA) to develop computer programs. The programs are developed by training the genetic program on data with known results (called the training set), the program then produces a solution that is tested and compared against the known result, the difference between the achieved result (prediction) and the known target value is measured using a fitness function (like root mean squared error). In the case presented in this dissertation, a good score is to approximate the target function, so, the closer the results the, the closer the prediction is to the actual data and consequently the better the score. The training process is an iterative process that continually tweaks and adjusts the solution until the program reaches a stop condition, this condition is usually defined either by the total number of runs, solution quality or other termination criteria, the result obtained by the resulting algorithm against the training set is learning ability of the GP. Next, the algorithm that was evolved using the before mentioned process is tested against another dataset with known results, but that the genetic program has not used for training, this is the test set, the result obtained from the test set tells us the ability that the program has to produce a general solution to a problem (generalization ability), and will give us an insight on how well the program is expected to behave with untested data.

In GP, as in other machine learning techniques the solution found to a problem can appear to be good in the training set (a data set that is used to evolve the algorithm) but perform poorly in other datasets where it was supposed to work just as well, this mediocre performance can be and usually is caused by overfitting, an over adjustment of the algorithm to the training set, that makes it perform poorly on other datasets. Overfitting greatly affects the ability for a solution to generalize for data where it has not performed the training on(O'Neill, Vanneschi, Gustafson, & Banzhaf, 2010). Since overfitting normally arises from a solution being overly matched to the training set used to derive it, one can and should look at ways to minimize or completely avoid this(Grus, 2015). GP uses nature-inspired genetic algorithms - these mimic the evolution of species in nature and uses a fitness function to simulate natural selection – to evolve a program that will perform some pre-determined task, like, in this case solving regression problems. There are several ways to avoid overfitting, like getting and preparing more data, splitting the dataset into several parts. In Standard GP, each generation is isolated from the others on everything but the individuals, so it passes no performance information from one generation to another. In nature, however, that is not the case; there are plenty of species that via socialization or other methods, share information that affects the overall performance of a generation, like knowledge, but also rewards or penalizes individuals that present some traits or behaviors.

Passing performance information is an interesting concept to implement and test in GP, so, a simple form of socialization will be applied, by passing performance information with the use of a list of poorly performing individuals.

1.1. OBJECTIVE.

The aim of this dissertation is to study and document the behavior of genetic programming using an approach (repulsor based method) that utilizes prior knowledge of the result obtained by other individuals in a validation set in the selection phase of GP and compare with the Classic GP method. In the repulsor based method, the prior knowledge will be used on an additional selection phase that will penalize individuals that performed well in training and poorly on the validation set. We aim to document and analyze the runs of Standard GP and GP Repulsors algorithms, document the results, and, compare them to access how the introduction of prior knowledge of results by the use of repulsors affects the performance in the learning phase and in the test phase (on a dataset that is not utilized in the learning process), thus checking if and how it impacts the learning and generalization ability of GP.

1.2. METHOD

A program will be developed to implement both Standard GP and Repulsor based GP. The program will be tested and validated against datasets that have known results. After being tested and validated, the genetic program will be executed thirty times on nine different datasets for each algorithm and algorithm setting, a total of seven possible setting combinations. As the program runs, results will be collected and stored in text files and on a database to be analyzed.

The chosen datasets are of two types, mathematical functions, and real-world data; the first category is created using mathematical functions to calculate the dependent variable based on the values provided in the variables, the second consists on real world data datasets with varied sizes and number of dimensions. The presented datasets were chosen based on their characteristics and not on the expected or obtained behavior. (No cherry-picking based on the results).

The settings that will be changed in this thesis are the selected algorithm (GP_CLASSIC, GP_REPULSORS_08 and GP_REPULSORS_09) and the overfit thresholds¹ of 1.01, 1.5 and 2.5 that the repulsor based algorithms will use.

To analyze the GP performance several statistics on performance and behavior are presented first using the global results and comparing them, then, dataset by dataset we compared the results generation by generation, to understand the evolution of the generated solutions.

¹ See section 2.6.2

1.3. DOCUMENT STRUCTURE.

This dissertation is divided into 9 Sections:

Section 1: This introduction.

Section 2: Theoretical framework.

In this section, we will introduce the concepts needed to understand this thesis correctly. We start with the base concepts of ML, then explain GA, from this we will build up to GP.

Section 3: Problem justification and literature review.

Review of current literature on the subject, justify why this thesis is pertinent and can add value to the field.

Section 4: Objectives and Research Questions.

This section includes the objectives of this thesis and the research questions that we will try to answer with this dissertation.

Section 5: Methodology

The way the problem will be approached, explain the software development, the data generation and collection methods and the way the runs will be analyzed.

Section 6: Genetic Programming Software.

Explain the software that was developed for this dissertation, Inputs, outputs, implementation of the algorithms, fitness evaluation methods, parameters and dataset preparation.

Section 7: Results and discussion.

This section includes an overview of the general results of the runs and a detailed analysis of each dataset results.

Section 8: Conclusions.

Conclusions obtained from the analysis.

Section 9: Limitations and recommendations for future works.

Limitations found during the development of the thesis and recommendations of future work to be done in this area.

2. THEORETICAL FRAMEWORK

2.1. BASE CONCEPTS

2.1.1. Machine Learning.

Arthur Samuel in 1959 defined machine learning as the “field of study that gives computers the ability to learn without being explicitly programmed” (Munoz, 2014). Machine learning can be broadly divided in supervised learning – where there are inputs and known outputs-, reinforcement learning and unsupervised learning – where the algorithm discovers any structure on its own. - This field can also be divided into categories based on the output, such as (but not limited to) decision trees, support vector machines, and genetic algorithms.

2.1.2. Optimization Problems.

An optimization problem consists of finding the best solution among a set of solutions; this set is called search space. Formally: An optimization problem is a pair (S,f) where S is the set of all possible solutions (search space), and f is the function (fitness function) that gives a value to all solutions, this number quantifies the quality of the solutions.

$f: S \rightarrow \mathbb{R}$, where S is the search space, and f is the fitness function that returns a quantity of how good a solution is.

The best solution is dependent on the problem being addressed, in minimization problems, the lower the fitness, the better the solution, on maximization problems the higher the fitness, the better.

2.1.3. Symbolic Regression.

Symbolic regression is a type of analysis that searches for a solution to fit a determinate data set, this solution is presented in the form of a mathematical expression (Hypothesis) with one or several variables that will try to match the values of the independent variables. This kind of problem is well suited for Genetic Programming (Vanneschi, 2014). We can represent the training set in the form of a matrix with the last column being the results we strive to achieve. In regression, we try to infer the values of the independent variables using a mathematical expression (the hypothesis) to one or several dependent variables.

$$x_4 + \frac{\left(\frac{x_2 + x_3}{x_4 x_4} + \frac{\frac{x_4 - x_2 + \frac{x_0}{x_3}}{x_4 \times \frac{x_0}{x_2}}}{\frac{x_0}{x_4 x_4}} \right) \times \frac{\frac{x_2 + x_5 + x_6 - x_0 + x_4 x_6 \times \frac{x_4}{x_3}}{x_4 x_4}}{x_2}}{x_2} + x_4$$

Figure 1: Equation found by symbolic regression².

Figure 1 shows a solution obtained for a dataset produced by Genetic Programming to solve a symbolic regression each x_n represents a different variable from the dataset.

² Formula image generated by WolframAlpha using a regression calculated by the GP program.

	No	Cement	Slag	Fly ash	Water	SP	Coarse Aggr.	Fine Aggr.	SLUMP(cm)	FLOW(cm)	Compressive Strength (28-day) (Mpa)
1	1	273.0	82.0	105.0	210.0	9.0	904.0	680.0	23.00	62.0	34.99
2	2	163.0	149.0	191.0	180.0	12.0	843.0	746.0	0.00	20.0	41.14
3	3	162.0	148.0	191.0	179.0	16.0	840.0	743.0	1.00	20.0	41.81
4	4	162.0	148.0	190.0	179.0	19.0	838.0	741.0	3.00	21.5	42.08
5	5	154.0	112.0	144.0	220.0	10.0	923.0	658.0	20.00	64.0	26.82
6	6	147.0	89.0	115.0	202.0	9.0	860.0	829.0	23.00	55.0	25.21

Figure 2: Concrete slump dataset to be modeled by a symbolic regression produced by GP

2.1.4. Learning Process and datasets.

In GA and as in other supervised learning methods, we use a dataset with known results to evolve the program; this data set is called training set and usually contains several variables and one or more results (for this dissertation we will use several variables and one result). The algorithm uses the solutions presented in this set to evaluate how well suited the solution is (fitness evaluation). Depending on the learning approach the original dataset is usually divided into Training, Test and Validation sets, the training set is used for learning, the test set is used to verify if the solution generalizes well and the validation data set is used to control the training process. It must be emphasized that the test set should not be utilized in the learning process, doing so will induce bias in the results, and similarly, neither should the training data be used to test the results as this would make the results appear better than what they are. Ideally, all sets should originate from the same distribution.

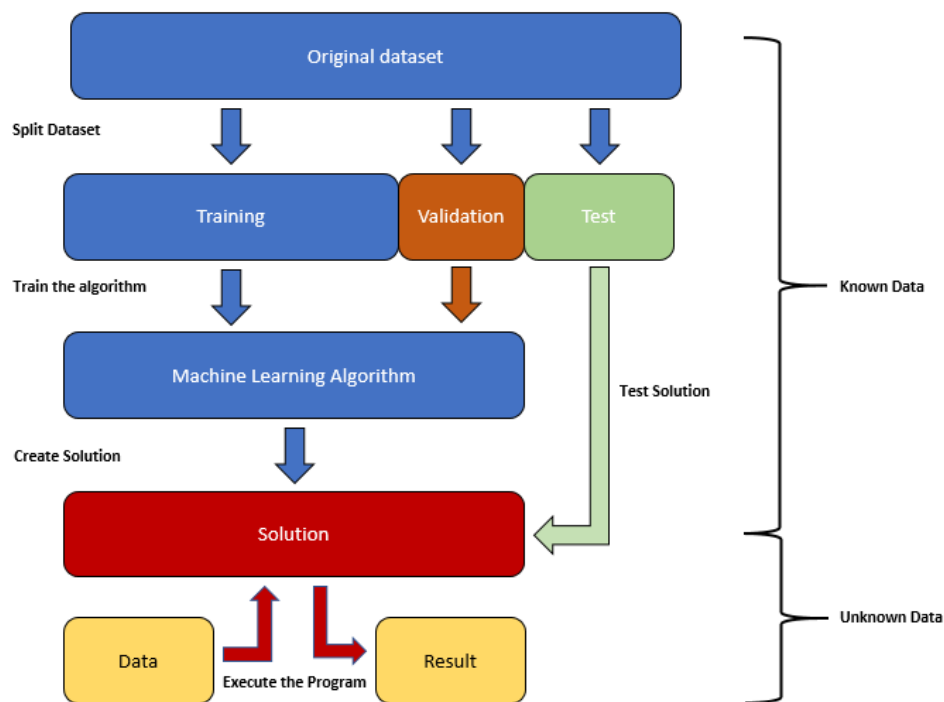


Figure 3: Learning Process

2.1.5. Overfitting.

Overfitting occurs when an algorithm or statistical model performs well with the training set, and, when used in other data sets, such as the test set or real world data, performs poorly, this usually results from an over-adjustment to the training data. Overfitting is a common problem in machine learning (Subramanian & Simon, 2013), and also presents itself in Genetic programming (Koza, 1992). Overfitting usually leads to poor generalization capabilities (Grus, 2015), as it is the result of a significant adaptation of the predictor to the training set. Common causes for overfitting can be having small training set, training set/variable ratio or overtraining on a dataset.

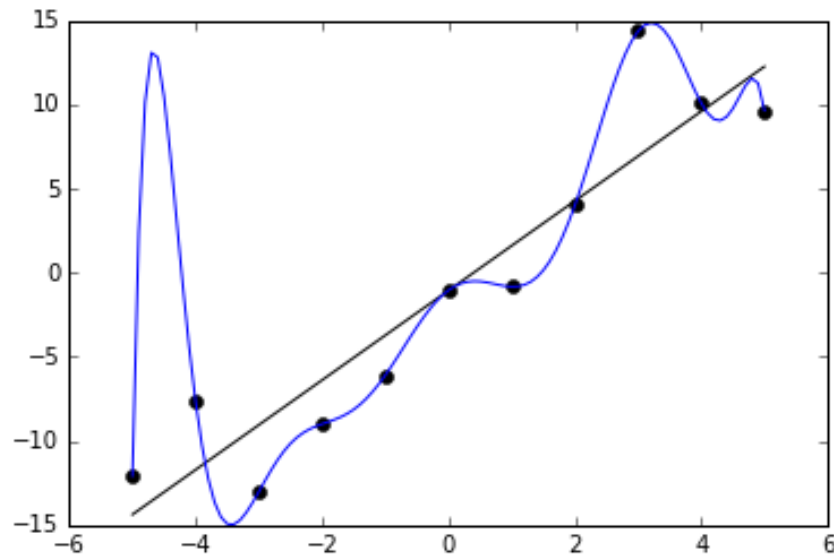


Figure 4: Dots represent real data, Blue Line overfitting as an over adjustment to the learning data³.

In ML the variance vs. bias trade-off can be thought as a choice between simple and inflexible algorithms (highly biased) and complex ones that can fit the training data very well if not perfectly. The first will usually ignore the noise, but may also miss trends and characteristics of the data; the more complex ones are more susceptible to be overfitted to the training set. Overfitting leads to high variance in the solution due to the over adaptation to not only the training data characteristics and trends but also to any noise or non-general detail present in the training data (Briscoe & Feldman, 2011).

Becket states that overfitting can be avoided by three main strategies (Becker & Seshadri, n.d.), penalizing complexity or simplifying, limiting the number of models considered, using a validation data set to control learning.

Several more refined techniques can be used to avoid overfitting such as regularization, increasing the size of the training set, change the fitness function (Koza, 1992; Vanneschi, 2014), manipulating the way the training set is presented to the algorithm (Goncalves & Carreiras, 2012). In this dissertation, we will access if the use of repulsors has any effect in reducing overfitting.

³ Image from Wikimedia Commons: https://commons.wikimedia.org/wiki/File:Overfitted_Data.png

2.2. EVOLUTION AND NATURAL SELECTION.

“One general law, leading to the advancement of all organic beings, namely, multiply, vary, let the strongest live and the weakest die.”(Darwin, 1859)

On his groundbreaking book ‘On the Origin of Species by means of natural selection’(Darwin, 1859) Charles Darwin argued that the better some individual deals with environmental pressure, the higher his chance of survival and, consequently, the higher the probability of reproducing and passing its traits to the next generation. Since the individuals of the new generation inherit the traits that made their progenitors better fitted to deal with the environment, these individuals will have themselves a better chance of dealing with environmental pressure and will survive long enough to have the opportunity of passing their genes to the next generation. It is important to note that in a Darwinian approach Individuals do not evolve, populations do.

2.3. GENOTYPE AND PHENOTYPE (ENCODING).

An individual’s phenotype is the collection of its observable traits and characteristics; this includes physical and behavioral features. The genotype is where these characteristics are encoded in the life sciences case, in the individual’s DNA. In life as we know it, everything is coded using four bases Adenine, Cytosine, Guanine, and Thymine (Uracil in RNA). In nature, genes encode a trait, like for example the eye color, in the computer, the genes encode the solutions to a problem, this encoding, can be done in a variety of ways, for example, real encoding and binary coded

In figure 5 a sample codification for a possible Traveling salesman problem solution, the encoding merely represents the cities, by their codes, in the order that they will be visited.

[1,5,8,3,2,11,7,0,9]

Figure 5: Text encoding of a traveling salesman problem solution.

Figure 6 presents a sample of binary coded encoding, with a possible solution to a knapsack problem, where each position represents an object is or is not in the sack.

[1,0,0,1,1,0,1,0,0,0,0,0]

Figure 6: Binary representation of a solution.

Figure 7 presents the encoding of an equation in RPN notation of programs generated to solve a symbolic regression problem.

```
['x9']  
['x4', 'x0', '-', 'x9', 'x2', '-', '+']  
['x9', 'x6', '*', 'x5', 'x4', '-', '/']  
['x3', 'x8', '+', 'x8', 'x1', '-', '*']  
['x2']  
['x0', 'x6', '*', 'x7', 'x9', '+', '*']  
['x2']  
['x10', 'x7', '/', 'x7', 'x8', '/', '+']
```

Figure 7: Text representation of RPN individuals

2.4. GENETIC ALGORITHMS.

Genetic algorithms are a heuristic search and optimization technique that uses a mechanism inspired in the way living organisms pass their traits from one generation to the next. The genetic algorithm as we know it was first presented by John Holland in the book *Adaptation in Natural and Artificial Systems*.

There is an analogy with nature, so solutions are individuals and a group of solutions is called a population.

Table 1 presents a mapping between nature and Computer program (Sivanandam & Deepa, 2008):

Table 1 – Nature/computer terminology mapping.

Nature	Computer
Population	Set of Solutions
Individual	Solution to a problem
Fitness	Quality of a solution
Chromosome	Encoding for a solution
Gene	Part of the encoded solution

With a Darwinian approach, GAs try to mimic evolution and natural selection. Evolution is achieved by using operations that create change into the individuals and consequently to the population, the main mechanisms of evolution are: crossover and mutation. Natural selection is attained by classifying individuals using a fitness function; this function will evaluate how good the individual is and will affect the chance it has to pass traits to the next generation. The fitness function is highly dependent on the problem being solved.

Genetic algorithms are appropriate for problems where an accurate solution (that may not be optimal) is needed and when there is no need for a real-time response.

Advantages of GA.

Some benefits of Genetic algorithms are the ability to provide a very accurate result and usually provide an easy to understand solution to a problem, contrary to neural networks, where the inner workings are very hard to interpret.

Limitations of GA.

Some of the limitations of GA are: the cost of computing the fitness function, they are highly stochastic method, each run usually provides a different solution, so it is better to run several runs with the same parameters to obtain a solution, high dependence on the initial population that is randomly generated, high dependence on parameter settings, that is very hard to tweak, they do not guarantee to find the optimal solution.

2.4.1. Standard Genetic Algorithm.

The Standard Genetic Algorithm can be represented as follows:

- 1 Create a population P of n random individuals.
2. Repeat until the termination criteria are satisfied.
 - 2.1. Calculate fitness of all individuals⁴.
 - 2.2. Create an empty population P'.
 - 2.3. Repeat until P' is filled.
 - 2.3.1. Choose a genetic operator, (crossover with probability P_c , otherwise, reproduction with probability $1-P_c$
 - 2.3.2. Select two individuals with a selection algorithm
 - 2.3.3. Apply operator selected in 2.3.1 to the individuals selected in 2.3.2.
 - 2.3.4. Apply mutation to individuals obtained in 2.3.3 with a probability of P_m .
 - 2.3.5. Insert resulting individuals into P'
 - 2.3.6. Iterate until P' contains n individuals.
 - 2.4. Replace P with P'.
3. Return the best individual.

2.4.2. Common Genetic Algorithm Parameters.

Parameters are dependent on the implementation; however, there are common parameters, some of them are presented below.

1. Number of characters of an individual - represents the size of the individual.
2. Population Size – Number of individuals that make the population.
3. Maximum number of generations – The number of iterations of the GA.
4. Selection Algorithm – The selection method to be used.
5. Crossover Probability – The probability of occurring crossover.
6. Mutation Probability – The probability of occurring mutations.
7. Elitism – If the algorithm uses elitism⁵.

2.4.1. Common Genetic Algorithm Termination Criteria.

The algorithm terminates if one or a combination of termination criteria is reached.

- 1- Reaching the defined number of maximum generations.
- 2- Achieving a determined fitness.
- 3- Reaching a set time limit running.
- 4- User action.

⁴ Except for tournament selection.

⁵ When using elitism, the best individual(s) found is/are kept and passed unaltered to the next generation.

2.4.2. Fitness Evaluation.

Fitness evaluation is the process that calculates how good solution an individual is a for the proposed problem, so, the fitness function can vary a lot as it's highly dependent on the problem being approached, equation 1 represents a possible fitness function for a traveling salesman problem, it's simply the sum of the Euclidean distances between the cities.

Equation 1

$$fitness = \sum_{i=0}^{n-1} \sqrt{((x_n - x_{n+1})^2 + (y_n - y_{n+1})^2)}$$

The interpretation of the fitness function also varies depending on the problem being solved, typical examples are maximization and minimization problems, where the goal is to get the maximum and minimum values possible respectively. The fitness function is used by the selection phase of the Genetic Algorithm. This evaluation is the mechanism that classifies the individuals by how good they are, and will provide the probability of the individual being selected to contribute to the next generation.

2.4.3. Phases of Genetic Algorithms.

Genetic algorithms start with an initialization phase, where the parameters are set, data prepared and initial population created, they then enter the iterative phase where generations evolve, this is called the main loop. In the main loop, the two most important phases of genetic algorithms are the Selection phase and the Variation phase, in the first, the algorithm selects the elements that will undergo changes on the second phase.

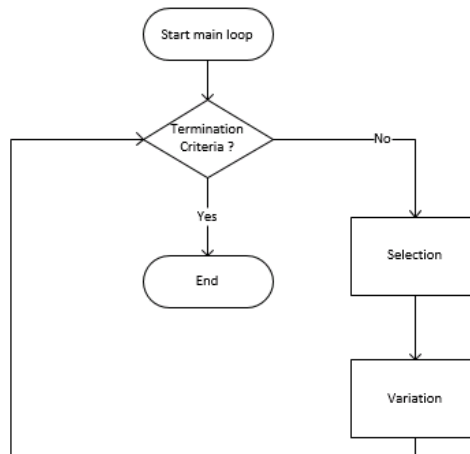


Figure 8: Main phases of Genetic Algorithms.

Figure 8 presents a simplified flowchart of the main phases of GA, the after initialized, the algorithm iterates through several generations until a termination or stop condition is reached.

2.4.4. Selection phase and selection methods.

This phase will determine which individuals will contribute to the genetic pool of the new generation. The major rule is: no matter which selection method is used, there shall always be a chance, no matter how small, for the most unfit individual of the population to be selected and have its traits passed to the next generation⁶.

The three selection operators that are more frequently used are fitness proportional selection, ranking selection and tournament selection.

2.4.4.1. Fitness Proportional Selection

In Fitness Proportional Selection, or roulette wheel selection, the probability of an individual being selected is proportional to its fitness. The probability of selection on a maximization problem using fitness proportional is presented in Equation 3.

Equation 2

$$P_i = \frac{f_i}{\sum_{j=0}^{n-1} f_j}$$

Table 2 and figure 9 show an example of the probability of selection by fitness proportional selection.

Table 2 – Fitness proportional selection probabilities.

Individual	Fitness	Probability
A	2	0,2
B	5	0,5
C	2	0,2
D	1	0,1

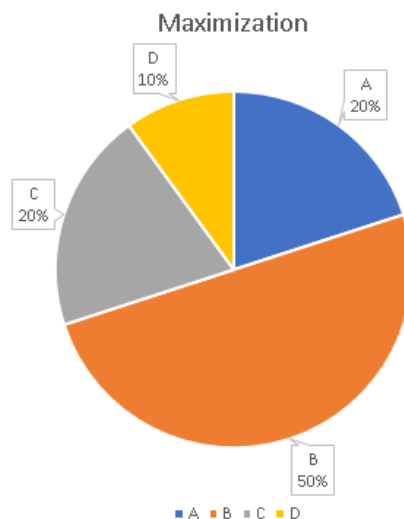


Figure 9: Fitness proportional selection probabilities.

⁶ This phase is the equivalent of natural selection in nature.

2.4.4.2. Ranking Selection

In Ranking Selection, the probability of an individual being selected is the result of a function (example: a linear one) that determines the individual's position in the selection queue. Equation 3 shows the formal probability of an individual being selected.

Equation 3

$$P_i = \frac{n}{\sum_{j=0}^{n-1} i_j}$$

Depending on the individual's position in the queue, a probability is assigned to them. It is based on this that an individual is selected or not. This algorithm does not take into account how much better than the other an individual is; it only uses its position in the rank.

Example: $n=3$ so $\{i_0, i_1, i_2\}$ and $f_0 > f_1 > f_2$, the ranking selection procedure is as follows :

1. Order the individuals in the population (sort them) from the worst to the best according to their fitness.
2. The probability of selecting an individual is a function of the position it occupies on the ranking. (it can be a linear function, exponential and others).
3. Perform a roulette wheel selection using these probabilities to select the individual.

Table 3 – Rank Selection order example

Rank	Individual
1	$i_{(2)}$
2	$i_{(1)}$
3	$i_{(0)}$

Using:

Equation 4

$$P_i = \frac{n}{\sum_{j=0}^{n-1} i_j}$$

We have the probabilities presented in equation 5 and figure 10.

Equation 5

$$P_{i_{(0)}} = \frac{3}{3 + 1 + 2} = \frac{1}{2} \quad P_{i_{(1)}} = \frac{2}{3 + 1 + 2} = \frac{1}{3} \quad P_{i_{(2)}} = \frac{1}{3 + 1 + 2} = \frac{1}{6}$$

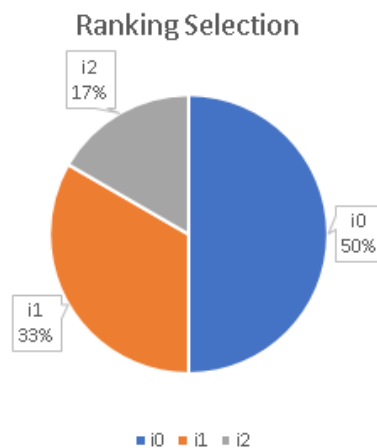


Figure 10: Example of ranking selection probabilities.

2.4.4.3. Tournament Selection

Tournament selection consists of random selection with repetition, samples of the population are selected, and the n best individuals of them are select to the variation phase. The steps involved are:

1. Randomly select a small subset of elements, with repetition (an element can appear several times in the subset).
2. Select the best in the subset
3. Repeat n times where n is the size of the set.

The size of the tournament is a parameter, the bigger the subset, the higher the probability of choosing the best in the population.

- Big Tournament size implies big selection pressure.
- Small Tournament size implies low selection pressure.

With this selection method, there is no need to calculate the fitness of all the individuals in the population, so it is less computationally intensive, only the individuals that are included in a tournament will have their fitness computed, this characteristic makes tournament selection a widely used selection method.

2.4.4.4. Steady State Selection

Steady state selection, splits the population into blocks, grouping the individuals by their fitness, then, performs the variations on a subset of the best individuals, keeping the ones with the best fitness(Sivanandam & Deepa, 2008) and using the new offspring to replace the individuals with bad fitness, so that the new generation is composed of the best individuals and the new individuals evolved from the best.

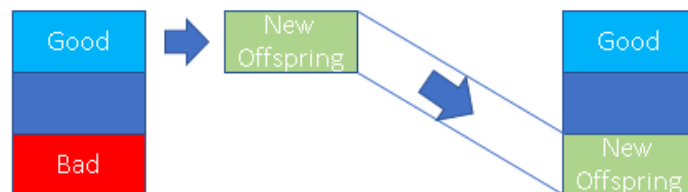


Figure 11: Steady state selection(Sivanandam & Deepa, 2008)

This selection method is similar to many organisms in nature where in practice parents compete with their offspring, and strong individuals can outlive one or more generations.

2.4.4.5. Reward Based Selection

This selection method chooses the individual not only by its fitness but also by combining with a cumulative reward given by all its parents.(Deb, Pratap, Agarwal, & Meyarivan, 2002)

2.4.5. Variation phase and variation methods.

This phase will induce variations on individuals (simulating natural evolution), it is composed of crossover (cx), mutation (mx) and reproduction (the last does not change anything on an individual), any of these phases happen with a given probability defined on the algorithm parameters.

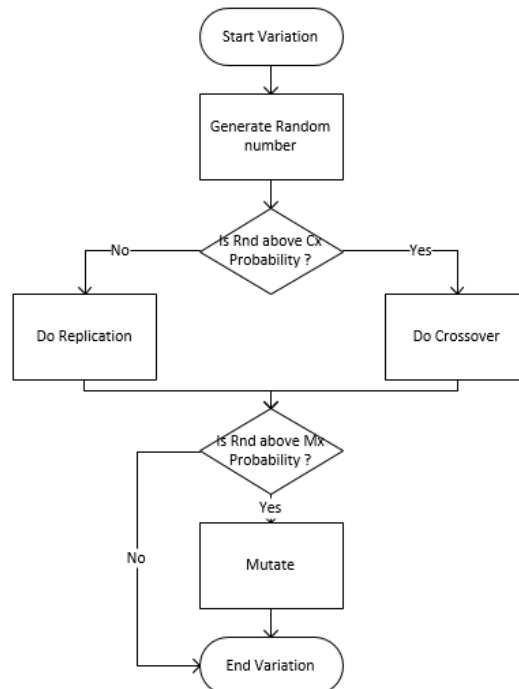


Figure 12: Variation phase.

2.4.5.1. Crossover

Crossover is the combination of individuals, where two individuals exchange their genetic material with the intent of producing offspring that is better than any of them. Depending on the encoding being used there are different crossover methods, figure 12 shows binary crossover between two individuals.

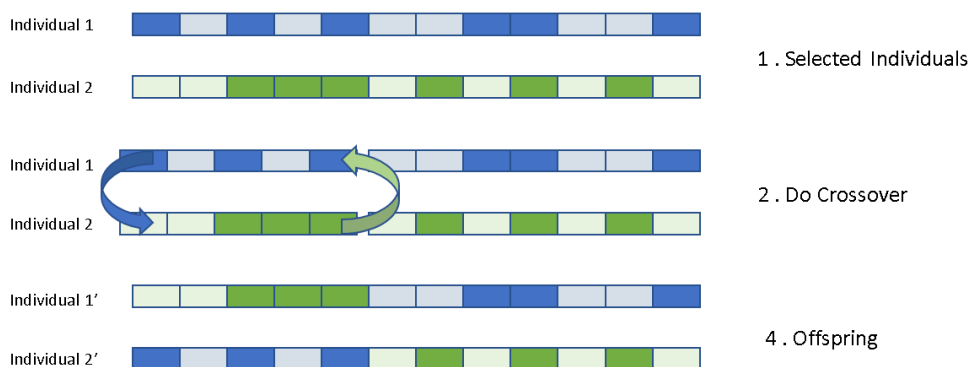


Figure 13 : Binary Crossover example

In binary crossover, random positions on the genome are 'marked' (at least one position), then two new individuals are created by exchanging the parts delimited by these markers, between the two parents. This type of crossover should only be used with individuals where there is no need to keep any syntactical consistency.

There are more complex variations of crossover that are used when syntactical coherence is necessary or when it is not acceptable to lose any individual's component into another individual, causing repetitions on one and omissions in the genome of the other. (example traveling salesman problem), one of these techniques, cycle crossover, avoids disrupting the individuals by using a cycle that only ends when it reaches a full loop through the individuals to be replaced.

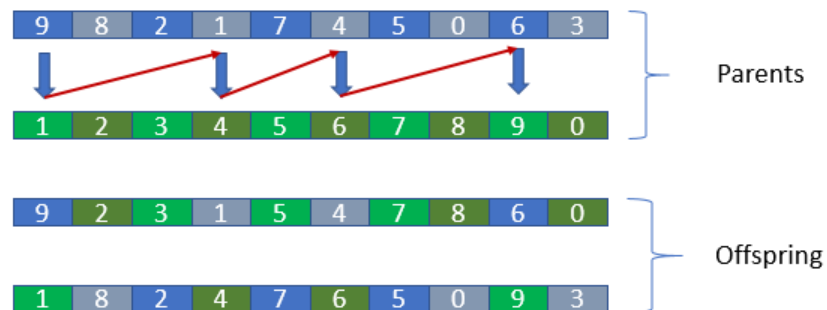


Figure 14: Cycle crossover

Figure 14 shows an example of cycle-crossover on a representation of a traveling salesman problem⁷, note that in the offspring there is no omission or repetition of the cities represented by numbers.

2.4.5.2. Mutation

Mutation is a change in the individual; this can be a simple binary mutation where one or more bits are swapped or a more complex mutation like Partial Inversion mutation



Figure 15: Binary mutation

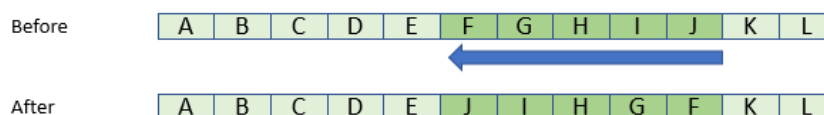


Figure 16: Inversion mutation

More complex variations of mutations, like inversion mutation, are used when it is not acceptable to lose an individual's component (example traveling salesman problem)

2.4.5.3. Reproduction

In reproduction, the offspring is an exact copy of the parents, i.e. no genetic operators are applied, and they are copied exactly as they are.

⁷ Common Computer Science scenario where we must choose the shortest path for a salesman to travel between several cities minimizing the distance travelled and avoiding visiting a city more than once.

2.5. CLASSIC GENETIC PROGRAMMING.

GP consists of the use of genetic algorithms to evolve computer programs. On a more formal way, Poli writes in his book 'A Field guide to Genetic Programming' (Poli, Langdon, & McPhee, 2008):

'Genetic programming (GP) is an evolutionary computation (EC) technique that automatically solves problems without requiring the user to know or specify the form or structure of the solution in advance. At the most abstract level GP is a systematic, domain-independent method for getting computers to solve problems automatically starting from a high-level statement of what needs to be done.'

A Common way to build the programs is by using a set of functions and terminal sets. Functions and terminal sets are dependent on the problem being solved. In regression problems, where mathematical expressions are created, the function set consists on mathematical functions like addition '+', subtraction '-', multiplication '*' and division '/',⁸ the terminal set is composed of the input variables and constants.

Figure 17 presents the RPN representation of a genetic program to approximate the sin function, generated from a dataset containing the values of the sin from -5 to 5 in 0.1 increments, the solution has not been simplified and presents signs of bloat (see section 2.7.5.4). Figure 18 presents the plot of the function (result) and the data where the algorithm trained (target).

```
1  x0 x0 + x0 x0 * x0 x0 * / * x0 x0 * x0 x0 x0 + * / x0 x0 - x0 / x0 x0 + - + x0 x0 * x0 + x0 x0 / x0 x0 + +  
+ - x0 x0 x0 / x0 x0 - - x0 x0 - x0 x0 - / / * - x0 x0 * x0 x0 x0 + * / x0 x0 * x0 x0 / + x0 x0 / x0 x0 -  
- / * x0 x0 * x0 x0 * + x0 x0 / x0 x0 + - x0 x0 - - - x0 x0 * x0 x0 x0 / / - x0 x0 x0 / x0 - - / * + + -  
x0 x0 + x0 x0 + x0 x0 * * x0 + + x0 + x0 x0 - x0 + / x0 x0 + x0 x0 - - x0 x0 - x0 x0 / * * x0 x0 / x0 x0 /  
- x0 x0 * x0 x0 - + * * x0 x0 - x0 x0 + / x0 x0 * x0 x0 / / - x0 x0 + x0 x0 / + x0 x0 * x0 x0 - / - / x0  
x0 * + / x0 x0 * x0 x0 * + x0 x0 x0 * x0 x0 x0 + * / x0 x0 * x0 x0 / + x0 x0 / x0 x0 - - / * x0 + - - + /
```

Figure 17: RPN representation of a solution.

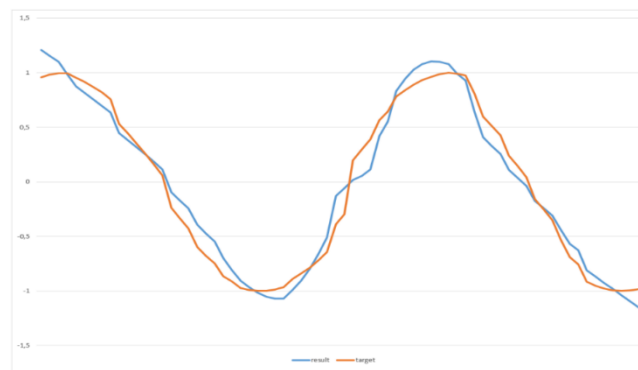


Figure 18: Plot of a solution found to approximate the sin function.

In other problems like the Santa Fe Trail Ant problem⁹(Koza, 1992) the terminal set is a set of actions {Left, Right, Move}, and the function set is composed of a single function IfFoodAhead, these are arranged to control the movement of an artificial ant on a maze.

⁸ In genetic programming, there is a widespread practice to use safe division. Safe division protect the program from errors that would be generated by divisions by 0. Safe division can return different values depending on the objective, koza proposed(Koza, 1994) to return a 1, but other authors(Keijzer, 2003) use different approaches to avoid function approximation by use of asymptotes.

⁹ The santa fe trail problem(Koza, 1992), simulates an ant searching for food pallets laid on a non-continuous manner on a thyroidal grid. The objective is to get the maximum number of pallets in the smallest number of steps and in the most compact program possible.

2.5.1. Classic Genetic Programming Algorithm.

GP employs genetic algorithms to produce individuals that are computer programs, so, like in genetic algorithms, in GP there are also the initialization, variation, and evaluation phases.

Classic GP Algorithm:

- 1 Create an initial population P of n random programs.
(Initialization)
2. Repeat until the termination criteria are satisfied.
 - 2.1. Calculate fitness of all individuals. (Except if using tournament selection)
 - 2.2. Create an empty population P'.
 - 2.3. Repeat until P' is filled.
 - 2.3.1. Choose a genetic operator, (crossover with probability P_c , otherwise, reproduction with probability $1-P_c$)
 - 2.3.2. Select two programs with a selection algorithm
 - 2.3.3. Apply operator selected in 2.3.1 to the individuals selected in 2.3.2.
 - 2.3.4. Apply mutation to programs obtained in 2.3.3 with a probability of P_m .
 - 2.3.5. Insert resulting individuals into P'
 - 2.3.6. Iterate until P' contains n programs.
 - 2.4. Replace P with P'
3. Return best program evolved.

Before starting the algorithm, we must:

- 1) Define a language to express the algorithm, example: RPN.
- 2) Select function set and terminal set if applicable.
- 3) Set the values of some parameters.
- 4) Define a fitness function.

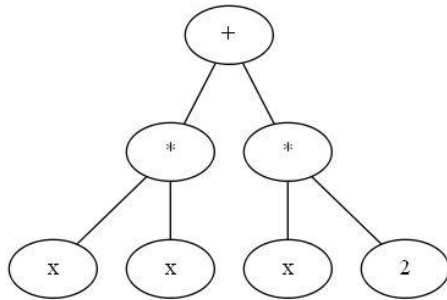
2.5.2. Program Representation

Programs can be represented in several forms, Koza (John R. Koza, n.d.; Koza, 1992) suggested the representation as trees due to the simplicity of implementation in computers using LISP¹⁰. Trees can be represented in a linear fashion too. Since in this case, I will be using polynomial expressions, the Reverse Polish Notation (RPN) also known as Postfix notation (these terms will be utilized interchangeably) will be employed and represented linearly. The infix notation (or algebraic notation) despite being the most common way to present mathematical expressions has the disadvantage of being more complex in computational terms, as it must be completely parsed before any evaluation can be performed, this arises from the different precedence that different operators have. RPN or Postfix eliminates the need for the interpreter to know the precedence of operators. RPN uses a stack based structure where operands are pushed first into the stack, and, an operator is applied to them in order. (Figure 19)

¹⁰ A computer language distinguished by its heavy use of parenthesis and prefixed notation.

The three examples presented in Figure 19 show the same program or expression, in this case, a simple quadratic equation, represented in these three forms: tree graph, infix or algebraic and postfix or RPN notation.

The Graphic representation of the tree:



Algebraic or infix notation:

$$x^2 + 2x$$

RPN or postfix notation:

$$x x * 2 x * +$$

Figure 19: Representation of an expression

How Postfix or RPN (Reverse Polish Notation) works:

RPN uses a list of elements to be evaluated; these elements are processed in order. When parsed, if the element is a number it is pushed to the stack, if it is an operand it pops the previously stacked element or elements and applies the operand to them, then, pushes the result onto the stack again, this process is repeated until there is nothing more to input, and only one value remains on the stack, this last value is the result.

Example:

Take the algebraic expression $((6+4)*5)/2$, its RPN representation is $5\ 4\ 6\ +\ *\ 2\ /$, and is evaluated as presented in table 4.

Table 4 – RPN processing example.

Input	Type	Stack	Notes
5	Operand	5	Push 5 into the stack
4	Operand	5 4	Push 4 into the stack
6	Operand	5 4 6	Push 6 into the stack
+	Operator	5 10	Pop 6 and 4 from the stack, add, push the result into the stack
*	Operator	50	Pop 5 and 10, multiply and push the result into the stack.
2	Operand	50 2	Push 2 into the stack
/	Operator	25	Pop 50 and 2, divide and push the result into the stack
	Result	25	

2.5.3. Common GP Parameters

To initialize the algorithm a set of parameters must be defined ‘a priori’, these are highly dependent on the problem being addressed, some of these parameters are:

- Population Size – Number of individuals that make up the population.
- Maximum number of generations – the number of generations to produce before terminating the program run.
- Selection Method – chooses the kind of selection that will be used, examples, fitness proportional, tournament or others.
- Tournament size – This parameter sets the number of individuals that participate in the tournament.
- Crossover Probability (Px) – Probability for the crossover to occur, this value also implies the probability of reproduction to happen, as it is 1-Px.
- Mutation Probability (Pm) – Probability of mutation.
- Mutation mode – Type of mutation to perform, point, subtree, etc.
- Elitism – If the best individual/Individuals of a previous generation are copied to the new generation.
- Initial Population generation mode – How the initial generation is created, full, grow or RHaH.
- Initial population depth – The maximum ‘size’ of the individuals in the initial generation.

2.5.4. GP Fitness evaluation

Intuitively, using figure 20, in a minimization problem, using distance as a fitness function, the smaller the area between the achieved result (blue), and the target result (orange) the fitter and better is the solution.

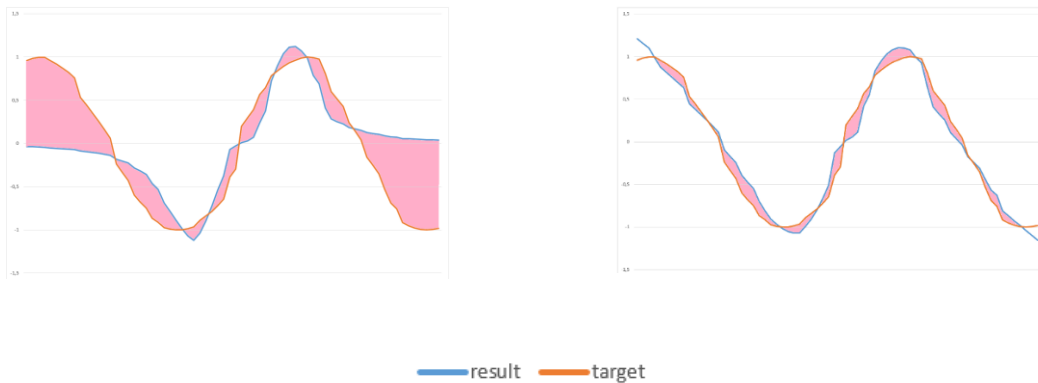


Figure 20: Representation of a solution to the sin function in different generations, left: early generation, right: late generation.

The fitness function used above is the minimization of RMSE(Vanneschi, 2014)(Zaji & Bonakdari, 2015) where the difference between the hypothesis presented by the individual and the data set result is calculated using equation 6.

Equation 6

$$E = \frac{1}{M} \sqrt{\sum_{i=1}^M (f(x_0, x_1, x_2, \dots, x_i) - y_i)^2}$$

In Equation 6 E represents the Error, obtained by averaging the squared errors of the found solution against the real data, result values are defined in x and target values in y. The lower the Error, the better the fitness.

2.5.5. Population Initialization and initialization methods.

The initial population is initialized at random. The initialization phase must produce individuals in the correct form for the problem being addressed, in other words, the individuals should be syntactically correct. If possible, the initial population should be as diverse as possible.

Initialization methods:

There are three main initialization algorithms to create the initial population, growth, full and ramped half and half.

Growth algorithm:

- 1) Create two sets, one with the Functions or operators (A) and one with the terminators or operands (B).
- 2) Define the tree depth D.
- 3) In tree depth 0 (the main node), randomly choose an operator from A.
- 4) Repeat until depth=D-1 or all branches are terminated with operands.
 - a. Select a random element from the set A U B. (a set of both operators and operands).
- 5) Terminate all branches that do not have an operand.

This method creates uneven trees, and the depth of the individuals is unpredictable.

Full algorithm:

- 1) Create two sets, one with the Functions or operators (A) and one with the terminators or operands (B).
- 2) Define the tree depth D.
- 3) In tree depth 0 (the main node), randomly choose an operator from A.
- 4) Repeat until depth=D-1.
 - a. Select a random element from the set A (a set with only operators).
- 5) Terminate all branches with an operand from set B.

This method creates trees that are always of the same size D; there are no *simpler* trees.

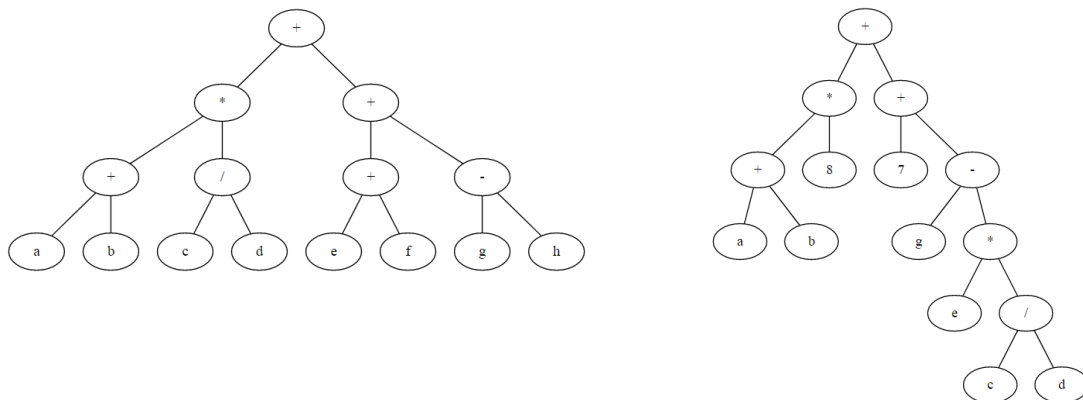


Figure 21: On the Left, a tree created with the full algorithm, and on the right, a tree initialized with the growth algorithm.

Ramped half and half

This algorithm addresses the problems of the previous two by creating an initial population that is generated using both growth and full algorithms and with different sizes, a population generated this way is more diverse in both size and structure.

- 1) Define how many trees we need to create. (population size)
- 2) Divide the population into D groups where D equals the maximum depth
- 3) Decide how many individuals we have in each group (Population size/D)
- 4) assign each group a maximum depth:
 - a. Group 1 - max depth = 1
 - b. Group 2 - max depth = 2
 - c. Group 3 - max depth = 3
 - d. ...
 - e. Group D - max depth = D
- 5) For each group, half of the individuals will be generated with the growth algorithm and the other half with the full algorithm using the corresponding maximum depth.

2.5.6. Selection Phase

In GP the selection methods do not differ much from the ones in Genetic algorithms. The only difference is in the fitness function that is applied. Due to the computational cost of evaluating the fitness of a Genetic Program, tournament selection is widely used in this phase avoiding the task of evaluating every individual in the population.

2.5.7. Variation Phase (Crossover and Mutation)

Variation techniques in GP must preserve the correct syntax of the program being created so that it still makes syntactical sense after the mutation or crossover.

2.5.7.1. Mutation

For mutation, a random node is selected from the tree, then depending on the method, that node is replaced with another from the same set (operators/operands) or a new sub-tree is grown starting on that node. Care must be taken to preserve syntactical coherence.

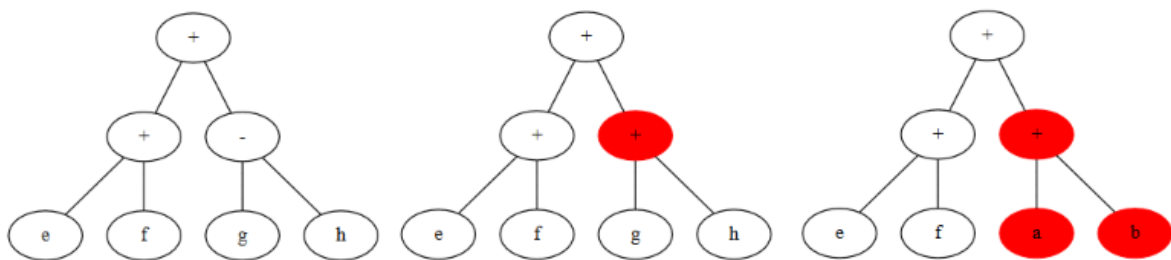


Figure 22: Mutation Example, left: the original tree, middle: point mutation and right: sub-tree mutation.

2.5.7.2. Crossover

Like in GA, crossover takes parts from both parents and combines them into new individuals. When performing a crossover, a node is selected in each of the parents, then, that node and all its siblings are *cut out* and swapped from one parent to the other. This important part is that the whole functional part is exchanged so that the RPN evaluation can be completed successfully. The syntax of the program must be maintained.

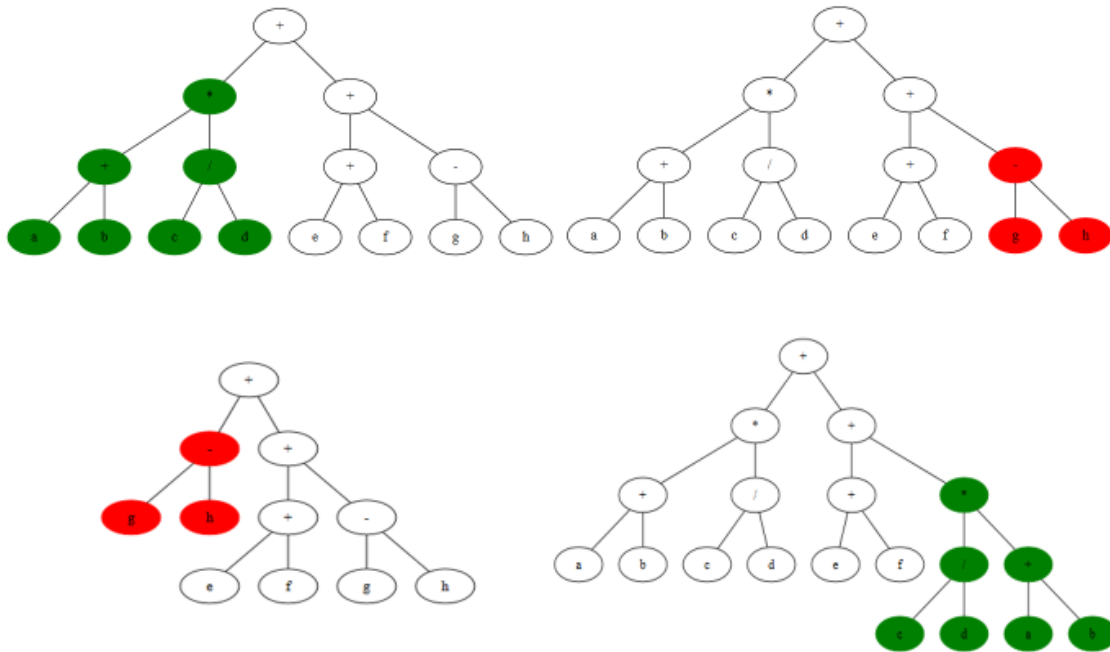


Figure 23: Crossover Example, above two parents, the selected part is in red and green, below, the siblings with the swapped parts still in their parent's color.

2.5.7.3. Reproduction

Like in G.A. in reproduction the offspring is an exact copy of the parents without the application of any genetic operators.

2.5.7.4. Bloat

Bloat is the rapid growth in the size of GP individuals without a corresponding increase in fitness. Bloat appears as useless chunks of code that have little or no effect in the solution result. These chunks of code are called introns in analogy to the block of DNA that do not transcribe to anything in living being's DNA. However, unlike biological DNA, introns in GP are processed and are evaluated. Bloat degrades GP performance, making successive generations take longer to evolve, creates solutions that are larger than needed and may limit the scalability of GP (Hutchison & Mitchell, 2003).

2.6. REPULSOR BASED GENETIC PROGRAMMING.

In this thesis, we extend classic GP with the use of repulsors. This section will set the ground for the repulsor based approach used in this dissertation.

2.6.1. Repulsors.

A repulsor is an individual that has some trait that caused him to overfit in a set that was not directly used in training, in this case, a validation set. The repulsors as they are found (through overfit detection) are placed on a repulsor list.

2.6.2. Overfit detection.

Overfit detection signals individuals as overfitters if they meet some predetermined criteria, the criteria is dependent on the approach. In the case of this dissertation we use the difference in fitness from the previous generation, see section 6.6 for details.

2.6.3. GP Repulsors Algorithm.

The algorithm is an extension of GP Classic with the inclusion of the overfit detection and the additional repulsor based selection phase. Where we select the individuals based on their semantic distance from the repulsor list. The further the solution is from the individuals in the list, the better.

So, the fittest individuals, will perform well in the standard selection with the standard fitness function (usually minimizing the difference), and will also perform well in the repulsor tournament (usually maximizing the difference between themselves and the repulsors).

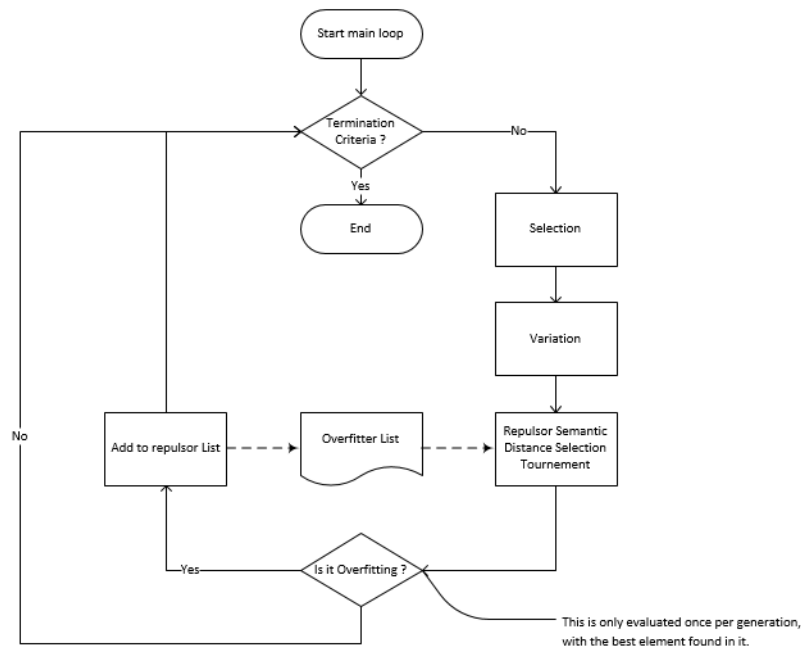


Figure 24: Repulsor based algorithm example.

A detailed implementation of how the algorithm was implemented for this dissertation can be found in section 6.4.2.

3. PROBLEM JUSTIFICATION AND LITERATURE REVIEW.

The relevance of this work arises from to the problems caused in machine learning by overfitting, and by the need of more information on the behavior of repulsor based algorithms. Overfitting is expensive, both in time and in processing power. An overfitted solution will usually be more computer intensive (Grus, 2015), many machine cycles are wasted in solutions that are overfitted and usually worthless. The reduction of overfitted solutions will also bring faster results due to the elimination of failed program generations that are eliminated by repulsors. Overfitting greatly affects generalization ability, M. O'Neill and L. Vanneschi(O'Neill et al., 2010) suggest that generalization ability is less studied in GP when compared with other areas of machine learning.

Generalization ability is of great importance when we want to use GP to support predictions or forecasting(Mouffok, 2014).

Overfitting is a problem that deeply affects not only GP but other machine learning techniques as well (Piotrowski & Napiorkowski, 2013). Searches for genetic programming, overfitting and repulsors on literature in the web of science and Mendeley show the results in table 5:

Table 5 – Search matches on Repulsors.

Search Terms	Web of Science ¹¹	Mendeley
Fitness	~5900	~7400
Fitness genetic programming	~670	~1780
Overfitting	~1070	~3560
Repulsors Genetic Programming ¹²	0	1

These numbers show that overfitting and fitness are important topics. Searches on the web of science show nothing in the subject of repulsors, but Mendeley shows 1 article by Leonardo Vanneschi and Steven Gustafson ' (Vanneschi & Gustafson, 2009), so this is certainly a topic worth looking more into.

A performance analysis is also useful, since the introduction of repulsors will increase the complexity of fitness evaluation, one must compare the computational cost of having few, but good quality solutions, against having a lot of bad solutions with some good ones in the middle, it may be the case that the cost of processing many solutions with a cheap fitness function becomes lower than obtaining few high-quality solutions. This will, of course, be highly dependent on the problem, but a case worth considering.

Jyothi Subramanain, argues that overfitting is a problem that occurs not only in high dimensional data but also with a low number of dimensions(Subramanian & Simon, 2013).

We found several articles describing techniques to improve generalization ability. Becker(Becker & Seshadri, n.d.) and Identified three main approaches to overfit avoidance and comprehensibility, Penalizing complexity or biasing towards simplicity , limiting the number of models and using a

¹¹ Results are filtered on the topic 'COMPUTER SCIENCE ARTIFICIAL INTELLIGENCE'

¹² This search used no topic filters.

validation set. Becker experimented in improving comprehensibility by lowering the complexity of solutions with positive results (Becker & Seshadri, n.d.).

Interval arithmetic and Linear scaling are presented by Keijzer(Keijzer, 2003) as a form of reducing overfitting in genetic programming applied to symbolic regression. One interesting part of this work are the tests made on protected operators, where, they show that protected operators are a problem.

Ivo Gonçalves (Goncalves & Carreiras, 2012) described a technique called 'Interleaved Sampling of Training Data' that changes the training set to present several small training sets to the evolutionary algorithm at a time in order to improve the generalization capacity.

Tuite uses a validation set to detect overfitting and then stops the evolution of the program completely to avoid further overfitting(Tuite, Agapitos, O'Neill, & Brabazon, 2011).

The concept of repulsor was described in the article by L. Vanneschi and S. Gustafson(Vanneschi & Gustafson, 2009). They used what they called repGP, '*repGP is similar to standard GP (stdGP from now on), except that it maintains a list L of individuals, called repulsors.*'. The concept of repulsor as described in this article is the base for this dissertation.

Passing knowledge from a validation set into the learning phase of a GP algorithm is usually frowned upon, due to the bias it introduces, but, in this case, the main objective is introducing favorable bias into the algorithm to produce fitter individuals.

With these papers available, we believe there is a need to further the study of repulsor based algorithms, and to study their behavior in several types of datasets.

Part of the problem resides on detecting when a program is overfitting; some work has been done in this are by Vanneschi, Castelli and Silva.(Vanneschi, 2010), this dissertation will use a different approach.

4. OBJECTIVES AND RESEARCH QUESTIONS.

4.1. OBJECTIVES

4.1.1. Main objectives.

The purpose of this dissertation is to analyze and document the use of repulsors in the selection phase of a genetic program, understand the dynamics of the repulsor based GP, compare repulsor based GP with classic GP, understand the effect on overfitting and try to improve the solutions found by GP.

These objectives will be accessed through several runs of the program with different parameters and using different functions generated and real-life datasets, and, compare the results obtained with classic GP.

4.1.2. Additional Objectives.

The creation of software that can later be used/adapted to other purposes and research in this field of study, with a common base so that later works can have comparable results. The objective of the software is to generate and collect data to document GP behavior when solving symbolic regressions, so the program will produce a vast amount of log data that will hinder its performance when solving the regression.

Contribute to the understanding of GP generalization.

4.2. RESEARCH QUESTIONS

- Can the use of repulsors affect/limit the overfitting effect that occurs in genetic programming? If so, what kind of effect will it have?
- Does the use of Repulsors have an impact on the ability to learn of a GP?
- Does the use of Repulsors affect the capacity to generalize?

5. METHODOLOGY

To test the use of repulsors a program was developed from scratch using python. The main steps are: Program Development and validation, Program runs and data collection and, Result analysis.

- 1) Program development and validation – Program is developed in python, according to the pre-set specification to the specifications, validation is performed against known results, like the sin dataset.
- 2) Perform the runs with different algorithms, values on the overfitting detection system and in different datasets.
- 3) Analyze and compare the collected results, reporting all obtained results.

The research philosophy adopted leans to positivism, we observe with a neutral standpoint, and perform a quantitative research. The methodology will be experimentation, and the techniques to test the hypothesis will be benchmarking against Standard GP supported by statistical significance testing.

5.1. SOFTWARE DEVELOPMENT.

The decision of developing the software from scratch instead of using pre-packaged one derives from the need to be able to modify the genetic program operations at will, have control and knowledge on how the steps are executed and to gain further knowledge of the inner workings and subtleties of the behaviors of GP.

5.2. PROCEDURE

The software will run 30 times for each dataset, algorithm, and setting totaling 1890 runs.

5.3. DATA COLLECTION.

Data is generated and stored in text files and MySQL database to be further analyzed.

5.4. RUN ANALYSIS.

- Summary statistics, mean, median, min max of the training and test sets.
- Number of repulsors found.
- Correlation analysis – Look for correlations between results.
- Detailed analysis of the runs on each dataset, with:
 - o Summary statistics.
 - o Repulsors found.
 - o Individual sizes and depth.
 - o Evolution by generation.
 - o Boxplot of fitness.
 - o Statistics relevance with Kruskal-Wallis test.

The Statistic test will be used as a support for the conclusions obtained from the fitness curves analysis. The statistic test chosen was Kruskal-Wallis because we needed a nonparametric test as the data does not follow a normal distribution. We will measure the performance of the algorithms on the minimization problem, both in the training set and test set, so we can measure its ability to learn and to generalize.

6. GENETIC PROGRAMMING SOFTWARE

6.1. MAIN GENETIC PROGRAMMING SOFTWARE

For the baseline genetic program, the software implements a standard approach to GP similar to the ones described by Koza in Genetic programming (Koza, 1992) and by Poli in 'The field guide to genetic programming.' (Poli et al., 2008). The main objective of the program is to document GP behavior when solving symbolic regressions, so it produced a vast amount of log data (some of it duplicate and stored in multiple locations) that hindered its performance when solving the regression. This performance penalty was a known and accepted compromise.

The programming style focused on readability above speed. The objective is a self-documenting code the closest to pseudo-code as possible.

Software was developed in Python 3,

Python version is Python 3.5.2 | Anaconda 4.2.0 (64-bit) | (default, July 5 2016, 11:41:13) [MSC v.1900 64 bit (AMD64)] on win32.

Python was chosen due to the simplicity and ease of coding and considering that is one of the major languages used in data science. Version 3 was chosen as it is the most current. Ease of use, the verbosity of the language and generalized use in data science were also considered when selecting Python.

All code has been written avoiding, where possible, the use of specific language constructs or techniques that cannot be replicated in other common languages such as JAVA, to achieve this, the code was written as close to the pseudo-code form of the function as possible. Readability is always preferred to speed when writing the program. The goal was self-documenting code that anyone with limited programming knowledge can read and understand what is being done. (McConnell, 2004). Some care will be taken so that the program can be easily parallelized so that it can be later used with parallel processing platforms such as GPUs (Langdon & Banzhaf, 2008) or Hadoop MapReduce and spark platforms.

Logging was done into text files and a MySQL database.

The programs source code can be obtained at <https://github.com/jcanelhas/repulsors01>

6.1.1. Algorithm validation and tests.

To validate the program results datasets with straightforward and predictable equations were used, the sin function presented in the '*fittable_sin*' data set was the main test data set. This function is well known, predictable as is also used in 'A field guide to genetic programming' (Poli et al., 2008) that provided a sound basis for tests.

Tests show expected predictable results in both results and learning rate and sine function approximation behavior, so we assume that the algorithm has been correctly implemented.

6.2. INPUTS AND OUTPUTS

6.2.1. Inputs.

Inputs are text files, separators are semi colons ';' and decimal symbol is the point '.'.

Equation 7 shows the standard input, where the Xs represent the inputs and the y the output variables.

Equation 7

$$x_0; x_1; x_2; \dots; x_n; y$$

6.2.2. Outputs

The program outputs the results to plain text files and a MySQL database. The objective is to have the data to be analyzed in the database and have all the outputs and generations in text files. With this approach, if more data is needed, it can be extracted from the text files and place it in the database without having to rerun the program.

Each run writes its outputs into a folder with the name matching the RUNID; this RUNID is a text representation of the timestamp of when the run was started the, inside this directory lie several other directories and files: File Samples and Formats can be found in the annexes.

Table 6 – File List.

Name	Location	Description
Fitfile_%id%.txt	root	Includes training, validation and test fitness. One row per generation.
Hallofshame.txt	root	Repulsors found on the run.
Overfit.txt	Root	Training, validation and test fitness on the iterations where overfitting was detected.
Regcomparisson.txt	Root	Expected and Predicted regression result per iteration.
Resultfmla.txt	Root	The best program found.
Runparameters.csv	Root	Parameters used in the run.
Testset.txt	Root	Test set used in the run
Trainingset.txt	Root	Training set employed in the run
Validationset.txt	Root	Validation set used in the run
Workfile.csv%id%.txt	Root	Information for each generation contains Header with more relevant parameters, then one entry per generation, these entries have time stamps and show the best values for the current iteration as well as the best individuals found so far and in the current iteration

Table 7 –Directory List

Name	Description
generations	Contains the generations that are created, each generation is a text file of the Python list. With big populations, this directory grows extremely fast.
graphs	Contains .dot files of the best program achieved in each generation, this file can be opened with Graphviz or another program/site that can handle this file. One text file is created for each generation.
plots	This directory contains one PNG file for each generation of the training regression results.
resulttables	Contains files with the results of the program in all sets (training, test, and validation) on all iterations

All the data used for the analysis is also available in the MySQL database, this makes the data processing faster and simpler if needed, the text files can be used to retrieve more information without having to re-rerun the program.

Table 8 –MySQL Table List

Name	Description
tbl_repulsors	Repulsors found in the runs.
tbl_runs	Run data
tbl_runs_detail	Run data by iteration
tbl_runs_parameters	Run parameters
tbl_runs_populations	Generated population per iteration and run (debug only)

6.2.1. Data processing and analytics software.

The run data is collected during the execution and is then processed using analytics software.

Most of the data manipulation was made using KNIME(Berthold et al., 2009). Statistics were calculated using KNIME workflows and validated when needed with R-Studio.

The Kruskal-Wallis test was performed in both R studio and KNIME with concordant results.

6.3. IMPLEMENTATION AND OPERATION

6.3.1. Primitives/Operators.

The program will use four primitives or operators: addition '+', subtraction '-', multiplication '*' and protected division '//¹³', protected division was chosen despite its known drawbacks, the protected division operates as described in equation 8:

Equation 8

$$a//b = \begin{cases} \frac{a}{b} & \text{if } b \neq 0 \\ PDOut & \text{if } b = 0 \end{cases}$$

PDOut is a parameter, usually assigned the value of 1. This approach, lets us penalize the use of divisions by 0 simply by using a large number for *PDOut*.

6.3.2. Variables.

The variables will be selected randomly, with equal probability from a set of the size defined in the *NVARIABLES* parameter.

6.3.3. Constants.

When needed the program creates constants by using the *uniform* function to obtain a float in the interval [-5,5]. The *VARIABLERATIO* parameter controls the probability of constant generation.

6.3.4. Termination Criteria

If not interrupted, the program terminates if one of the following termination criteria occurs:

- The number of maximum generations defined in the *MAX_GENERATIONS* parameter is reached.
- If the fitness value is less than the value defined in the *FITNESSTHRESHOLD* parameter
- If an error occurs.

6.3.5. Individual and Population Structure.

The linear representation of the tree has been selected due to the simplicity in crossover and mutation operations; this representation is placed on a Python list. A population is a list of individuals.

¹³ For simplicity, the standard division symbol '/' will be used.

6.4. ALGORITHMS.

We based the Standard GP implementation on Kozas Standard GP, the repulsor algorithms developed have, in addition to the phases of classic GP (Selection and Variation), two more phases, the repulsor tournament phase, and the overfit detection phase.

The overfit detection is a phase where the individuals are tested to see if they perform poorly on the validation set. When the overfit detection system detects one individual, it places it on the repulsor list. This repulsor list is kept and passed to all succeeding generations.

The repulsor tournament phase selects the best individual from the current population found by performing a tournament like selection, where the fitness is the maximization of the distance of the individual to the ones in the repulsor list.

Several algorithms were developed, implemented and tested until a satisfactory implementation was achieved, the selected implementations are GP_REPULSORS_08 and GP_REPULSORS_09¹⁴.

6.4.1. Standard GP Algorithm Implementation.

This implementation is the standard GP based on the implementations described by Koza(Koza, 1992) and Poli(Poli et al., 2008) on their books. The selection method is tournament selection, and the mutation method is subtree mutation. In this implementation, the original dataset is split in Training and Test sets.

1. Generate random population
2. Repeat until a termination condition is found:
 - a. Repeat $n/2$ times where n is the size of the population.
 - i. Select elements by tournament selection.
 - ii. With a probability P_x perform crossover, otherwise, copy the individuals by reproduction.
 - iii. With a probability of P_m perform mutation
 - iv. Copy elements to new population
 - b. Replace original population with the new population.
 - c. Verify termination conditions.
3. Return best individual found.

The implemented Standard GP algorithm was named GP_CLASSIC and will be named as such in the result tables.

¹⁴ This naming convention results from the number of algorithms developed and tested, previous versions were kept and we also kept their numbers in the dissertation.

6.4.2. Repulsor Based GP Implementation.

This implementation extends Standard GP with three steps: the use of a validation set, overfit detection and Repulsor tournament. The original data set is split into three parts, training, validation and test sets (the test set is never used in the training process). The overfit detection system compares the current solution with the validation set, and if it detects overfitting, the current solution is added into the repulsor list. There is a new selection phase where the fitness is calculated by the semantic distance from results of the individuals in the repulsor list, the larger the distance the better.

6.4.2.1. GP Repulsors 08

The new selection phase occurs after the standard selection phase.

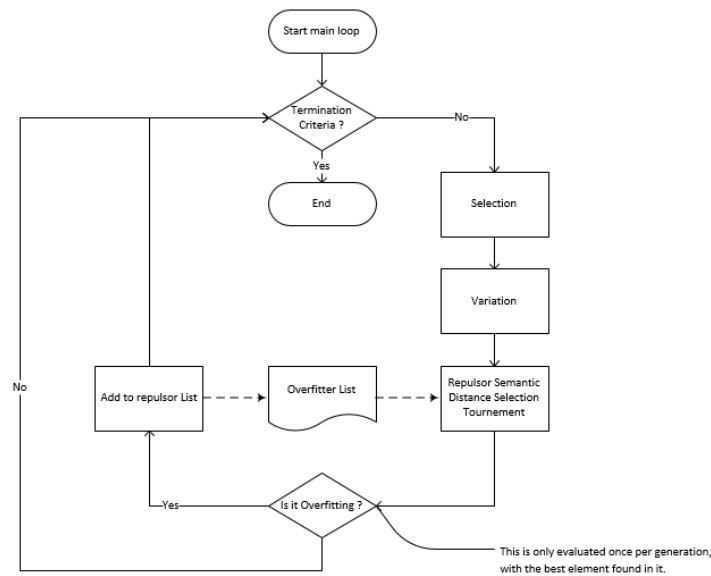


Figure 25:GP Repulsors 08 algorithm simplified diagram.

1. Generate random population
2. Repeat until a termination condition is found:
 - a. Repeat $n/2$ times where n is the size of the population.
 - i. Select elements by tournament selection.
 - ii. With a probability x perform crossover, otherwise, copy the parents using reproduction.
 - iii. With a probability of y do mutation
 - iv. Copy elements to new population
 - b. Replace original population with the new population.
 - c. Repeat n times where n is the size of the population.
 - i. Select element by tournament selection using the distance from the repulsor table.
 - ii. Copy the best element to the new population.
 - d. Replace population with new population
 - e. Verify if the generation's best element is overfitting, if so add it to the repulsor list.
 - f. Verify termination conditions.
3. Return best individual found.

6.4.2.2. GP Repulsors 09

The new selection phase occurs before the standard selection phase.

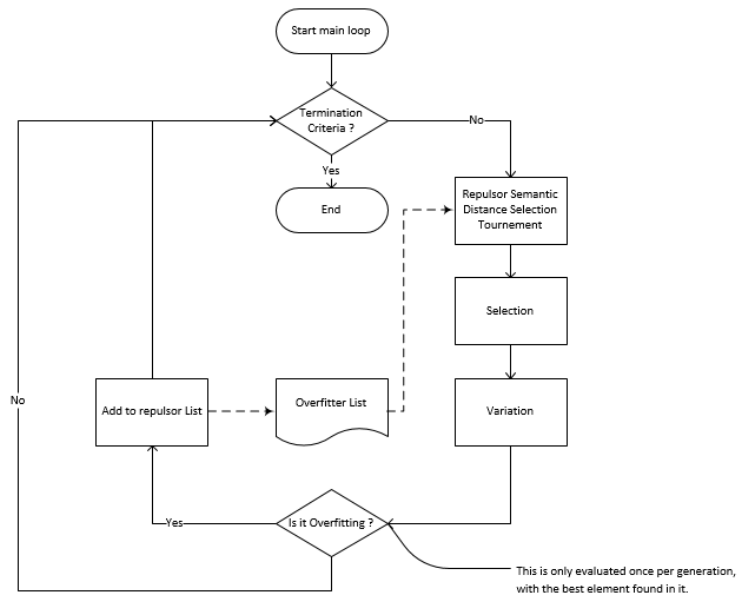


Figure 26:GP Repulsors 09 algorithm simplified diagram.

1. Generate random population
2. Repeat until a termination condition is found:
 - a. Repeat $n/2$ times where n is the size of the population.
 - i. Select element by tournament selection using the distance from the repulsor table.
 - ii. Copy the best element to a new population.
 - b. Replace population with new population
 - c. Repeat n times where n is the size of the population.
 - i. Select elements by tournament selection.
 - ii. With a probability x perform crossover, otherwise, perform reproduction.
 - iii. With a probability of y do mutation
 - iv. Copy elements to new population
 - d. Replace original population with the new population.
 - e. Verify if the generation's best element is overfitting, if so add it to the repulsor list.
 - f. Verify termination conditions.
3. Return best individual found.

6.5. FITNESS EVALUATION.

Functions used in the program for fitness evaluation.

6.5.1. Individual fitness on the training set.

For the classic GP parts, the fitness function is the root mean Square deviation, this function is of simple implementation and is widely used in symbolic regression problems.

Equation 9

$$RMSD = \sqrt{\frac{\sum_{t=1}^n (\hat{y}_t - y_t)^2}{n}}$$

Where $\hat{y}_t$ is the predicted value and y_t is the computed value for regression, n is the number of variables in the individual.

The smaller the RMSD, the better. We seek to minimize the difference.

6.5.2. Repulsor distance evaluation.

When using repulsors, every selected individual is evaluated against a validation set, if found unfit, using the overfit detection method explained next, the program adds it to a list of 'bad' solutions known as the repulsors list, from now on every individual is also tested against this list, we test the individuals for semantic similarity, the test is performed as the mean of the sum of the RMSD from the solution provided by the individual and the solutions present in the repulsors list.

Equation 10

$$Distance = \frac{1}{m} \sum_{j=1}^m \left(\frac{1}{n} \sum_{i=1}^n \sqrt{(a_{ji} - b_i)^2} \right)$$

Equation 10 presents the repulsor distance calculation where m is the number of repulsors, n the number of observations, a is the values of the repulsors and b the value of the individual.

The farther from the results of the repulsors in the list, the better. We seek to maximize the distance between the individual and the repulsors in the list.

6.6. OVERFITTING DETECTION METHOD.

Overfit detection is made against a validation set. The parameter OVERFITTHRESHOLD is used, if at any time the validation fitness is above the previous validation fitness times the OVERFITTHRESHOLD then we consider that the solution is overfitting. The overfit detection is made once per generation on the end of the generation.

$$\text{Equation 11} \quad \text{overfitting} = \begin{cases} 0 & \text{if : } \text{validationfitness} < \text{previousvalidationfitness} \cdot \text{OVERFITTHRESHOLD} \\ 1 & \text{if : } \text{validationfitness} \geq \text{previousvalidationfitness} \cdot \text{OVERFITTHRESHOLD} \end{cases}$$

In equation 11, *validationfitness* is the current generation fitness against the validation data set, *previousvalidationfitness* is the previous generation fitness against the validation dataset, and OVERFITTHRESHOLD is a real number.

Figure 23 shows the overfitting detection system at work. If the current iteration validation fitness is detected as overfitting the solution is added to the repulsor list. In methods GP_Repulsors08 and GP_Repulsors09 overfitting detection is done at the end of each generation.

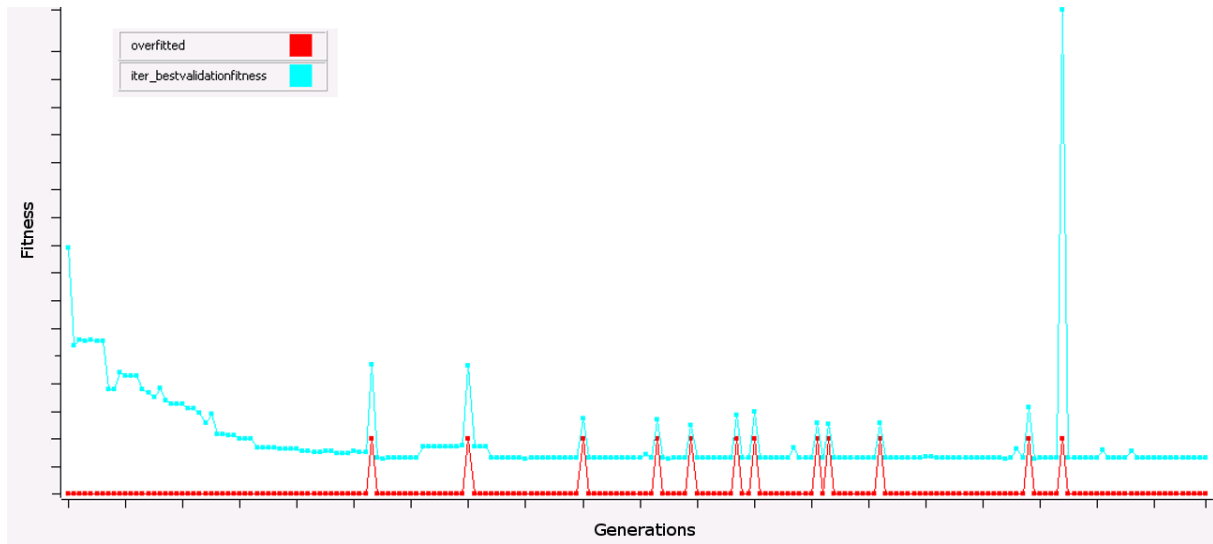


Figure 27: Overfit detection at 1.5 times the fitness difference

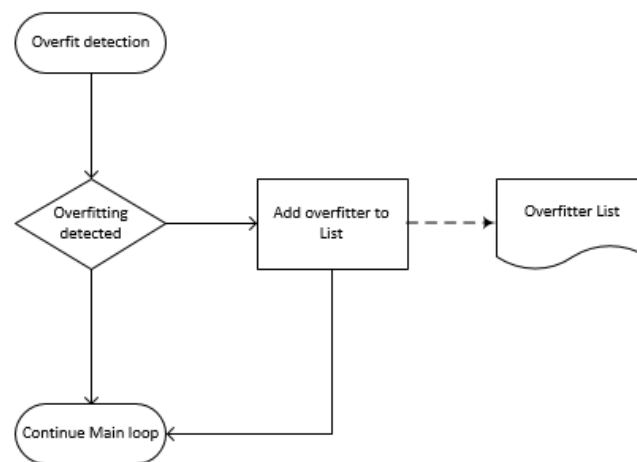


Figure 28: Overfit Detection System schematic.

6.7. PARAMETERS.

Table 9 – Genetic Program Parameters.

Parameter	Value	Description
Tree_generation_mode	RHaH	Population generation mode is ramped half and half
Elitism	TRUE	Use Elitism
Tournament size	4	Size of the GP selection tournament
Repulsor Tournament size	2	The size of the repulsors tournament.
Program Depth	6	Initial maximum program depth.
Max Program depth	12	Maximum program depth allowed
Crossover Rate	0.9	Crossover probability
Mutation Rate	0.1	Mutation probability
Mutation Mode	subtree	Type of mutation performed: subtree
Population size	200	Size of the population
Variable Ratio	0.99	The ratio of variables generated vs. constants, 99%.
OVERFITTHRESHOLD	1.5	Threshold when to consider overfitting is happening
	1.01	
	2.5	
FITNESSTHRESHOLD	10^{-8}	Value to consider the solutions has been found.
PENALIZEAASSIMPTOTES	FALSE	Return a penalization if safe division is used
TRAININGTESTRATIO	0.7	Training set to test set ratio
VALIDATIONSETRATIO	0.3	Validation set to training set ration
DATASETPERCENTAGETOUSE	Varies	The percentage of the original dataset that will be used.

We've set the values of global parameters based on various criteria; most of them were chosen based on the values employed by literature on the subject (Poli et al., 2008); some were selected by an educated guess, and others were the result of experimentation. Ramped half and half tree generation mode has been chosen to provide the best variety of trees to start, the size of population, tournaments and max tree size were chosen to keep the running time of the algorithm in reasonable limits. Crossover and mutation probabilities use a random uniform distribution provided by the python random library. The parameters were chosen knowing that some tradeoffs had to be made due to processing capacity and time needed to generate all the data.

There was also an accepted compromise in Parameter tweaking vs. unified testing, we made sure that we were only changing the OFT and Algorithm choice, and that no other parameter was changed during the testing, this can somewhat limit the outcomes of the runs as there was no parameter optimization based on the nature of the dataset.

6.8. DATASET PREPARATION AND USE.

6.8.1. Dataset Descriptions.

Two categories of data sets are used: Mathematical functions and regression datasets taken from the UCI Machine Learning Repository (Lichman, 2013). Function 05 and function 07 use random data in the inputs, the sin function and Function 08 that uses a continuous interval $[-5;5]$.

Table 10 – Mathematical Function datasets description.

File	Function applied
fittable_sin.txt	$\sin(x)$
fittable_fn05.txt	$x_0 \cdot x_1 \cdot x_2 + \frac{x_4}{x_5} \cdot x_6$
fittable_fn07.txt	$x_0 \cdot x_1 \cdot x_2 + \frac{x_3}{x_4} \cdot x_5 \cdot x_6 \cdot x_7 \cdot x_8 - x_9 - \frac{x_{10}}{x_{11}} \cdot x_{12} \cdot x_{13} \cdot x_{14} + x_{15}$
fittable_fn08.txt	$x_0 \cdot x_1 + \sin((x_0 - 1) - (x_1 - 1))$

Several symbolic regression data sets from the UCI (Lichman, 2013) were used, the data was prepared when needed to be imported into the program. All selected datasets had no missing values and are indicated for regression analysis. The datasets were chosen based solely on their characteristics.

Table 11 –UCI real-life regression datasets description.

File	Data Description
Winequality_red	Wine Quality red dataset.(Cortez, Cerdeira, Almeida, Matos, & Reis, 2009) ¹⁵
CCPP folds	Data points collected from a combined cycle power plant over six years. ¹⁶
Concrete slump	Dataset with concrete data, the output is the compressive strength. ¹⁷
Concrete	Concrete compressive Strength dataset.(Yeh, 1998). ¹⁸
LD50 toxicity	Lethal Dose of 260 compounds with known % F, 234 compounds with known LD50 and 662 compounds with known % PPB. ¹⁹

If needed, it is possible to use only a part of the dataset at any given time; the size is defined by a ratio of the original dataset set by parameter. In classic GP (DATASETPERCENTAGETOUSE), this will become the dataset that will be further split. At each run the needed samples will be picked at random. This new data set is then again randomly divided into test and training sets in a predefined proportion (a ratio set in the parameters, TRAININGTESTRATIO), if repulsors are used then the set is once again split into validation and training sets (again set by a parameter, VALIDATIONSETRATIO). This kind of division ensures that the test set size is the same and the validation and training sets contain the same number of elements that the training set had in classic GP. The elements in the test set are NEVER used in the training process.

¹⁵ Available at : <https://archive.ics.uci.edu/ml/datasets/Wine>.

¹⁶ Available at : <https://archive.ics.uci.edu/ml/datasets/Combined+Cycle+Power+Plant>

¹⁷ Available at : <https://archive.ics.uci.edu/ml/datasets/Concrete+Slump+Test>.

¹⁸ Available at <https://archive.ics.uci.edu/ml/datasets/Concrete+Compressive+Strength>.

¹⁹ Available at : <http://personal.disco.unimib.it/Vanneschi/toxicity.txt>

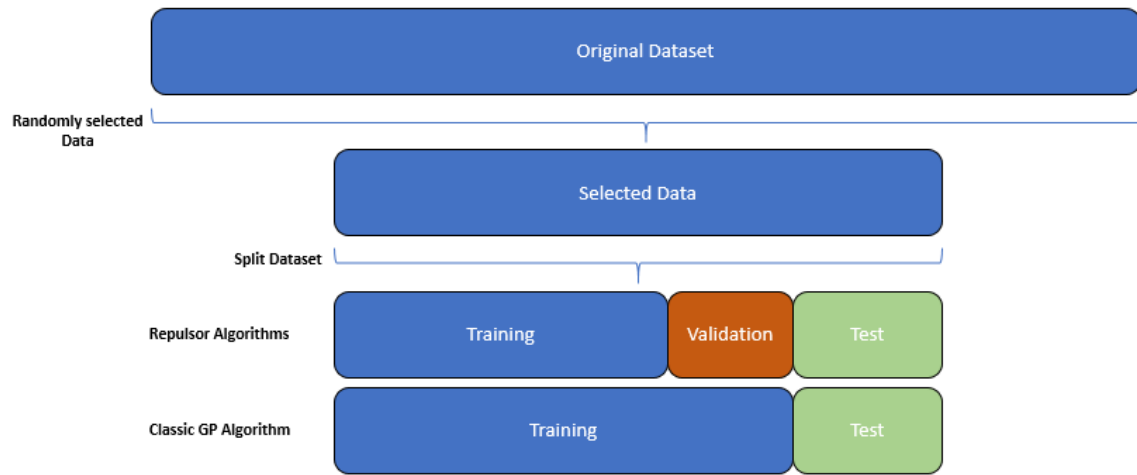


Figure 29: Data set partitioning.

6.8.2. Dataset Sizes.

Function generated based data sets were created with 100 samples in each. When processing the real-life data sets, whenever possible, we used all the available dataset, however, due to computational and time constraints, some datasets were truncated, the sample selection is random and performed at the start of the run, the DATASETPERCENTAGETOUSE parameter controls the size.

Table 12 – Mathematical Function datasets sizes.

Data set		Set Size			
Name	Dimensions	Test	Train Classic	Train Reps	Validation
fittable_sin.txt	1	30	70	49	21
Fittable_fn05.txt	6	30	70	49	21
Fittable_fn07.txt	16	30	70	49	21
Fittable_fn08.txt	2	30	70	49	21

Table 13 – UCI real-life regression datasets sizes.

Data set		Set Size			
Name	Dimensions	Test	Train Classic	Train Reps	Validation
winequality-red	11	48	111	78	33
CCPP	4	58	133	94	39
concrete_slump	7	31	71	51	21
Concrete	8	309	711	505	206
Toxicity	626	71	163	115	48

Runs:

To obtain the data for analysis, the algorithms were run 30 times for each dataset, and each overfit threshold setting producing a total of 1890 runs to be used in this analysis.

7. RESULTS AND DISCUSSION

7.1. GLOBAL RESULT ANALYSIS.

This section compares the final results obtained with the nine datasets using GP CLASSIC and GP REPULSORS we will analyze centrality measures, compare them. Then we will compare the number of repulsors found.

Results presented are related to a minimization problem, the lower the values, the better the fitness.

7.1.1. Training fitness Centrality Measures.

Tables 13 and 14 show the mean and median fitness of the best solutions found on 30 runs on the training set.

Table 14 – Mean Training Set Fitness at iteration 200.

Algorithm	Classic	Repulsors 08			Repulsors 09		
Data Set / O.F.T.	N.A.	1.01	1.5	2.5	1.01	1.5	2.5
winquality-red	0.598	0.766	0.68	0.603	0.827	0.693	0.572
CCPP	9.536	10.976	11.328	11.404	12.094	14.046	8.753
Concrete slump	3.902	6.851	4.911	4.075	6.363	5.646	4.382
Concrete	10.588	12.074	9.901	9.479	12.36	9.766	9.819
LD50 Toxicity	1707	1797	1664	1611	1841	1796	1643
Fittable_sin	0.379	0.363	0.456	0.413	0.389	0.369	0.48
Fittable_fn05	0.116	0.311	0.144	0.289	0.406	0.391	0.22
Fittable_fn07	0.803	0.755	0.921	0.834	0.861	1.079	0.839
Fittable_fn08	0.632	0.627	0.619	0.634	0.623	0.6	0.599

Table 15 –Median Training Set Fitness at iteration 200.

Algorithm	Classic	Repulsors 08			Repulsors 09		
Data Set / O.F.T.	N.A.	1.01	1.5	2.5	1.01	1.5	2.5
winquality-red	0.604	0.752	0.647	0.599	0.809	0.683	0.563
CCPP	8.943	8.447	9.815	11.631	7.205	11.829	7.609
Concrete slump	3.552	6.349	4.794	3.384	6.062	4.737	3.951
Concrete	10.392	11.87	9.082	9.137	12.122	8.734	8.96
LD50 Toxicity	1661	1797	1659	1601	1757	1762	1602
Fittable_sin	0.428	0.317	0.514	0.463	0.409	0.417	0.51
Fittable_fn05	0.068	0.198	0.164	0.182	0.216	0.216	0.156
Fittable_fn07	0.546	0.684	0.549	0.631	0.685	0.831	0.556
Fittable_fn08	0.647	0.631	0.609	0.653	0.61	0.601	0.604

In the training set repulsor based GP appears to outperform classic GP. This is supported by the mean and median values, in 7 out of 9 cases the lower values were found by GP repulsors, this means that using repulsors improves the learning ability of the algorithm.

7.1.2. Test fitness Centrality Measures.

Tables 15 and 16 show the mean and median fitness of the best solutions found on 30 runs on the test set.

test set.

Table 16 – Mean Test Set Fitness at iteration 200.

Algorithm	Classic	Repulsors 08			Repulsors 09		
Data Set / O.F.T.	N.A.	1.01	1.5	2.5	1.01	1.5	2.5
winquality-red	3.3	0.801	1.02	1.38	0.921	1.06	2.242
CCPP	16.873	11.275	16.796	26.61	13.188	42.081	21.695
Concrete slump	8.482	14.651	6.994	12.662	7.638	11.016	6.35
Concrete	13.231	12.606	13.311	10.237	14.189	10.34	10.663
LD50 Toxicity	2805	3603	3978	4393	3448	2796	4571
Fittable_sin	0.52	0.974	7.1	0.932	0.917	0.444	1.525
Fittable_fn05	0.159	0.434	0.274	0.835	0.481	0.39	0.411
Fittable_fn07	1.706	1.569	1.762	1.808	1.417	1.988	2.726
Fittable_fn08	0.723	0.843	1.091	0.695	0.88	0.973	0.932

Table 17 – Median Test Set Fitness at iteration 200.

Algorithm	Classic	Repulsors 08			Repulsors 09		
Data Set / O.F.T.	N.A.	1.01	1.5	2.5	1.01	1.5	2.5
winquality-red	0.778	0.807	0.815	0.867	0.847	0.852	0.877
CCPP	11.48	8.314	10.545	13.538	7.975	13.347	8.909
Concrete slump	4.365	7.731	6.029	5.475	7.001	8.228	5.817
Concrete	10.739	12.671	9.383	9.922	12.333	9.295	9.804
LD50 Toxicity	2408	2420	2514	2539	2248	2252	2443
Fittable_sin	0.472	0.604	0.61	0.556	0.468	0.485	0.608
Fittable_fn05	0.093	0.208	0.139	0.22	0.251	0.199	0.189
Fittable_fn07	1.196	1.263	1.619	1.546	1.227	1.652	1.388
Fittable_fn08	0.671	0.721	0.703	0.662	0.683	0.707	0.715

In the test set repulsor based GP has better performance in 8 out of 9 cases, if the median is used to test the centrality then the results better with repulsors in 4 out of 9 of the datasets. This supports that repulsors achieve better results than GP Classic when generalizing.

7.1.3. Global Centrality Measures comparison.

Table 18 shows what algorithm type was used to achieve the best median result of 30 runs on each dataset. Xs mark the algorithm that performed better in Train and test sets, if an algorithm performed better in both datasets then it is considered best.

Results support that when applied to 9 data sets, in 5 of them Repulsor GP based algorithms performed better on both training and test set, meaning that not only do they learn better but also generalize better.

Table 18 – Algorithm where the Best Median was found.

Data Set / O.F.T.	Train		Test		Best
	CLASSIC GP	REP GP	CLASSIC GP	REP GP	
winquality-red		X	X		-
CCPP		X		X	REP GP
Concrete slump		X	X		-
Concrete		X		X	REP GP
LD50 Toxicity		X		X	REP GP
Fittable_sin		X		X	REP GP
Fittable_fn05	X		X		CLASSIC GP
Fittable_fn07	X		X		CLASSIC GP
Fittable_fn08		X		X	REP GP

7.1.4. Best element.

Table 19 shows Median Generation where the best solution was found.

Table 19 – Generation where the best solution was found (training set).

Method	Standard	Repulsors 08			Repulsors 09		
Data Set / O.F.T.		1.01	1.5	2.5	1.01	1.5	2.5
CCPP	199	198	199	199	193	197,5	199
winequality-red	198	117	193	197	108	142,5	189,5
concrete	199	195,5	199	200	196	197	199,5
concrete_slump	198	196,5	198	198,5	191	194,5	197,5
Fittable_fn05.txt	144	132	148	120	139	103,5	110
Fittable_fn07.txt	194	182,5	137	161,5	164,5	61	135,5
Fittable_fn08.txt	92,5	148,5	140	107	145,5	166	145,5
fittable_sin.txt	181,5	185,5	170,5	192,5	192	175,5	164
LD50 Toxicity	199,5	197	198	198	196	187,5	199

Table 18 by itself does not guarantee that the element found in earlier iterations is better than an element found in a later iteration using a different method.

7.1.5. Number of Repulsors Found.

Table 20 – Mean number of Repulsors found.

Method	Rep 08			Rep 09		
Data Set / O.F.T.	1.01	1.5	2.5	1.01	1.5	2.5
winequality-red	79,467	16,333	3,267	76,067	11,167	1,967
CCPP	58,567	9,733	1,300	82,100	6,733	0,733
concrete_slump	61,567	6,600	0,767	79,633	3,367	0,533
concrete	53.433	0.967	0.4	72.667	0.6	0.467
LD50 Toxicity	45.033	6.933	0.967	55.033	2.633	2.1
fittable_sin.txt	28,567	6,133	1,600	50,167	5,767	2,133
fittable_fn05.txt	70,233	36,767	13,233	58,567	16,867	4,667
fittable_fn07.txt	80,633	10,233	5,000	77,733	13,433	6,400
fittable_fn08.txt	19,367	4,867	0,100	21,233	4,333	1,300

Table 21 – Median number of Repulsors found.

Method	Rep 08			Rep 09		
Data Set / O.F.T.	1.01	1.5	2.5	1.01	1.5	2.5
winequality-red	82	11,5	0	76	6	1
CCPP	60	0	0	83,5	1,5	0
concrete_slump	63	3	0	78,5	3	0
concrete	51.5	0	0	75	0	0
LD50 Toxicity	41	2.5	0	55	1	0
fittable_sin.txt	30	2	0	54,5	1	0
Fittable_fn05.txt	75,5	38,5	9,5	64	13,5	2,5
Fittable_fn07.txt	89	4,5	0	93,5	8,5	5
Fittable_fn08.txt	14,5	0	0	8,5	1	0

It is intuitive and demonstrated experimentally that the higher the overfit detection threshold, the fewer the number of repulsors found, this behavior is replicated in all analyzed datasets and algorithms. This is supported by a correlation test detects an inverse correlation between overfit threshold and the number of repulsors found.

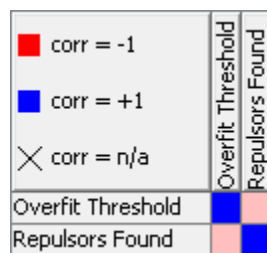


Figure 30: Repulsor count correlation Matrix

7.2. DETAILED ANALYSIS OF INDIVIDUAL DATASET RESULTS.

Each dataset will be detailed in the items:

- 1) Summary statistics for training and test sets – Summary statistics of test and train dataset, the training set result provides us with information on the training ability of the algorithms, and is useful for determining learning rates and for problems where all the data is known. The test set results provide us insights on the generalization ability of the algorithm.
- 2) Number of repulsors found – Median of the repulsors found on the run, the mean can be found on the annexes.
- 3) Best Individual Iteration is the iteration number where the best solution was found for the training set – The average of the generation number where best training result found, this will provide insight on when the run finished the learning process.
- 4) Individual size and depth – Size analysis.
- 5) Centrality measure graphical analysis – Line charts of the evolution of the mean and median through the 200 generations of the runs.
- 6) Conclusions and regression comparison for the individual – Some insights in the results of the run, box plot of the test set results to see how coherent they are, We will also use the Kruskal-Wallis test for statistical validation on the differences in the results for the used algorithms. A correlation table is also used to check for correlations in the results.

For the Kruskal-Wallis test the hypotheses are:

Null Hypotesys: H_0 : There is no difference in the outcome of GP when using GP CLASSIC or GP REPULSORS algorithms.

Alternative Hypotesys : H_a : The results obtained using GP CLASSIC and GP REPULSORS are different.

The Kruskal-Wallis test is merely used as a support for the observations made in the graphical comparison.

We also picked some results for correlation analysis,

Table 22 – Correlation analysis field descriptions.

Field	Description.
Overfit threshold	Parameter to control when the algorithm considers that a solution is overfitting.
Solution Depth	The Depth of the solution tree.
Solution Size	The size of the solution i.e., the number of elements in it.
Besttrainfitness	The best train fitness achieved.
Besttestfitness	The best test fitness achieved.
Last Improvement	The last iteration where the solution was improved.
Repulsors Found	The number of repulsors found.

7.2.1. Detailed Analysis of the Fittable_sin dataset.

The fittable_sin dataset consists of a single variable regression result of applying the sin function to elements of the interval [-5;5], with a stepping of 0.1 between each value.

Equation 12

$$f(x) = \sin(x)$$

This function was used as a test during the development of the program due to its known behavior, and simplicity to visually and quickly check the results.

7.2.1.1. Summary statistics.

Table 23 – fittable_sin summary statistics for the Training set.

Algorithm	O.F.T.	Min.	Max.	Mean	Median	S.D.	Var.
GP_CLASSIC	N/A	0.094	0.616	0.379	0.428	0.170	0.029
GP_REPULSORS_08	1.01	0.093	0.722	0.363	0.317	0.192	0.037
GP_REPULSORS_08	1.5	0.039	0.734	0.456	0.514	0.197	0.039
GP_REPULSORS_08	2.5	0.104	0.705	0.413	0.463	0.169	0.028
GP_REPULSORS_09	1.01	0.110	0.697	0.389	0.409	0.172	0.030
GP_REPULSORS_09	1.5	0.097	0.709	0.369	0.417	0.202	0.041
GP_REPULSORS_09	2.5	0.114	0.695	0.480	0.510	0.150	0.022

Table 24 – fittable_sin summary statistics for the Test set.

Algorithm	O.F.T.	Min.	Max.	Mean	Median	S.D.	Var.
GP_CLASSIC	N/A	0.113	1.821	0.520	0.472	0.360	0.130
GP_REPULSORS_08	1.01	0.098	10.470	0.974	0.604	1.844	3.401
GP_REPULSORS_08	1.5	0.046	196.394	7.100	0.610	35.753	1278.290
GP_REPULSORS_08	2.5	0.126	8.455	0.932	0.556	1.560	2.435
GP_REPULSORS_09	1.01	0.164	9.942	0.917	0.468	1.811	3.281
GP_REPULSORS_09	1.5	0.095	1.439	0.444	0.485	0.306	0.094
GP_REPULSORS_09	2.5	0.147	13.265	1.525	0.608	2.922	8.537

The best value achieved in training was 0.039 using algorithm GP_REPULSORS_08 with OFT 1.5, it is more than two standard deviations below the mean of the 30 runs used, the run that produced this solution scored equally high in the test set returning a fitness of 0.046, this performance is twice the result achieved by the second best. This is, however, a unique result from a series of runs that present a very high standard deviation and variance. Considering the total of 30 runs, the better performing Algorithm in the training set was GP_REPULSORS_08 with OFT 1.01; this did not, however, correspond to the same results in the test set. The better-performing algorithms in the test set, using the 30 runs, are the GP_CLASSIC with a median of 0.472 and GP_REPULSORS_09 with a mean of 0.444.

7.2.1.2. Repulsors.

Table 25 – Median number of repulsors found, fittable_sin.

Algorithm	O.F.T.		
	1.01	1.5	2.5
GP_REPULSORS_08	20	2	0
GP_REPULSORS_09	54.5	1	0

The number of repulsors diminishes as the overfit threshold increases. GP_REPULSORS_09 found approximately the double of repulsors when the OFT was set to 1.01 (high sensibility to overfitting), this may be an indication that the order of the selection phases (standard and repulsor based) has some effect on the outcome.

7.2.1.3. Best Individual Generation.

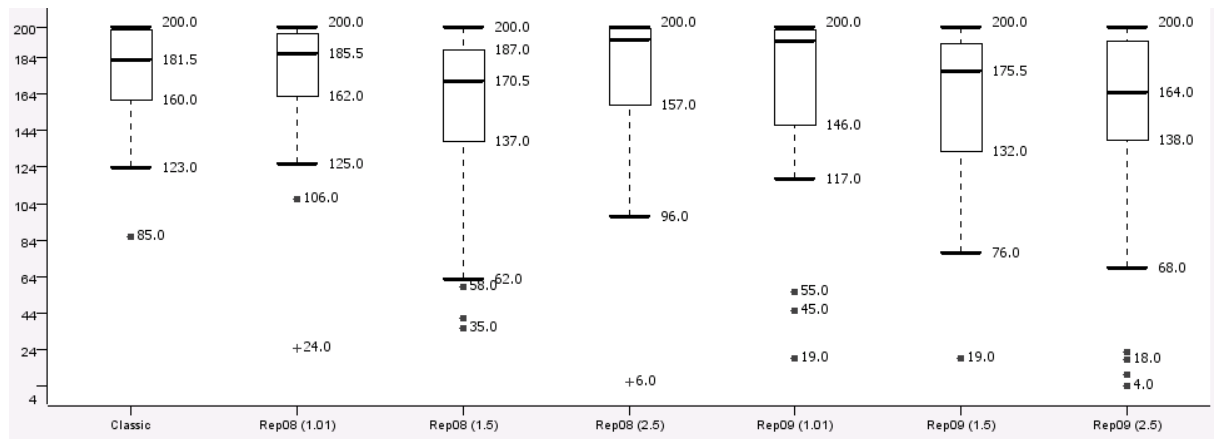


Figure 31:fittable_sin, Best Individual Generation.

The learning median stays above and the all the distributions are skewed to the maximum size, meaning that the training ended at a high generation.

7.2.1.4. Individual Size.

Table 26 – fittable_sin, Individual depth and size median.

Algorithm	Classic	Repulsors 08			Repulsors 09		
		1.01	1.5	2.5	1.01	1.5	2.5
Measure / O.F.T.	N.A.						
Tree depth	10	10	10	10	10	10	10
Individual depth	187	193	200	158	168	184	186

The programs tended to grow to the depth limit allowed; this can be seen by the median tree depth of 10 levels.

7.2.1.5. fittable_sin centrality Measures Graphical Comparison.

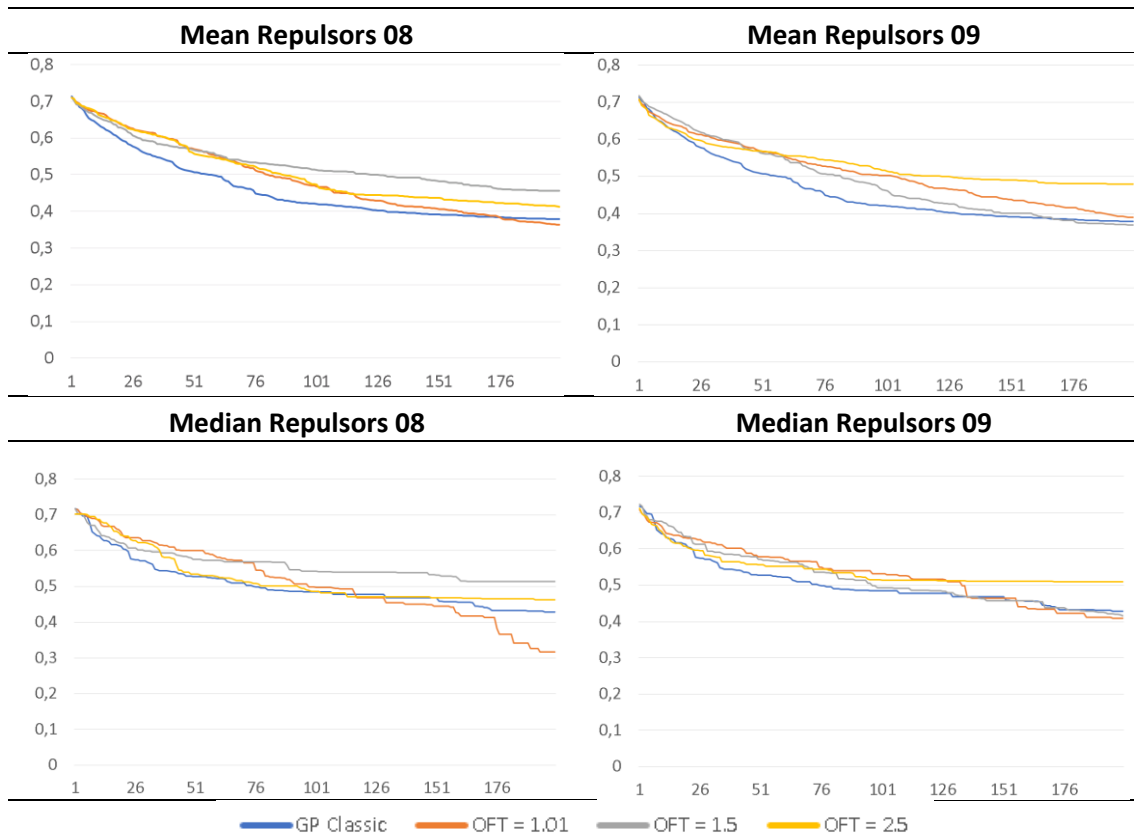


Figure 32: fittable_sin, training set graphical centrality comparison.

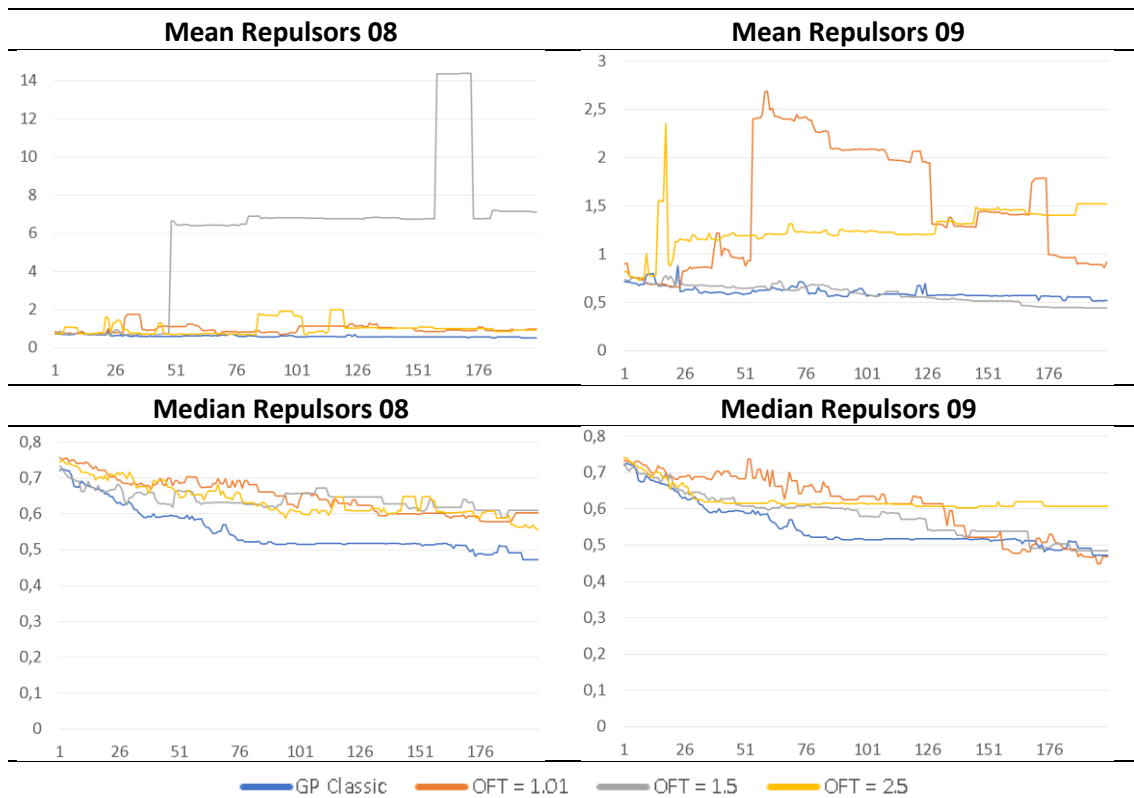


Figure 33: fittable_sin, test set graphical centrality comparison.

7.2.1.6. Individual Conclusions for fittable_sin.

There is a clear difference in the learning rate of GP_REPULSORS_08 with OFT 1.5 that is also present but less pronounced in GP_REPULSORS_09. This may be related to the number of repulsors found. In the test set. It was the best performer on both centrality measures of 30 runs, in all scenarios but the median of GP_REPULSORS_09 OFT 1.01 that managed to produce a marginally better result. The mean line charts indicate severe overfitting of some solutions of GP_REPULSORS_09 OFT 1.01 and 2.5 and GP_REPULSORS_08 OFT 1.5

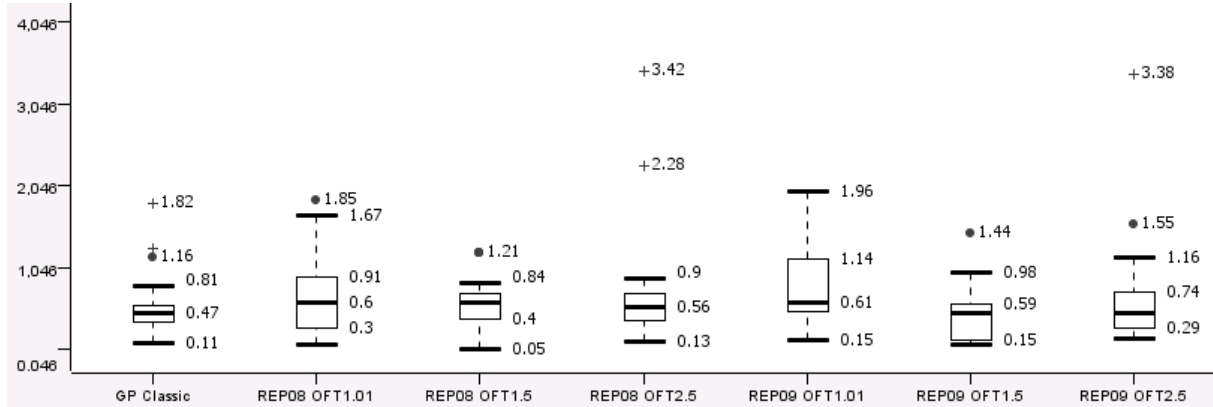


Figure 34: Boxplot of the best fitness on the fittable_sin test set.

Table 27 – Kruskal-Wallis Test for fittable_sin.

Method	Train Pv	Test Pv
GP Repulsors 08	0.506	0.655
GP Repulsors 09	0.260	0.055

A Kruskal-Wallis test indicates that with a significance level of 0.10 GP_REPULSORS_09 has statistically different medians. Just barely exceeding the p-value acceptable for a 0.05 significance on the test set

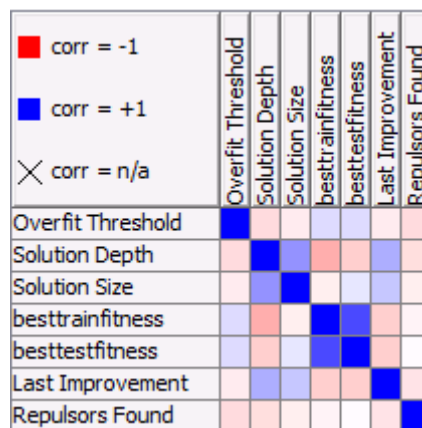


Figure 35: Correlation matrix for fittable_sin dataset.

Training and test fitnesses show a positive correlation; Training fitness is negatively correlated with the depth of the solution.

7.2.2. Detailed Analysis of the Fittable_fn05 dataset

The fittable_fn05 is a dataset created using the function:

$$\text{Equation 13} \quad f(x) = x_0 \cdot x_1 \cdot x_2 + \frac{x_4}{x_5} \cdot x_6$$

There are 100 samples in the dataset, the variables $x_0, x_1, \dots, x_6$ have random values in the closed interval $[-1, 1]$.

7.2.2.1. Summary statistics.

Table 28 – fittable_fn05 summary statistics for the Training set.

Algorithm	O.F.T.	Min.	Max.	Mean	Median	S.D.	Var.
GP_CLASSIC	N/A	4.794E-09	0.943	0.116	0.068	0.179	0.032
GP_REPULSORS_08	1.01	5.220E-09	1.277	0.311	0.198	0.385	0.148
GP_REPULSORS_08	1.5	3.161E-09	0.486	0.144	0.164	0.115	0.013
GP_REPULSORS_08	2.5	3.715E-09	1.147	0.289	0.182	0.348	0.121
GP_REPULSORS_09	1.01	0.049	1.342	0.406	0.216	0.396	0.157
GP_REPULSORS_09	1.5	8.622E-09	1.297	0.391	0.216	0.386	0.149
GP_REPULSORS_09	2.5	3.587E-09	0.856	0.220	0.156	0.233	0.054

Table 29 – fittable_fn05 summary statistics for the Test set.

Algorithm	O.F.T.	Min.	Max.	Mean	Median	S.D.	Var.
GP_CLASSIC	N/A	1.988E-09	1.272	0.159	0.093	0.261	0.068
GP_REPULSORS_08	1.01	9.662E-09	2.148	0.434	0.208	0.535	0.286
GP_REPULSORS_08	1.5	2.105E-09	1.445	0.274	0.139	0.392	0.154
GP_REPULSORS_08	2.5	4.819E-09	14.271	0.835	0.220	2.608	6.799
GP_REPULSORS_09	1.01	0.053	1.525	0.481	0.251	0.449	0.201
GP_REPULSORS_09	1.5	1.168E-08	1.189	0.390	0.199	0.332	0.110
GP_REPULSORS_09	2.5	5.698E-09	1.771	0.411	0.189	0.516	0.266

Only one of the tested algorithms failed to find the solution for the problem described in this dataset, the minimum is in all cases below $10E-7$, there is no point in singling any algorithm out for achieving the minimum value as all of them are near 0 and close together, any difference can be due to rounding errors induced in the code.

In both centrality measures, the best performer was the GP_CLASSIC algorithm.

7.2.2.2. Repulsors.

Table 30 – fittable_fn05, median number of repulsors found.

Algorithm	O.F.T.		
	1.01	1.5	2.5
GP_REPULSORS_08	75.5	38.5	9.5
GP_REPULSORS_09	64	13.5	2.5

The number of repulsors diminishes as the overfit threshold increases. This behavior is expected.

7.2.2.3. Best Individual Generation.

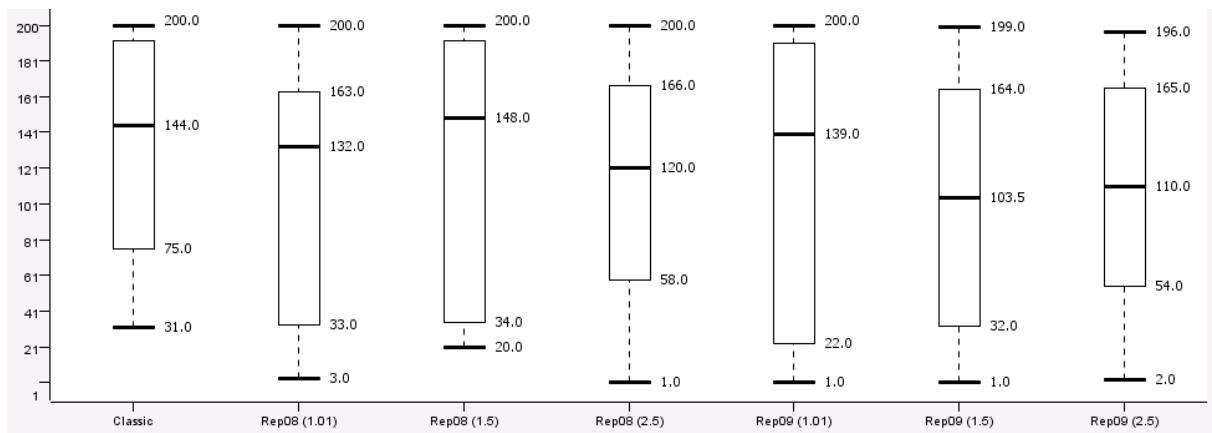


Figure 36:fittable_fn05, Best Individual Generation.

The best individual appears in the middle of the allowed generations; distributions appear normal unless slightly skewed to the higher values.

7.2.2.4. Individual Size.

Table 31 – fittable_fn05, individual size and depth Median.

Algorithm	Classic	Repulsors 08			Repulsors 09		
		1.01	1.5	2.5	1.01	1.5	2.5
Individual Tree Depth	10	4.5	8.5	6	4	5	6.5
Individual Size	76	9	24	18	8	9	21

In this dataset, the individual's sizes were well within limits allowed and in only one situation did the program block the growth of the trees (GP_CLASSIC). A noteworthy result was GP_REPULSORS_09 OFT 1.01; the individuals were rather small, this also appears to have affected the performance of the algorithm as the median result on the test set was rather poor.

7.2.2.5. fittable_fn05 Centrality Measures Graphical Comparison.

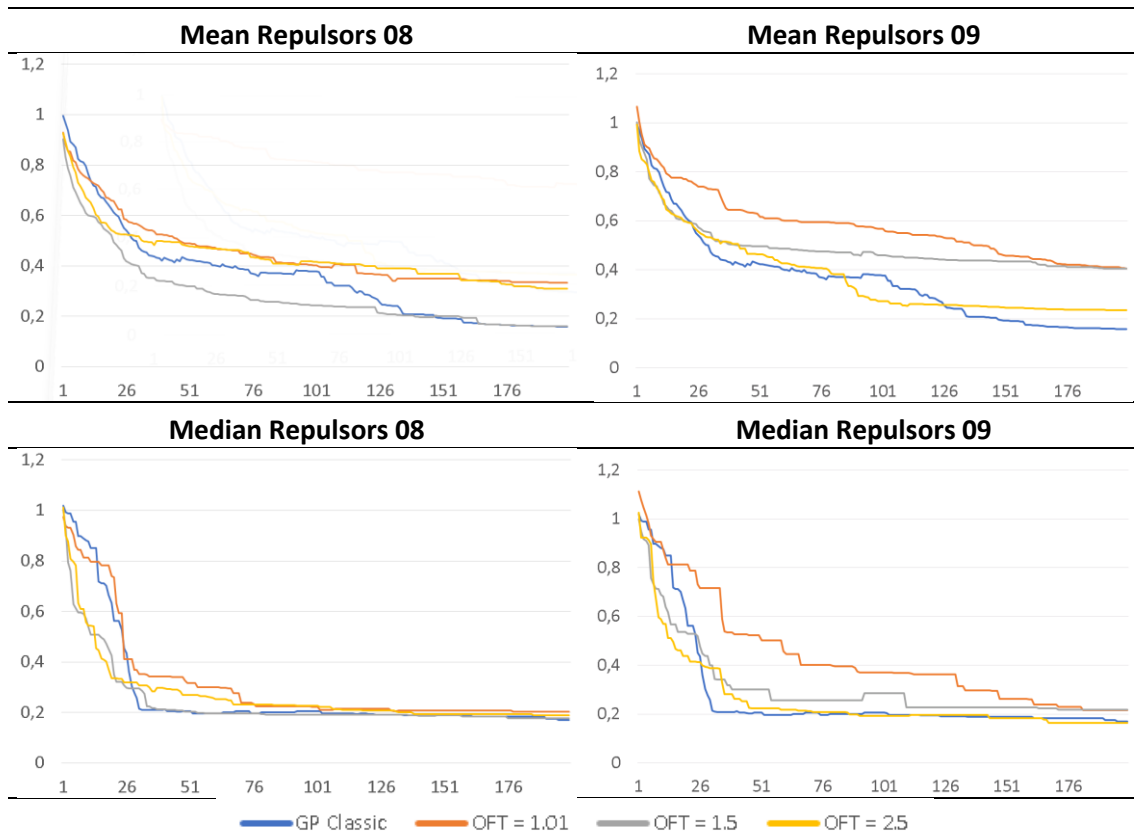


Figure 37: fittable_fn05, training set graphical centrality comparison.

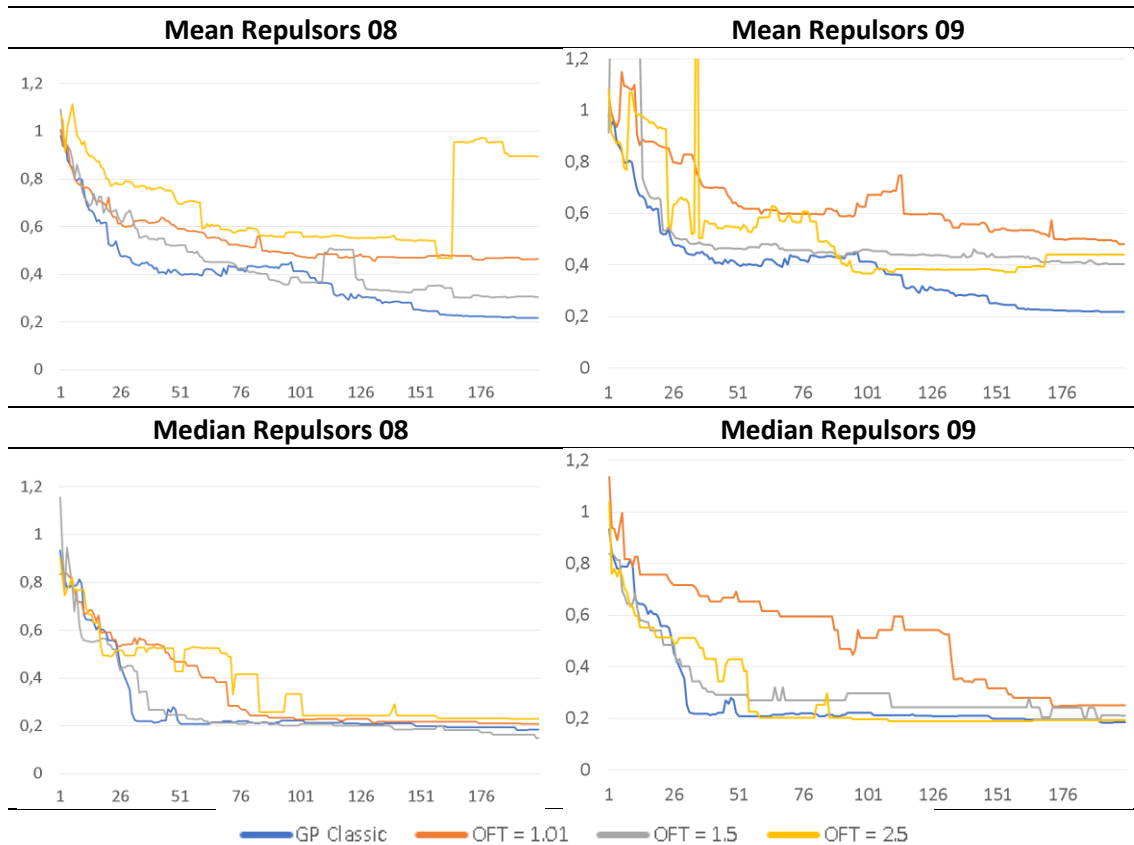


Figure 38: fittable_fn05, test set graphical centrality comparison.

7.2.2.6. Individual Conclusions for fittable_fn05.

This dataset shows a very fast learning rate, possibly due to the solution being quickly found by nearly all algorithms. The learning performance of GP_REPULSORS_09 OFT 1.01 lagged over the other methods but managed to catch up around the 175th iteration. The simplicity of the function being approached

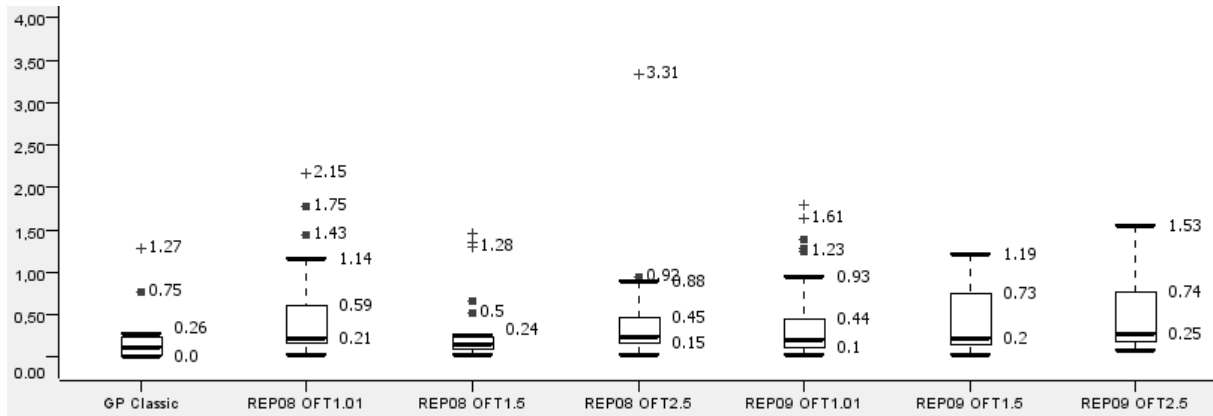


Figure 39: Boxplot of the best fitness on the fittable_fn05 test set.

Table 32 – Kruskal-Wallis Test for fittable_fn05.

Method	Train Pv	Test Pv
GP Repulsors 08	0,041	0,053
GP Repulsors 09	0,001	0,010

On the test set, both Algorithms have statistical significant median differences between them and the Classic Algorithm, GP_REPULSORS_09 is clearly at the 0.05 significance level, and GP_REPULSORS_08 is just 0.003 above it. On the training set, both algorithms are also below the 0.05 threshold.

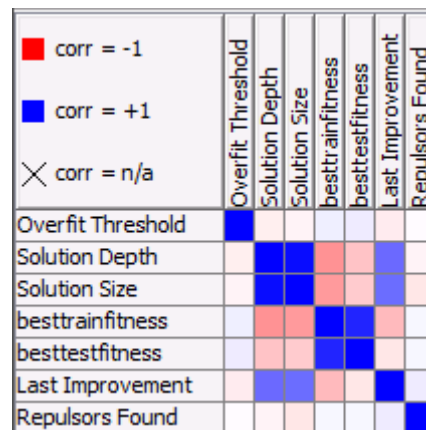


Figure 40: Correlation matrix for fittable_fn05 dataset.

There is a strong correlation between size and depth, training and test fitnesses also show a strong correlation; this may mean that the model trains and generalizes well, this is supported by the line plot of figures 33 and 34. There is a negative correlation between the sizes and fitnesses. It's interesting the negative correlation between the training fitness and the solution sizes.

7.2.3. Detailed Analysis of the Fittable_fn07 dataset

The fittable_fn07 is a dataset created using the function:

$$\text{Equation 14} \quad f(x) = x_0 \cdot x_1 \cdot x_2 + \frac{x_3}{x_4} \cdot x_5 \cdot x_6 \cdot x_7 \cdot x_8 - x_9 - \frac{x_{10}}{x_{11}} \cdot x_{12} \cdot x_{13} \cdot x_{14} + x_{15}$$

There are 100 samples in the dataset, the variables $x_0, x_1, \dots, x_{13}$ have random values in the interval $[-1,1]$. This dataset was created to test the ability of the program to handle multiple variables effectively.

7.2.3.1. Summary statistics.

Table 33 – fittable_fn07 Summary statistics for the Training set.

Algorithm	O.F.T.	Min.	Max.	Mean	Median	S.D.	Var.
GP_CLASSIC	N/A	0.241	2.100	0.803	0.546	0.530	0.281
GP_REPULSORS_08	1.01	0.386	2.245	0.755	0.684	0.409	0.167
GP_REPULSORS_08	1.5	0.298	2.088	0.921	0.549	0.573	0.329
GP_REPULSORS_08	2.5	0.236	2.315	0.834	0.631	0.528	0.278
GP_REPULSORS_09	1.01	0.410	1.987	0.861	0.685	0.452	0.204
GP_REPULSORS_09	1.5	0.464	2.203	1.079	0.831	0.520	0.271
GP_REPULSORS_09	2.5	0.325	1.978	0.839	0.556	0.580	0.337

Table 34 – fittable_fn07 Summary statistics for the Test set.

Algorithm	O.F.T.	Min.	Max.	Mean	Median	S.D.	Var.
GP_CLASSIC	N/A	0.490	11.084	1.706	1.196	1.906	3.633
GP_REPULSORS_08	1.01	0.588	3.707	1.569	1.263	0.864	0.746
GP_REPULSORS_08	1.5	0.615	4.456	1.762	1.619	0.945	0.894
GP_REPULSORS_08	2.5	0.688	4.613	1.808	1.546	0.968	0.937
GP_REPULSORS_09	1.01	0.431	3.139	1.417	1.227	0.751	0.564
GP_REPULSORS_09	1.5	0.607	4.942	1.988	1.652	1.143	1.306
GP_REPULSORS_09	2.5	0.493	32.091	2.726	1.388	5.647	31.890

The best absolute and mean training values were obtained using REPULSORS_08, GP_CLASSIC achieved the best median.

On the test set, the best absolute and mean were found with REPULSORS_098 with OFT 1.01, but once again the best median was obtained using GP_CLASSIC

7.2.3.2. Repulsors.

Table 35 – fittable_fn07, median number of repulsors found.

Algorithm	O.F.T.		
	1.01	1.5	2.5
GP_REPULSORS_08	89	4.5	0
GP_REPULSORS_09	93.5	8.5	5

The number of repulsors diminishes as the overfit threshold increases.

7.2.3.3. Best Individual Generation.

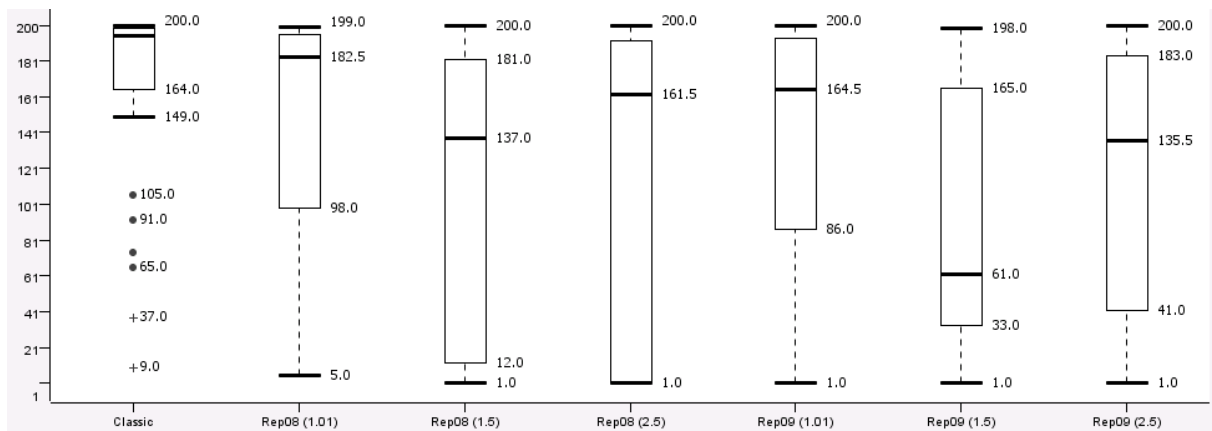


Figure 41:fittable_fn07, Best Individual Generation.

The best solutions were found after the 100th generation, but the distributions were much wider in shape (higher SD). Best Medians, using GP CLASSIC were found at high generations.

7.2.3.4. Individual Size.

Table 36 – fittable_fn07, Individual size and depth Median.

Algorithm	Classic	Repulsors 08			Repulsors 09		
		1.01	1.5	2.5	1.01	1.5	2.5
Individual Tree Depth	10	8.5	9.5	10	6.5	3.5	10
Individual Size	51	28	24	38	23	8	74

The tree depth was high but did not reach the limit in some cases. Nevertheless, if we look at figure 42, we may conclude that in some scenarios the algorithm could have used some more training.

7.2.3.5. fittable_fn07 Centrality Measures Graphical Comparison.

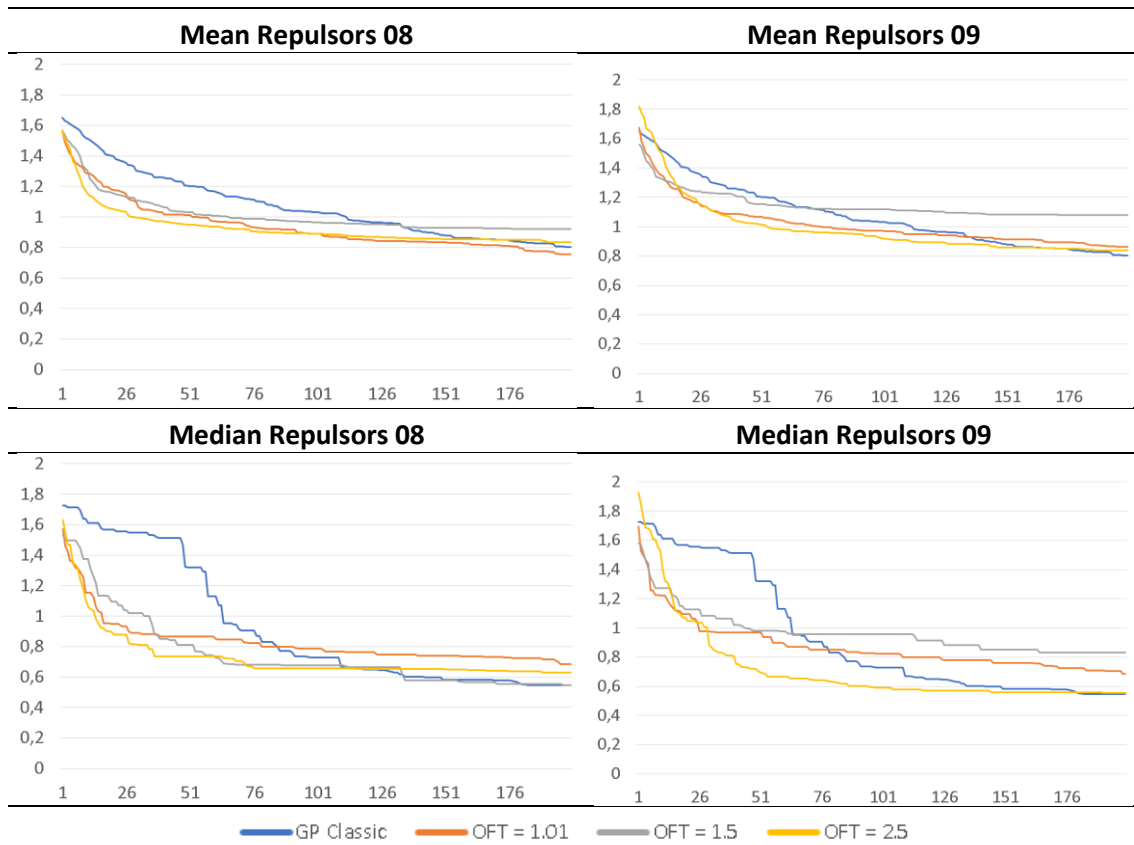


Figure 42: fittable_fn07, training set graphical centrality comparison.

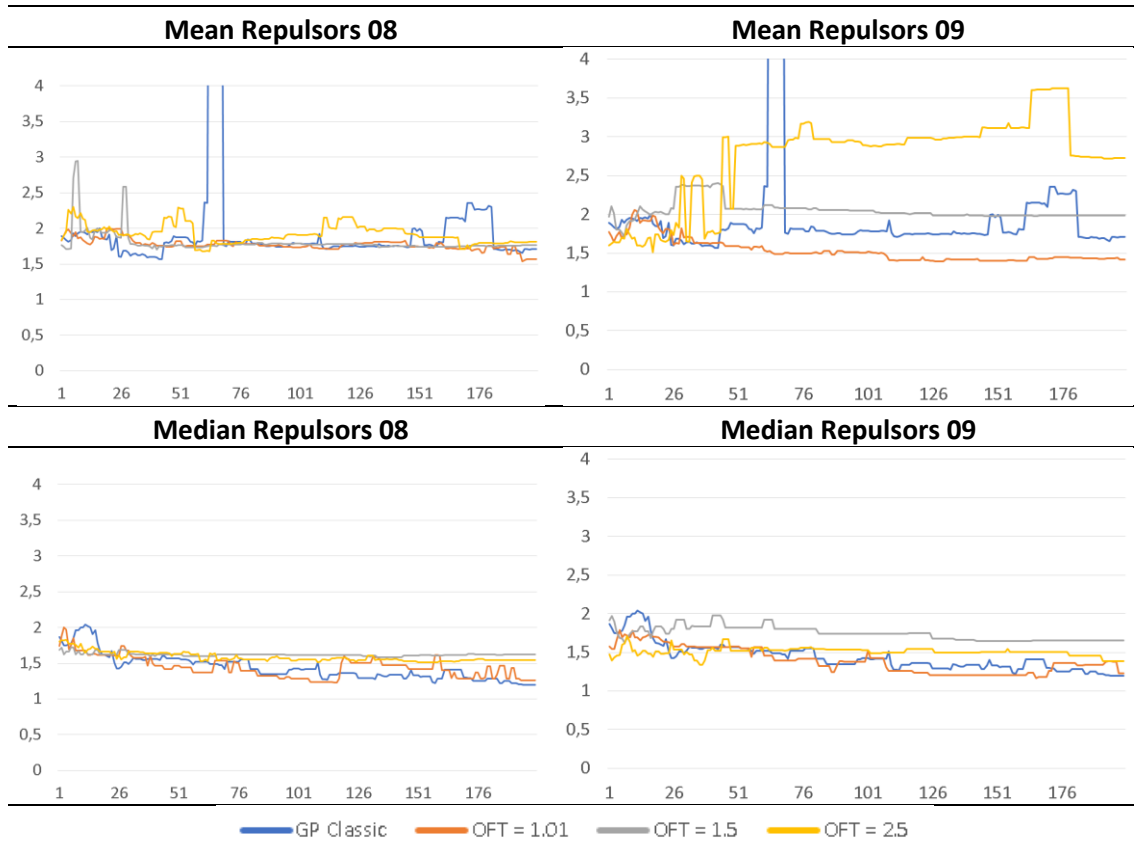


Figure 43: fittable_fn07, test set graphical centrality comparison.

Taking into consideration the plots in figure 42, it is very evident that the repulsor based algorithms learn much faster, in the median plots it takes standard GP around 50 more generation to reach a comparable result.

7.2.3.6. Individual Conclusions for fittable_fn07.

Figure 44 shows that the results are very close to each other, independently on the used algorithm.

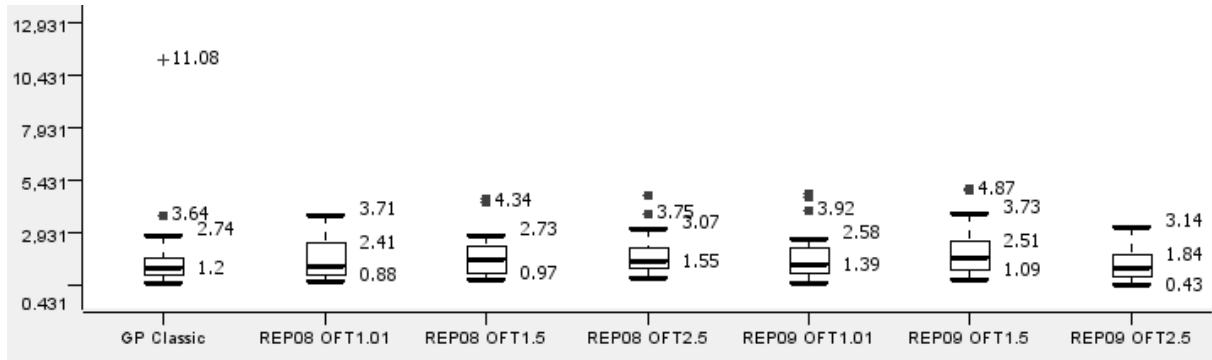


Figure 44: Boxplot of the best fitness on the fittable_fn07 test set.

Table 37 – Kruskal-Wallis Test for fittable_fn07.

Method	Train Pv	Test Pv
GP Repulsors 08	0.973	0.690
GP Repulsors 09	0.068	0.442

Using the Kruskal-Wallis test result, there doesn't seem to be relevant statistical difference between the medians on either set; this value supports the similarity in the test set curves in the graphical analysis and the box plot.



Figure 45: Correlation matrix for fittable_fn07 dataset.

There appears to be a weak correlation between the test and train fitnesses; this is supported by the line plot in Figures 38 and 39. Strong negative correlation between training fitness and individual depth and size.

7.2.4. Detailed Analysis of the Fittable_fn08 dataset

The fittable_fn08 is a dataset created using the function:

$$\text{Equation 15} \quad f(x) = x_0 \cdot x_1 + \sin((x_0 - 1) - (x_1 - 1))$$

There are 100 samples in the dataset, the variables x_0, x_1 have random values in the interval $[-1, 1]$. This is a crescent function with a small periodic variability.

7.2.4.1. Summary statistics.

Table 38 – fittable_fn08 summary statistics for the Training set.

Algorithm	O.F.T.	Min.	Max.	Mean	Median	S.D.	Var.
GP_CLASSIC	N/A	0.553	0.692	0.632	0.647	0.046	0.002
GP_REPULSORS_08	1.01	0.551	0.703	0.627	0.631	0.044	0.002
GP_REPULSORS_08	1.5	0.531	0.709	0.619	0.609	0.054	0.003
GP_REPULSORS_08	2.5	0.499	0.721	0.634	0.653	0.059	0.003
GP_REPULSORS_09	1.01	0.544	0.715	0.623	0.610	0.059	0.003
GP_REPULSORS_09	1.5	0.527	0.672	0.600	0.601	0.037	0.001
GP_REPULSORS_09	2.5	0.492	0.703	0.599	0.604	0.058	0.003

Table 39 – fittable_fn08 summary statistics for the Test set.

Algorithm	O.F.T.	Min.	Max.	Mean	Median	S.D.	Var.
GP_CLASSIC	N/A	0.557	2.414	0.723	0.671	0.324	0.105
GP_REPULSORS_08	1.01	0.567	3.931	0.843	0.721	0.595	0.354
GP_REPULSORS_08	1.5	0.589	7.695	1.091	0.703	1.439	2.069
GP_REPULSORS_08	2.5	0.562	1.612	0.695	0.662	0.185	0.034
GP_REPULSORS_09	1.01	0.597	3.234	0.880	0.683	0.588	0.346
GP_REPULSORS_09	1.5	0.538	3.378	0.973	0.707	0.691	0.478
GP_REPULSORS_09	2.5	0.567	4.488	0.932	0.715	0.709	0.502

In this function, the repulsor based algorithms outperformed GP_CLASSIC in all measures.

In the training set, the best performer was GP_REPULSORS_09, but, in the test set, the best values were obtained with GP_REPULSORS_08. Differences in performance were small.

7.2.4.2. Repulsors.

Table 40 – fittable_fn08, median number of repulsors found.

Algorithm	O.F.T.		
	1.01	1.5	2.5
GP_REPULSORS_08	14.5	0	0
GP_REPULSORS_09	8.5	1	0

The number of repulsors diminishes as the overfit threshold increases. No repulsors found above OFT 1.5.

7.2.4.1. Best Individual Generation.

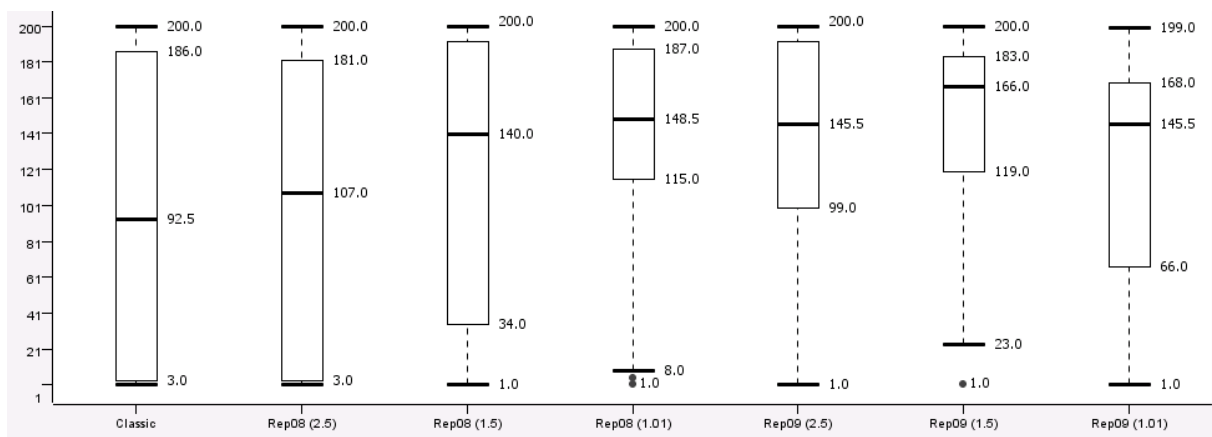


Figure 46:fittable_fn08, Best Individual Generation.

The best solutions were obtained around the middle of the allowed generations (100th generation), the distribution curve is wide in most algorithms.

7.2.4.2. Individual Size.

Table 41 – fittable_fn08, Individual size and depth Median.

Algorithm	Classic	Repulsors 08			Repulsors 09		
	N.A.	1.01	1.5	2.5	1.01	1.5	2.5
Individual Tree Depth	8.5	10	10	9	10	10	10
Individual Size	59	97	88	77	83	111	101

Most of the algorithms reached their maximum allowed size, however in this dataset, looking at figure 47, the results appear to have stabilized, and it appears that little gain can be obtained by running it for more generations.

7.2.4.3. fittable_fn08, Centrality Measures Graphical Comparison.

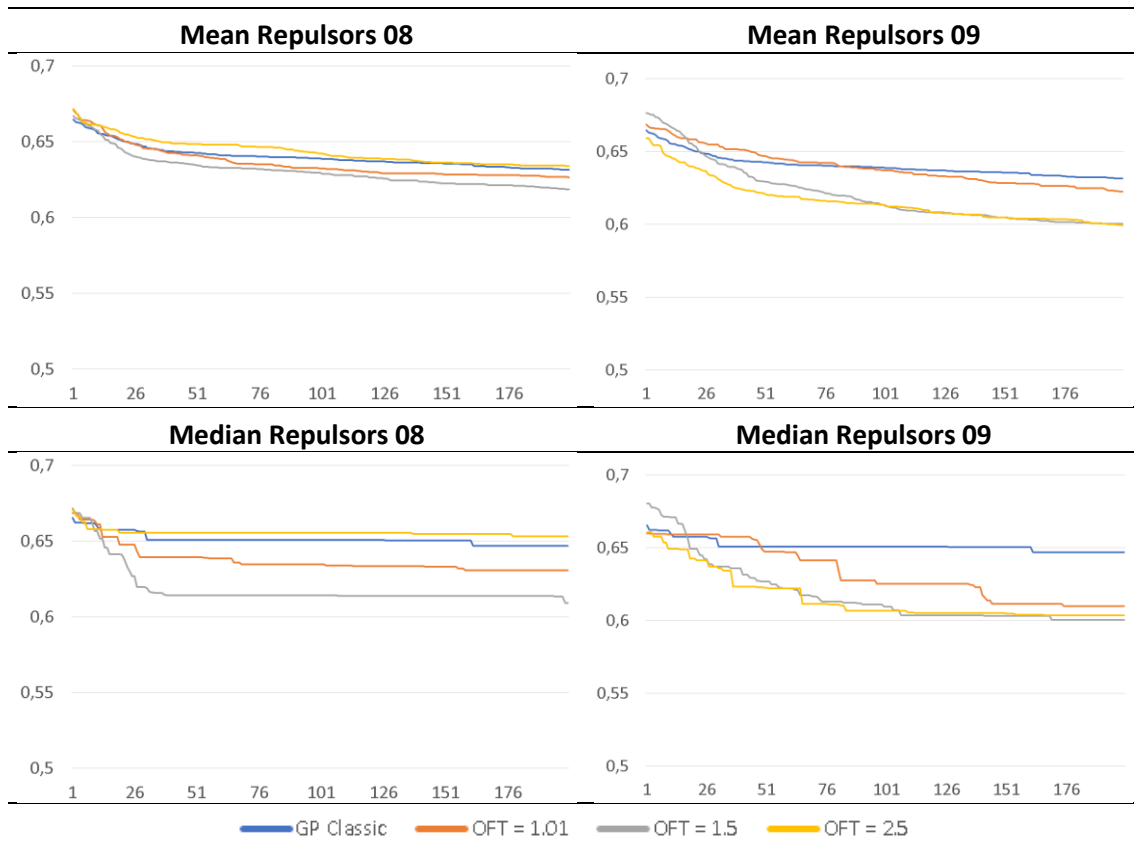


Figure 47: fittable_fn08, training set graphical centrality comparison.

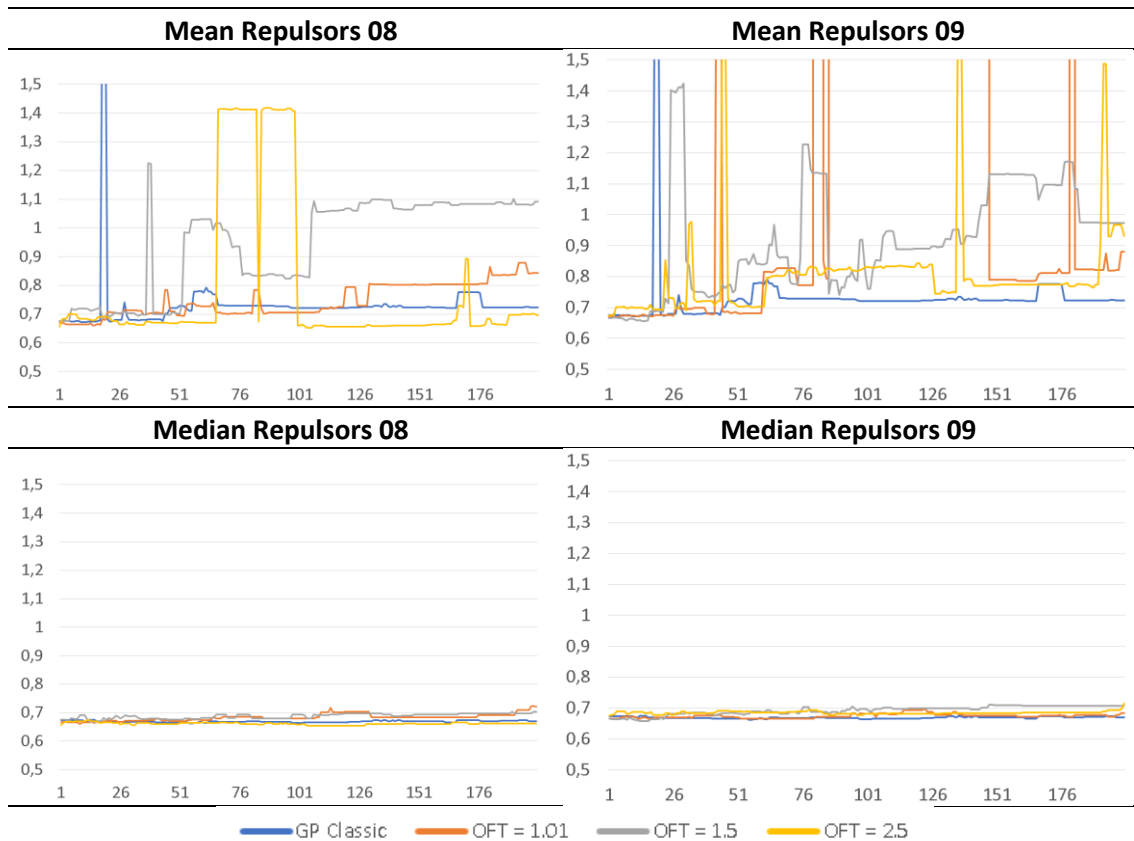


Figure 48: fittable_fn08, test set graphical centrality comparison.

The mean of both GP_REPULSORS algorithms present extreme overfitting on the mean when compared with the CLASSIC_GP; this does not happen in the median value.

7.2.4.1. Individual Conclusions for fittable_fn08.

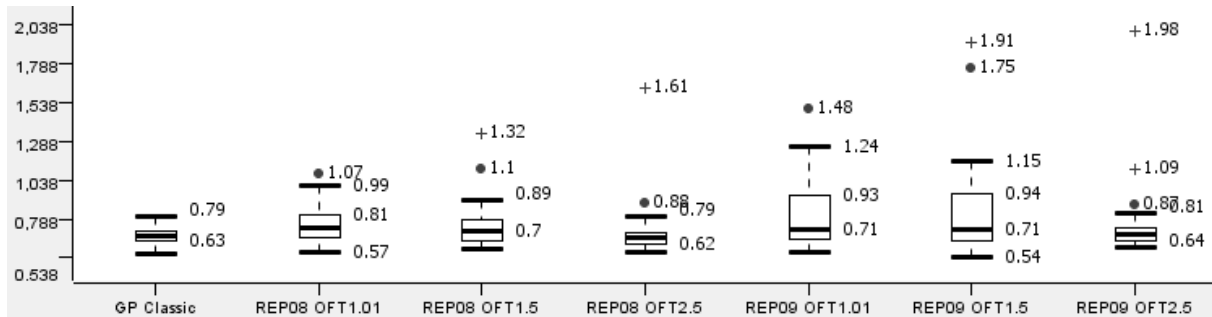


Figure 49: Boxplot of the best fitness on the fittable_fn08 test set.

The solutions show no major difference; this is supported by the Kruskal-Wallis test, that reports no statistical difference among the tested sets.

Table 42 – Kruskal-Wallis Test for fittable_fn08.

Method	Train Pv	Test Pv
GP Repulsors 08	0.949	0.116
GP Repulsors 09	0.287	0.391

The median values are similar as supported by the line charts of the previous page and the Kruskal-Wallis test.

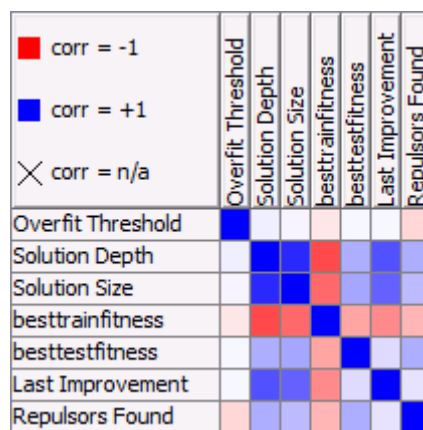


Figure 50: Correlation matrix for fittable_fn08 dataset.

Strong Correlation between sizes and depth indicate the fitter the individual is the bigger it also is.

7.2.5. Detailed Analysis of the CCPP Folds dataset.

Combined Cycle Power Plant data set (CCPP) summary statistics. Is a real life dataset.

‘The dataset contains 9568 data points collected from a Combined Cycle Power Plant over six years (2006-2011) when the power plant was set to work with full load.’

Further information can be found on the appendixes.

Table 43 – CCPP folds, Summary statistics for the Training set.

Algorithm	O.F.T.	Min.	Max.	Mean	Median	S.D.	Var.
GP_CLASSIC	N/A	4.296	23.752	9.536	8.943	4.465	19.936
GP_REPULSORS_08	1.01	4.297	26.891	10.976	8.447	6.687	44.722
GP_REPULSORS_08	1.5	4.442	33.961	11.328	9.815	7.300	53.292
GP_REPULSORS_08	2.5	4.096	36.256	11.404	11.631	5.701	32.502
GP_REPULSORS_09	1.01	5.000	37.576	12.094	7.205	9.119	83.147
GP_REPULSORS_09	1.5	4.155	54.676	14.046	11.829	10.134	102.692
GP_REPULSORS_09	2.5	4.069	32.026	8.753	7.609	5.600	31.360

Table 44 – CCPP folds, Summary statistics for the Test set.

Algorithm	O.F.T.	Min.	Max.	Mean	Median	S.D.	Var.
GP_CLASSIC	N/A	5.026	156.798	16.873	11.480	27.317	746.210
GP_REPULSORS_08	1.01	4.858	31.030	11.275	8.314	6.491	42.138
GP_REPULSORS_08	1.5	4.425	84.909	16.796	10.545	17.541	307.702
GP_REPULSORS_08	2.5	5.097	296.913	26.610	13.538	53.304	2841.362
GP_REPULSORS_09	1.01	4.613	55.744	13.188	7.975	10.989	120.752
GP_REPULSORS_09	1.5	3.988	753.209	42.081	13.347	134.881	18192.920
GP_REPULSORS_09	2.5	4.864	346.493	21.695	8.909	61.643	3799.888

Best training performance was obtained using GP_REPULSORS_09 with OFT 2.5. in the Test set the best median found was using GP_REPULSORS_09 and OFT 1.01.

GP_REPULSORS_09 OFT 1.5 show a big variance when compared to the other algorithm/parameter combinations.

7.2.5.1. Repulsors.

Table 45 – CCPP folds, Median number of repulsors found.

Algorithm	O.F.T.		
	1.01	1.5	2.5
GP_REPULSORS_08	60	0	0
GP_REPULSORS_09	83.5	1.5	0

The number of repulsors diminishes as the overfit threshold increases. Almost no repulsors found above OFT 1.5.

7.2.5.1. Best Individual Generation.

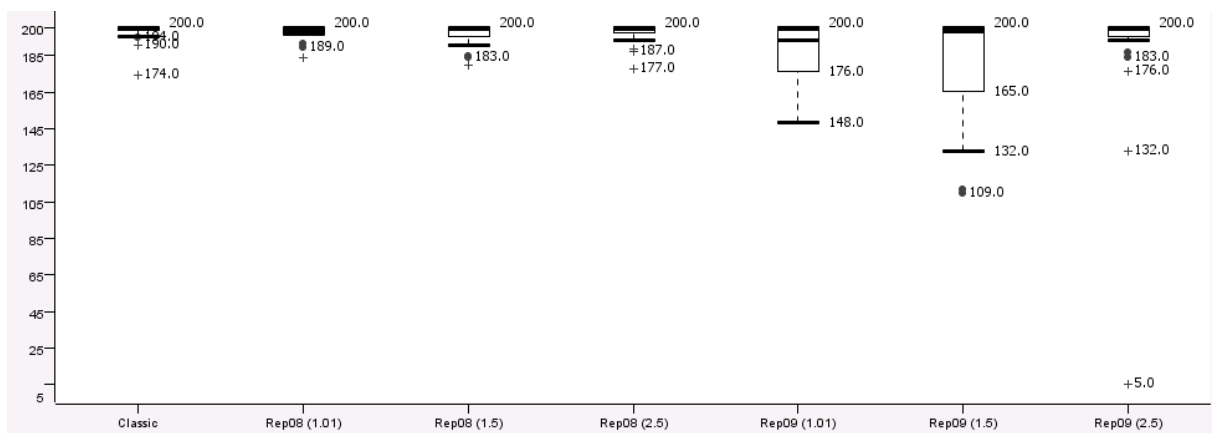


Figure 51: CCPP folds, Best Individual Generation.

The best individuals were found in the last generations allowed; this means that the algorithms could be further trained in this dataset and would probably achieve better results.

7.2.5.2. Individual Size.

Table 46 – CCPP folds, Individual size and depth Median.

Algorithm	Classic	Repulsors 08			Repulsors 09		
Measure / O.F.T.	N.A.	1.01	1.5	2.5	1.01	1.5	2.5
Individual Tree Depth	10	10	10	10	10	10	10
Individual Size	178	141	191	206	121	186	195

Evidently, all algorithms had their sizes limited by the algorithm. However looking at figure 52 the training appears to be complete or at least there is little to gain by continuing the algorithm for more generations.

7.2.5.3. CCPP folds, Centrality Measures Graphical Comparison.

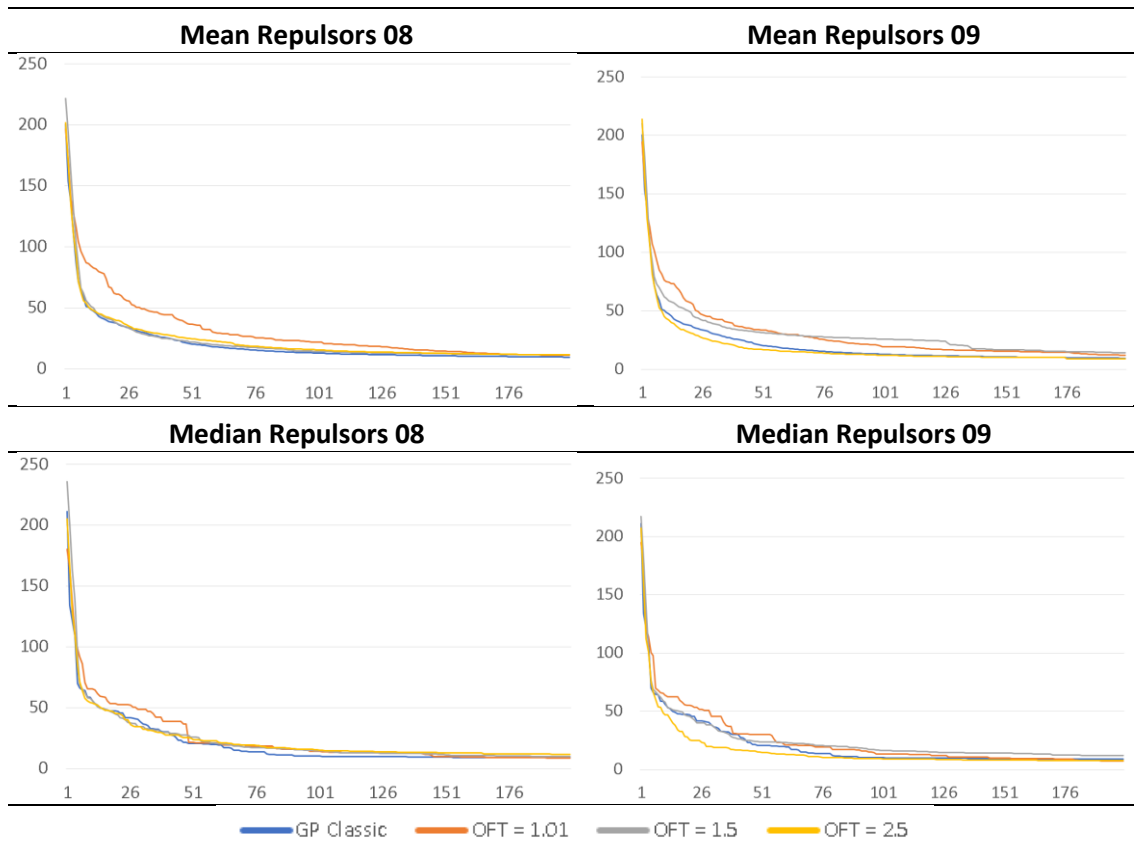


Figure 52: CCPP Folds, training set graphical centrality comparison.

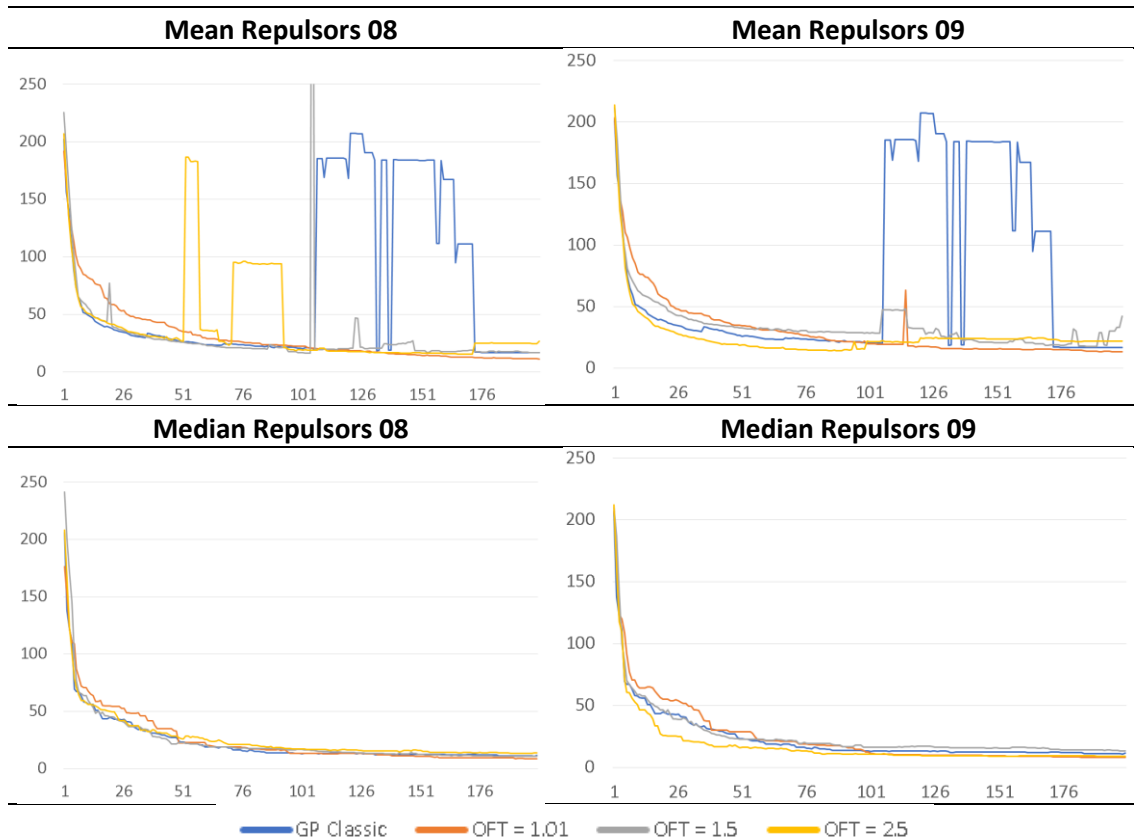


Figure 53: CCPP Folds, test set graphical centrality comparison.

7.2.5.4. Individual Conclusions for CCPP folds dataset.

All algorithms present a similar training behavior with the GP_REPULSORS_08 algorithm and show little variance with GP_REPULSORS_09. On the test set, there is an interval of generations where GP_CLASSIC presents a very poor result.

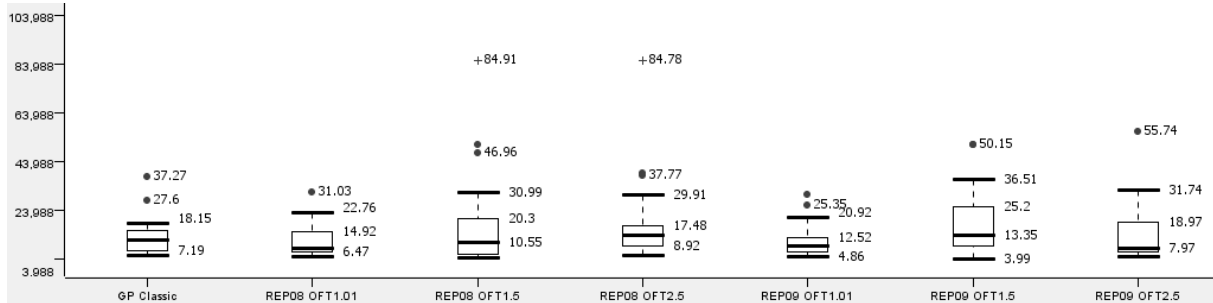


Figure 54: Boxplot of the best fitness on the CCPP folds test set.

Table 47 – Kruskal-Wallis Test on ccpp folds test data set.

Method	Train Pv	Test Pv
GP Repulsors 08	0.895	0.544
GP Repulsors 09	0.174	0.239

The Kruskal Wallis test shows no significant statistical difference between the algorithm's results; this is also evident in the box plot of figure 50 and line plots of figures 48 and 49.

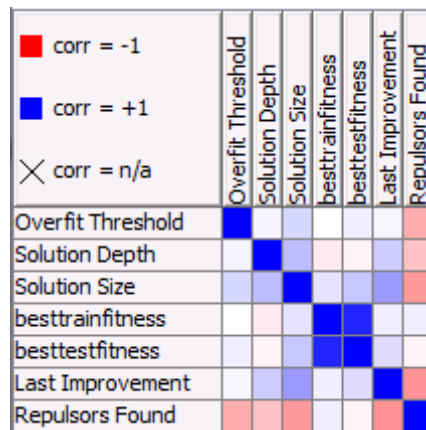


Figure 55: Correlation matrix for CCPP folds dataset.

Strong correlation between the training and test fitness indicates that the algorithm performed a good learning phase. The number of repulsors found has an inverse correlation with the solutions size.

7.2.6. Detailed Analysis of the Concrete Slump Folds dataset.

The concrete slump dataset is a real-life dataset. The data set includes 103 data points. There are 7 input variables, and 3 output variables in the data set. More information is available on the appendixes.

7.2.6.1. Summary statistics.

Table 48 – concrete slump, Summary statistics for the Training set.

Algorithm	O.F.T.	Min.	Max.	Mean	Median	S.D.	Var.
GP_CLASSIC	N/A	2.392	8.070	3.902	3.552	1.261	1.590
GP_REPULSORS_08	1.01	3.317	11.645	6.851	6.349	2.107	4.438
GP_REPULSORS_08	1.5	2.273	10.743	4.911	4.794	1.821	3.316
GP_REPULSORS_08	2.5	2.222	12.627	4.075	3.384	2.047	4.189
GP_REPULSORS_09	1.01	3.793	10.760	6.363	6.062	1.685	2.838
GP_REPULSORS_09	1.5	1.833	11.108	5.646	4.737	2.356	5.551
GP_REPULSORS_09	2.5	2.136	8.233	4.382	3.951	1.578	2.490

Table 49 – concrete slump, Summary statistics for the Test set.

Algorithm	O.F.T.	Min.	Max.	Mean	Median	S.D.	Var.
GP_CLASSIC	N/A	2.679	92.355	8.482	4.365	16.184	261.922
GP_REPULSORS_08	1.01	3.636	165.416	14.651	7.731	29.514	871.069
GP_REPULSORS_08	1.5	3.559	19.012	6.994	6.029	3.552	12.620
GP_REPULSORS_08	2.5	2.918	146.770	12.662	5.475	27.529	757.860
GP_REPULSORS_09	1.01	4.644	21.587	7.638	7.001	3.222	10.380
GP_REPULSORS_09	1.5	3.145	70.812	11.016	8.228	12.625	159.383
GP_REPULSORS_09	2.5	2.281	18.959	6.350	5.817	3.374	11.381

In the training set, the best median value was obtained by GP_REPULSORS_08. The test set best median was found with GP_CLASSIC.

Minimum value was found using Repulsors 09 Algorithm.

7.2.6.2. Repulsors.

Table 50 – concrete slump, Median number of repulsors found.

Algorithm	O.F.T.		
	1.01	1.5	2.5
GP_REPULSORS_08	63	3	0
GP_REPULSORS_09	78.5	3	0

The number of repulsors diminishes as the overfit threshold increases. At OFT 2.5 the number of repulsors is null.

7.2.6.1. Best Individual Generation.

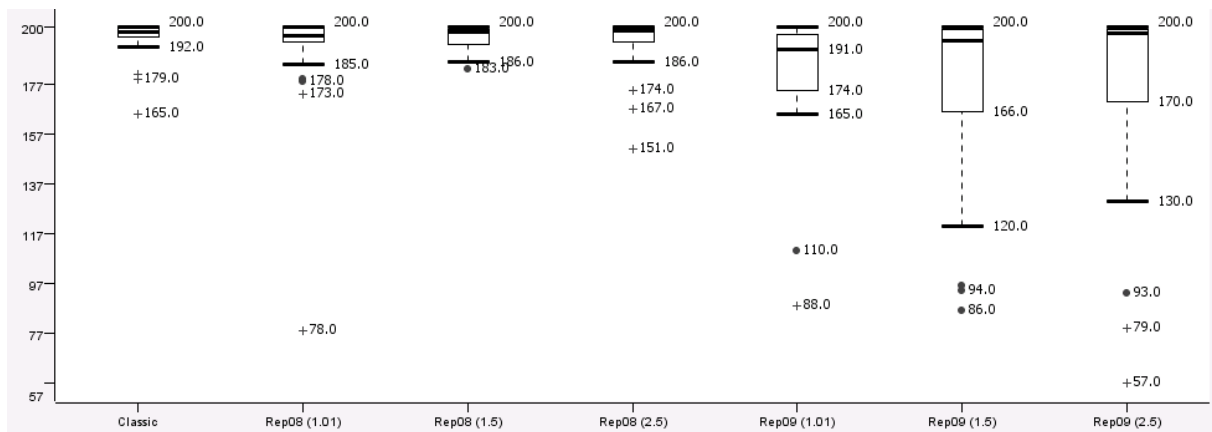


Figure 56: Concrete Slump, Best Individual Generation.

The best solutions were found in higher generations; this may indicate that the model could use some more training, the plots confirm this as they do not flatten in the end.

7.2.6.2. Individual Size.

Table 51 – concrete slump, Individual size and depth Median.

Algorithm	Classic	Repulsors 08			Repulsors 09		
Measure / O.F.T.	N.A.	1.01	1.5	2.5	1.01	1.5	2.5
Individual Tree Depth	10	10	10	10	10	10	10
Individual Size	134	107	145	142	141	165	148

Individuals tended to grow to the maximum allowed depth; the training should have been continued, this conclusion is supported by figure's 57 plot and by figure 56 that shows us that the best solutions are found very late on the run.

7.2.6.3. concrete slump, Centrality Measures Graphical Comparison.

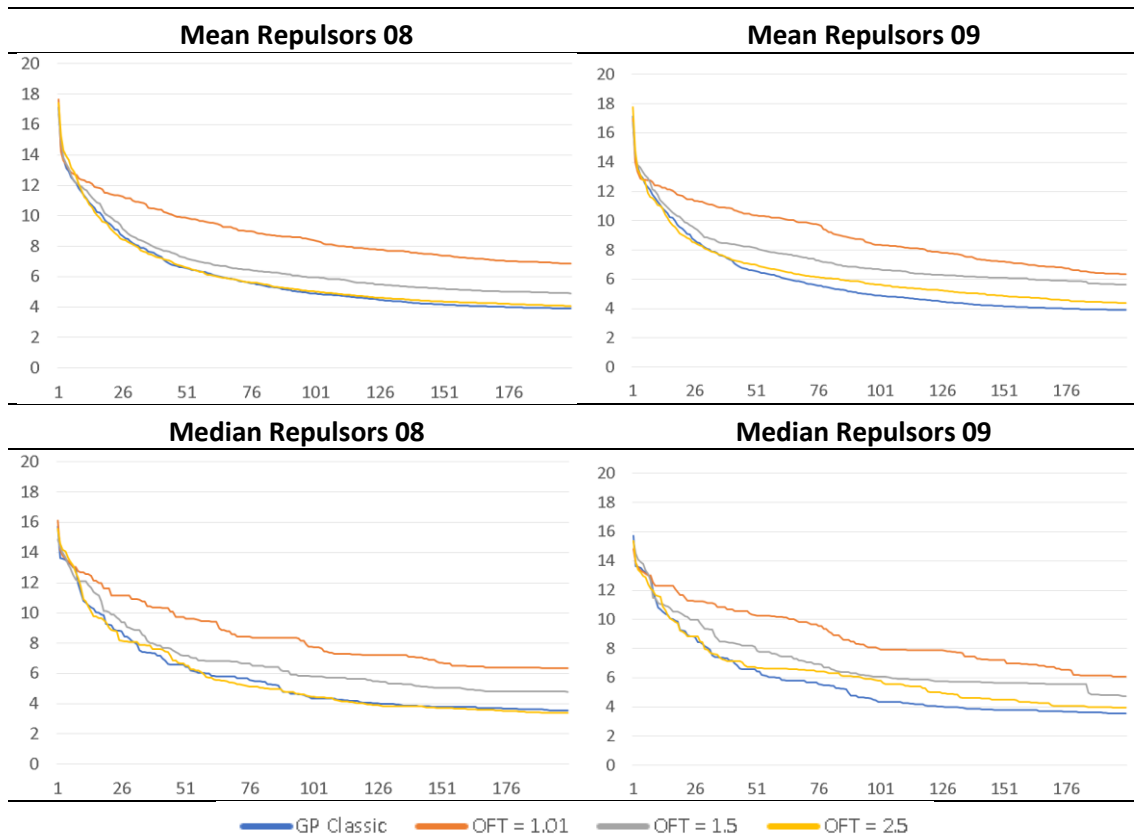


Figure 57: concrete slump, training set graphical centrality comparison.

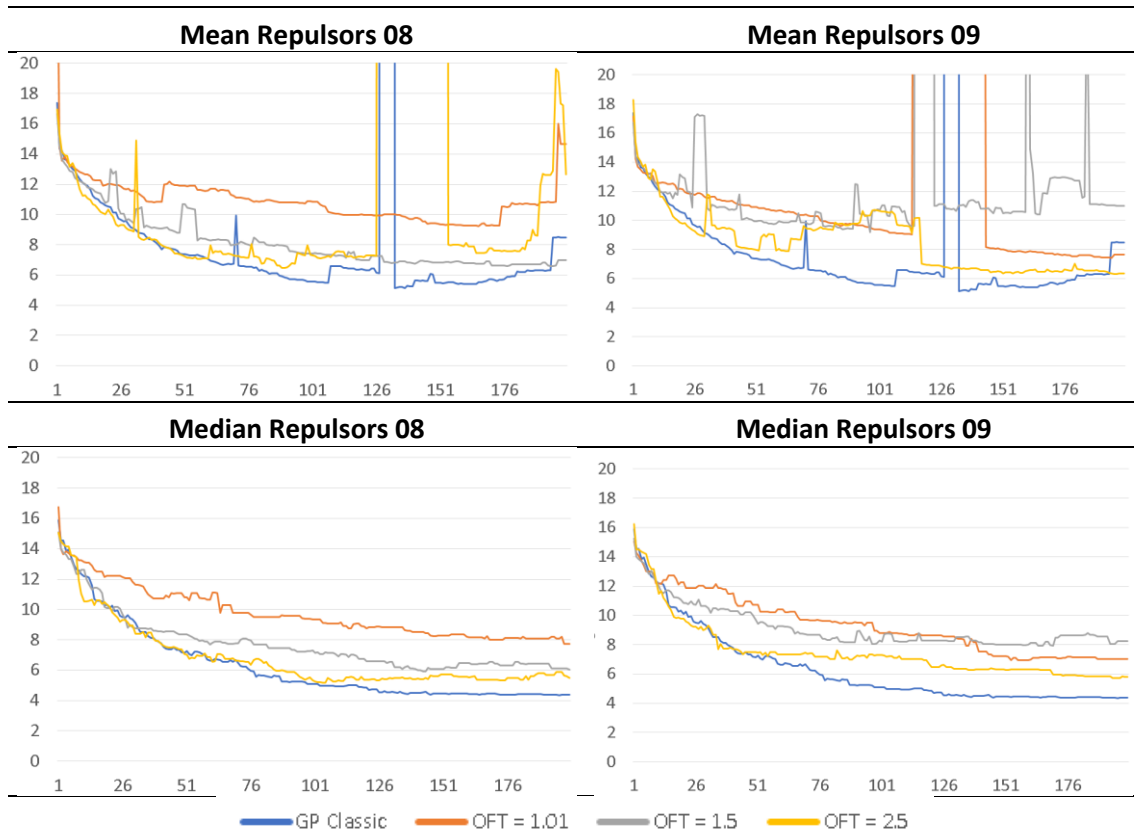


Figure 58: concrete slump, test set graphical centrality comparison.

7.2.6.4. Individual Conclusions for Concrete Slump Dataset.

In both algorithms, the Classic method appears to have a more predictable behavior.

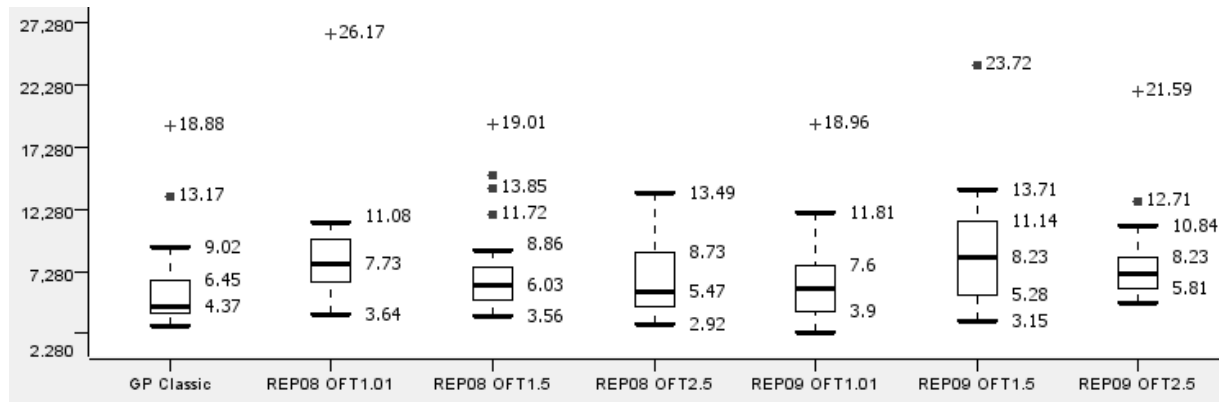


Figure 59: Boxplot of the best fitness on the Concrete Slump test set.

The medians are different; this indicates that there is a difference between methods, this is confirmed by the Kruskal-Wallis test.

Table 52 – Kruskal-Wallis Test for the concrete slump dataset.

Method	Train Pv	Test Pv
GP Repulsors 08	4.23E-07	0.007
GP Repulsors 09	7.48E-06	0.007

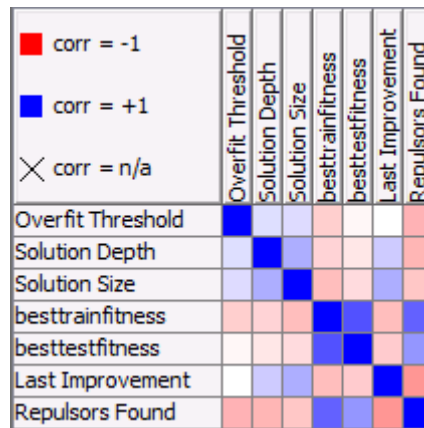


Figure 60: Correlation matrix for the concrete slump dataset.

Training and test fitnesses are correlated, indicating successful training and generalization. The number of repulsors found is correlated with the training and test fitness.

7.2.7. Detailed Analysis of the Concrete compressive strength dataset.

The concrete compressive strength (Concrete CS) dataset is a real-life dataset. A number of instances 1030, Number of Attributes 9 Attribute breakdown 8 quantitative input variables, and 1 quantitative output variable. More information is available on the appendixes.

7.2.7.1. Summary statistics.

Table 53 – Concrete CS, Summary statistics for the Training set.

Algorithm	O.F.T.	Min.	Max.	Mean	Median	S.D.	Var.
GP_CLASSIC	N/A	6.960	22.217	10.588	10.392	2.755	7.590
GP_REPULSORS_08	1.01	8.679	15.835	12.074	11.870	1.609	2.589
GP_REPULSORS_08	1.5	7.421	20.280	9.901	9.082	2.516	6.332
GP_REPULSORS_08	2.5	7.886	12.423	9.479	9.137	1.235	1.525
GP_REPULSORS_09	1.01	8.458	18.236	12.360	12.122	2.120	4.494
GP_REPULSORS_09	1.5	6.645	20.579	9.766	8.734	3.061	9.371
GP_REPULSORS_09	2.5	7.157	21.104	9.819	8.960	2.672	7.139

Table 54 – Concrete CS, Summary statistics for the Test set.

Algorithm	O.F.T.	Min.	Max.	Mean	Median	S.D.	Var.
GP_CLASSIC	N/A	7.880	78.411	13.231	10.739	12.569	157.986
GP_REPULSORS_08	1.01	9.072	15.305	12.606	12.671	1.529	2.337
GP_REPULSORS_08	1.5	7.659	102.485	13.311	9.383	17.066	291.241
GP_REPULSORS_08	2.5	7.288	21.469	10.237	9.922	2.637	6.954
GP_REPULSORS_09	1.01	8.723	51.289	14.189	12.333	7.563	57.201
GP_REPULSORS_09	1.5	6.923	22.264	10.340	9.295	3.277	10.736
GP_REPULSORS_09	2.5	7.522	21.262	10.663	9.804	3.037	9.223

In the training set, the best median was found using GP_REPULSORS_08 with OFT 1.5. The test set's best median was found using GP_REPULSORS_09 with OFT 1.5

7.2.7.2. Repulsors.

Table 55 – Concrete CS, Median number of repulsors found.

Algorithm	O.F.T.		
	1.01	1.5	2.5
GP_REPULSORS_08	51.5	0	0
GP_REPULSORS_09	75	0	0

The number of repulsors diminishes as the overfit threshold increases. Virtually no repulsors were found above OFT 1.01

7.2.7.3. Best Individual Generation.

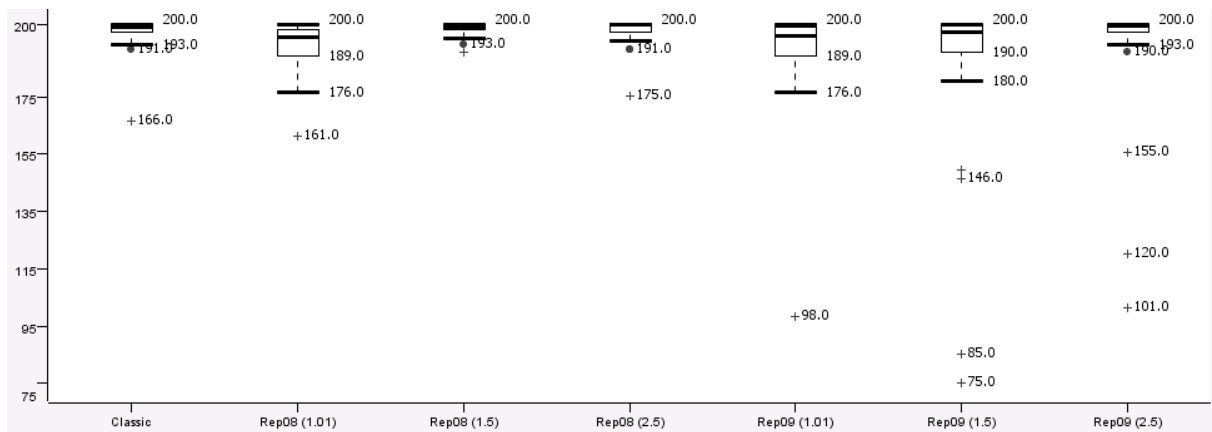


Figure 61: Concrete CS, Best Individual Generation.

The Best solutions were found late in the runs. This dataset could have used more training.

7.2.7.4. Individual Size.

Table 56 – Individual size and depth Median.

Algorithm	Classic	Repulsors 08			Repulsors 09		
		1.01	1.5	2.5	1.01	1.5	2.5
Measure / O.F.T.	N.A.						
Tree Depth	10	10	10	10	10	10	10
Tree Size	152	95	156	185	117	170	182

Sizes seem to be limited by the algorithm settings. Both figures 61 and 62 support the need for more training as the best solutions were found at late generations and the plot does not seem to have reached a flat.

7.2.7.5. Concrete CS, Centrality Measures Graphical.

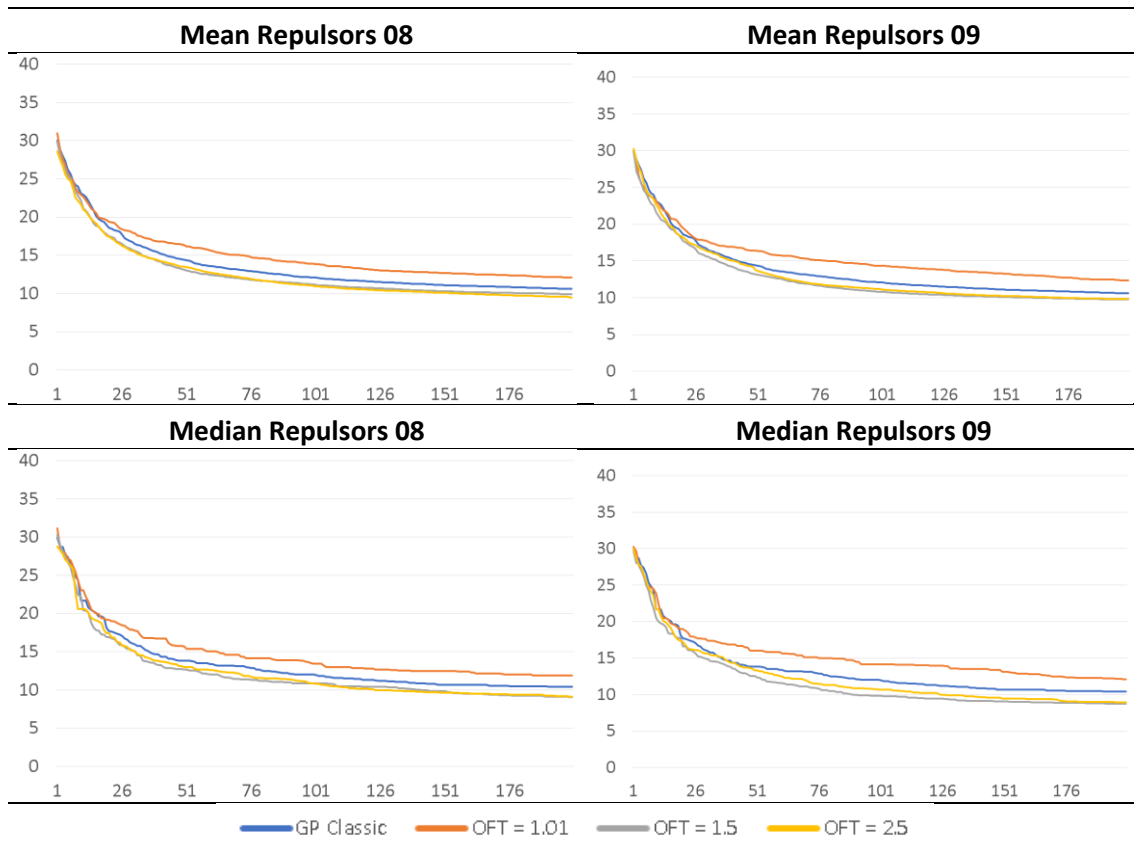


Figure 62:fittable_sin, training set graphical centrality comparison.

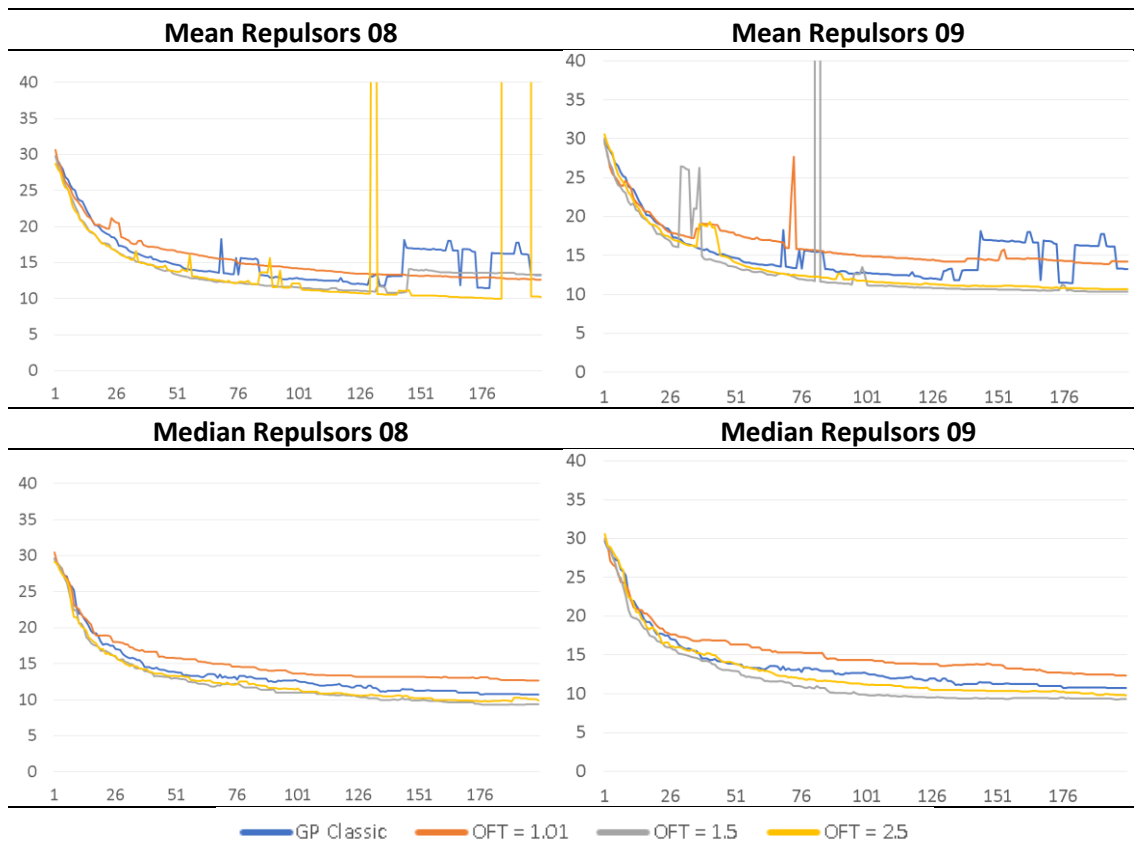


Figure 63: fittable_sin, test set graphical centrality comparison.

7.2.7.6. Individual Conclusions for Concrete Dataset.

This dataset presents no significant overfitting.

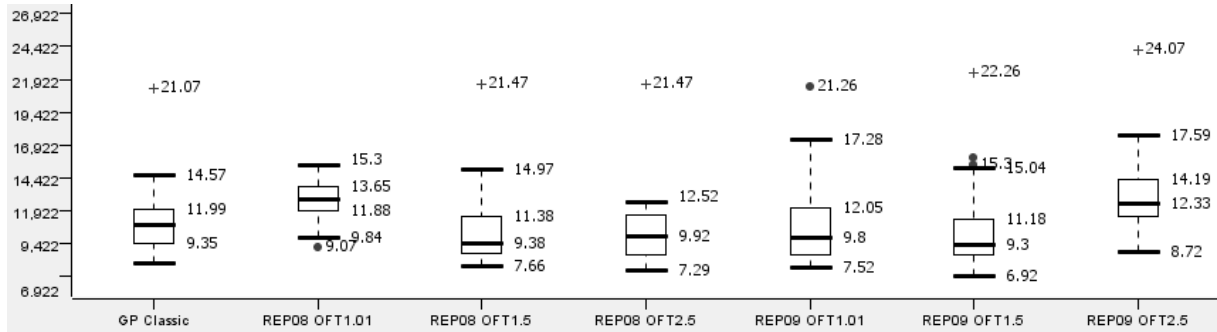


Figure 64: Boxplot of the best fitness on the Concrete CS, test set.

Table 57 – Kruskal-Wallis Test for the Concrete CS dataset.

Method	Train Pv	Test Pv
GP Repulsors 08	5.35E-06	4.87E-05
GP Repulsors 09	2.81E-05	0.001

The medians of the dataset results in the sets appear different, and the Kruskal Wallis test supports this, the behavior can also be confirmed in the test set line plots of Figures 58 and 59.

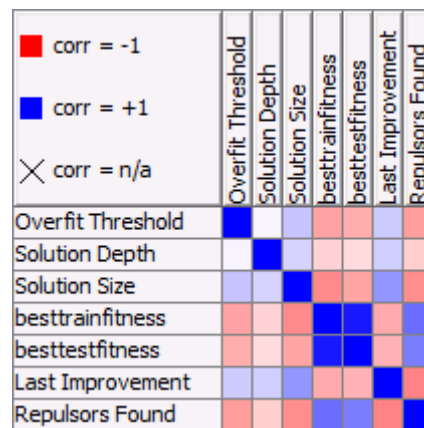


Figure 65: Correlation matrix for the Concrete CS dataset.

A strong relationship between training and test set results is evident in this dataset.

7.2.8. Detailed Analysis of the Wine Quality red dataset.

Wine Quality red dataset (WQR), This is the analysis of one dataset from the Two datasets are included, related to red and white vinho verde wine samples, from the north of Portugal. The goal is to model wine quality based on physicochemical tests (see [Cortez et al., 2009],). There are 11 inputs and one output. More information can be found in the appendices.

7.2.8.1. WQR, Summary statistics.

Table 58 – WQR, Summary statistics for the Training set.

Algorithm	O.F.T.	Min.	Max.	Mean	Median	S.D.	Var.
GP_CLASSIC	N/A	0.490	0.686	0.598	0.604	0.049	0.002
GP_REPULSORS_08	1.01	0.604	1.099	0.766	0.752	0.118	0.014
GP_REPULSORS_08	1.5	0.448	1.199	0.680	0.647	0.141	0.020
GP_REPULSORS_08	2.5	0.523	0.721	0.603	0.599	0.054	0.003
GP_REPULSORS_09	1.01	0.607	1.145	0.827	0.809	0.142	0.020
GP_REPULSORS_09	1.5	0.538	1.019	0.693	0.683	0.111	0.012
GP_REPULSORS_09	2.5	0.447	0.706	0.572	0.563	0.069	0.005

Table 59 – WQR, Summary statistics for the Test set.

Algorithm	O.F.T.	Min.	Max.	Mean	Median	S.D.	Var.
GP_CLASSIC	N/A	0.590	74.029	3.300	0.778	13.362	178.533
GP_REPULSORS_08	1.01	0.563	1.151	0.801	0.807	0.140	0.020
GP_REPULSORS_08	1.5	0.610	5.914	1.020	0.815	0.935	0.874
GP_REPULSORS_08	2.5	0.533	9.552	1.380	0.867	1.735	3.010
GP_REPULSORS_09	1.01	0.639	2.007	0.921	0.847	0.277	0.077
GP_REPULSORS_09	1.5	0.625	3.546	1.060	0.852	0.632	0.400
GP_REPULSORS_09	2.5	0.576	26.299	2.242	0.877	5.266	27.728

In the training set, the best results were obtained by GP_REPULSORS_09, OFT 2.5. In the Test set, the GP_REPULSORS_08 with OFT 1.01 was the one that obtained better results.

7.2.8.2. Repulsors.

Table 60 – WQR, Median number of repulsors found.

Algorithm	O.F.T.		
	1.01	1.5	2.5
GP_REPULSORS_08	82	11.5	0
GP_REPULSORS_09	76	6	1

The number of repulsors diminishes as the overfit threshold increases.

7.2.8.3. Best Individual Generation.

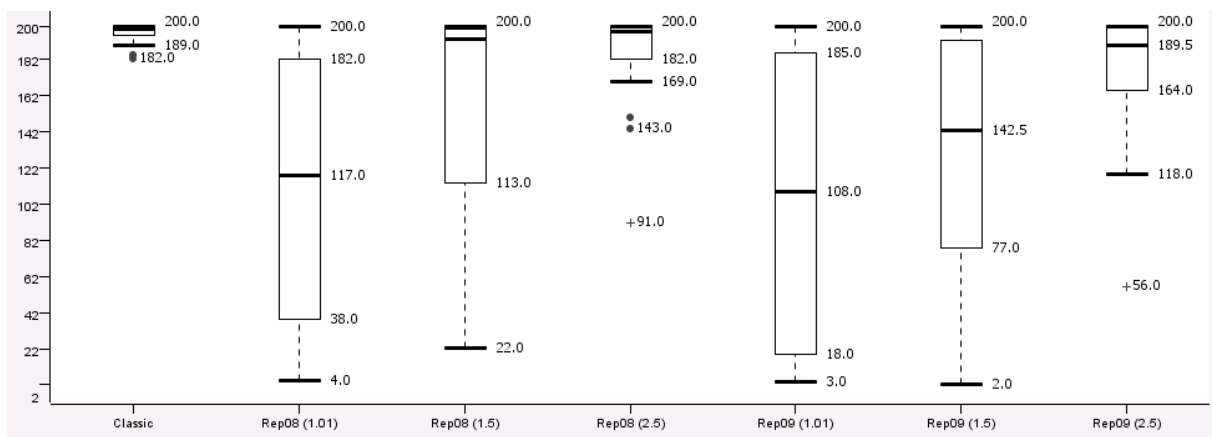


Figure 66: WQR, Best Individual Generation.

The best elements were found, in most cases in the latest generations. A noteworthy exception is the Repulsor based algorithms with OFT 1.01, where the median was found around generation 120 in both algorithms.

7.2.8.4. Individual Size.

Table 61 – Individual size and depth Median.

Algorithm	Classic	Repulsors 08			Repulsors 09		
	N.A.	1.01	1.5	2.5	1.01	1.5	2.5
Individual tree depth	10	4	10	10	3	10	10
Individual size	144	8	118	138	7	115	159

Most solutions grew up to the limit allowed, but, in both repulsors algorithms, when set to OFT 1.01, the depth seems to stop at depth 4, this is more relevant if we consider that the generation where the best solutions were found is also a behavioral exception.

7.2.8.5. WQR, Centrality Measures Graphical Comparison.

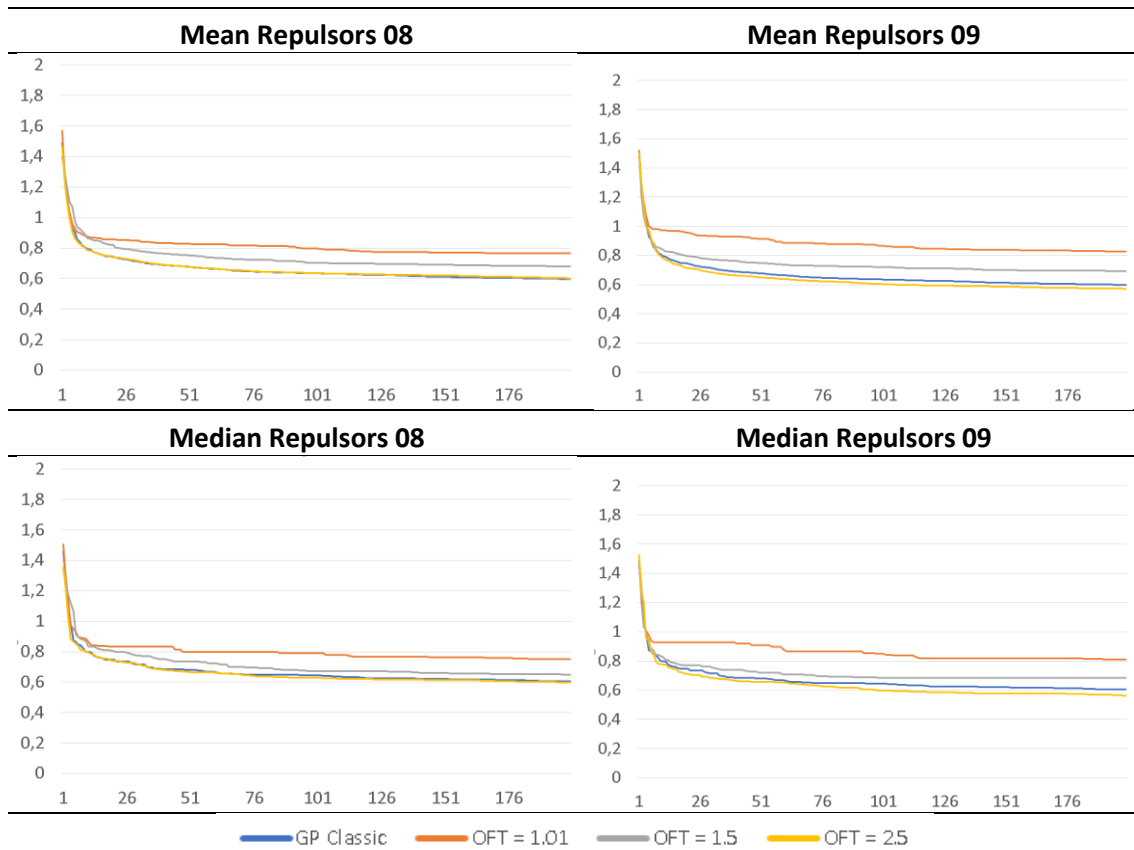


Figure 67: WQR, training set graphical centrality comparison.

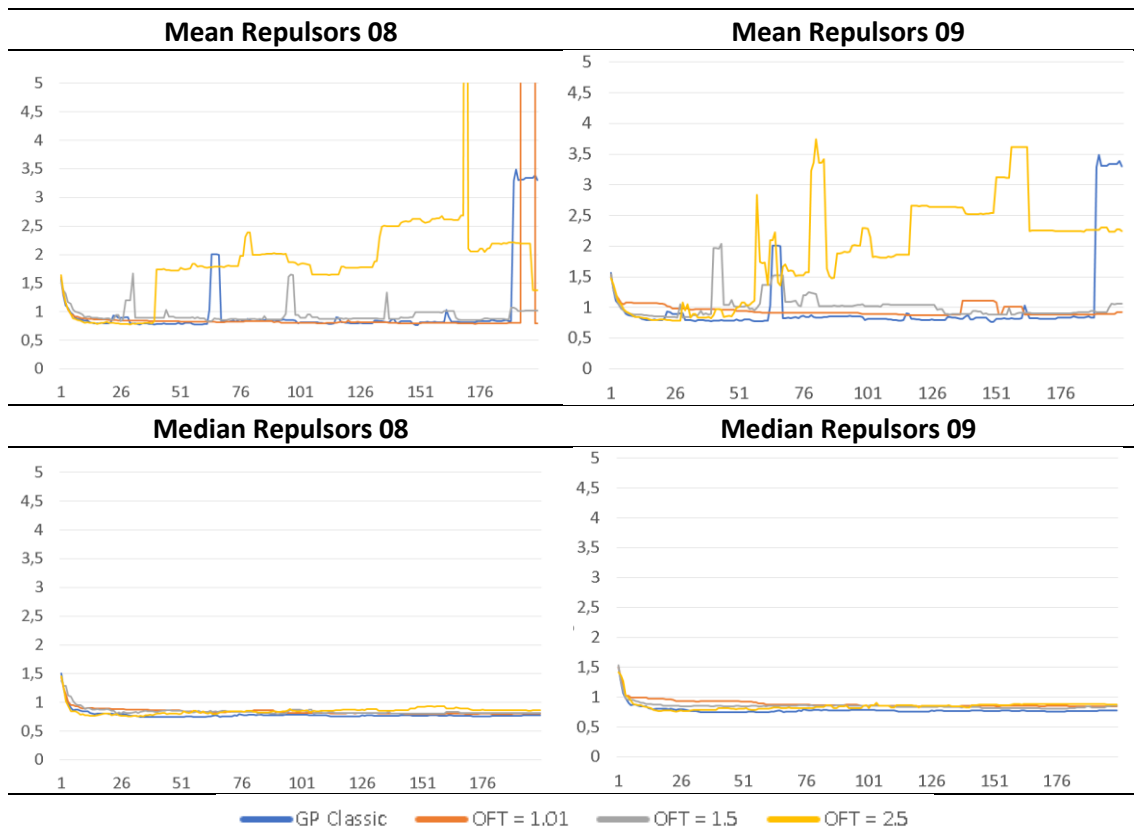


Figure 68: WQR, test set graphical centrality comparison.

7.2.8.6. Individual Conclusions for wine quality red dataset.

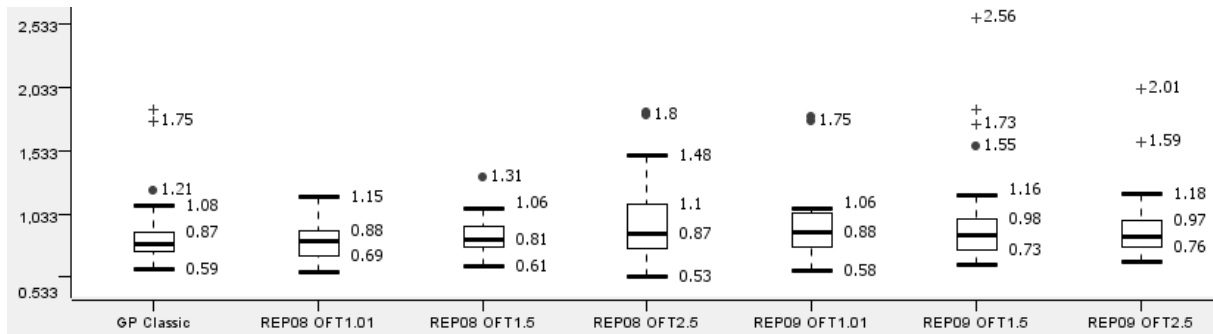


Figure 69: Boxplot of the best fitness on the wine quality red test set.

Table 62 – Kruskal-Wallis Test for the WQR dataset .

Method	Train Pv	Test Pv
GP Repulsors 08	3.42E-08	0.559
GP Repulsors 09	4.73E-11	0.816

In the training set, the Kruskal-Wallis test shows a significant difference in behavior, that is supported by the plots of figure 67. As for the Test set algorithms behaved in a very similar manner in this data set. The Kruskal-Wallis test validates the null hypothesis; there is no significant statistical difference between all results.

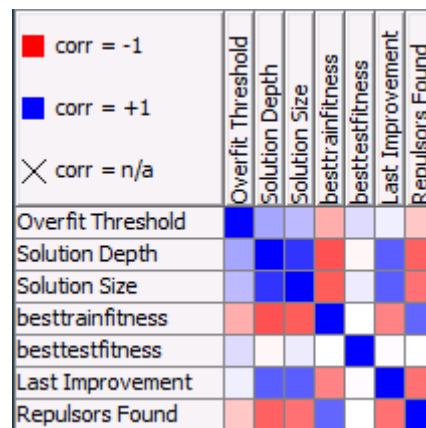


Figure 70: Correlation matrix for wine quality red dataset.

There are several noteworthy correlations, namely depth and size, and the training fitness vs. depth and size. The negative correlation between the number of repulsors found and the solution size and depth may be a sign that the use of repulsors produces smaller individuals, maybe by providing faster training.

7.2.9. Detailed Analysis of the LD50 Toxicity dataset.

The LD50 dataset as described in the paper Improving genetic programming for the prediction of pharmacokinetic parameters (Vanneschi, 2014) ‘Oral bioavailability (% F from now on) is the parameter that measures the percentage of the initial orally submitted drug dose that effectively reaches the systemic blood circulation after being filtered by the liver. Plasma protein binding level (% PPB) consists of the percentage of the initial drug dose which binds plasma proteins. Median oral lethal dose (LD50) refers to the amount of compound required to kill 50% of the test organisms (cavies), and it is currently one of the most used parameters to quantify the toxicity of a candidate new drug’.

It was chosen because it has a significant number of variables, and a low variable to sample ratio, making it a good candidate for overfitting.

7.2.9.1. Summary statistics.

Table 63 – LD50, Summary statistics for the Training set.

Algorithm	O.F.T.	Min.	Max.	Mean	Median	S.D.	Var.
GP_CLASSIC	N/A	1336	2412	1707	1661	230	52750
GP_REPULSORS_08	1.01	1427	2156	1797	1797	183	33508
GP_REPULSORS_08	1.5	1336	2102	1664	1659	169	28509
GP_REPULSORS_08	2.5	1245	1980	1611	1601	175	30703
GP_REPULSORS_09	1.01	1476	2452	1841	1757	227	51374
GP_REPULSORS_09	1.5	1220	2375	1797	1762	344	118641
GP_REPULSORS_09	2.5	1264	2582	1643	1603	265	70024

Table 64 – LD50, Summary statistics for the Test set.

Algorithm	O.F.T.	Min.	Max.	Mean	Median	S.D.	Var.
GP_CLASSIC	N/A	1685	6769	2805	2408	1203	1446268
GP_REPULSORS_08	1.01	1760	9571	3603	2420	2230	4971415
GP_REPULSORS_08	1.5	1835	15307	3978	2514	3356	11265072
GP_REPULSORS_08	2.5	1795	29828	4393	2539	5258	27649748
GP_REPULSORS_09	1.01	1568	28954	3448	2249	5019	25194126
GP_REPULSORS_09	1.5	1520	12903	2770	2252	2030	4121821
GP_REPULSORS_09	2.5	1651	19787	4571	2443	4977	24772931

GP_REPULSORS_08 achieved the best overall training set results; the best overall training set result was achieved using GP_REPULSORS_09.

For the test set, the best median was obtained with GP_REPULSORS_09 OFT 1.01, and the best individual result was obtained using GP_REPULSORS_09 OFT 1.5.

7.2.9.2. Repulsors.

Table 65 – LD50, Median number of repulsors found.

Algorithm	O.F.T.		
	1.01	1.5	2.5
GP_REPULSORS_08	41	2.5	0
GP_REPULSORS_09	55	1	0

The number of repulsors decreases as the value of the overfit threshold increases. Repulsors are practically not found with overfit threshold above 1.01.

7.2.9.1. Best Individual Generation.

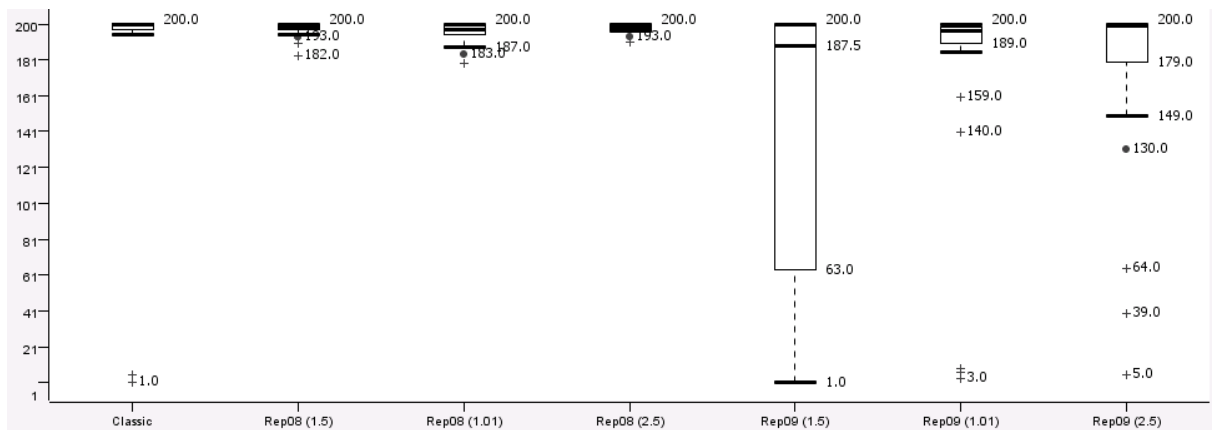


Figure 71:LD50 toxicity, Best Individual Generation.

The Best solutions are found in the later generations; this dataset could use some more generations to evolve.

7.2.9.2. Individual Size.

Table 66 – Individual size and depth Median.

Algorithm	Classic	Repulsors 08			Repulsors 09		
	N.A.	1.01	1.5	2.5	1.01	1.5	2.5
Individual Tree Depth	10	10	10	10	10	10	10
Individual size	144	8	118	138	7	115	159

The programs grew to the depth limit allowed; this can be seen by the median tree depth of 10 levels. The plots in figure 72 indicate that the algorithm could have been allowed further training with good results.

7.2.9.3. LD50, Centrality Measures Graphical Comparison.

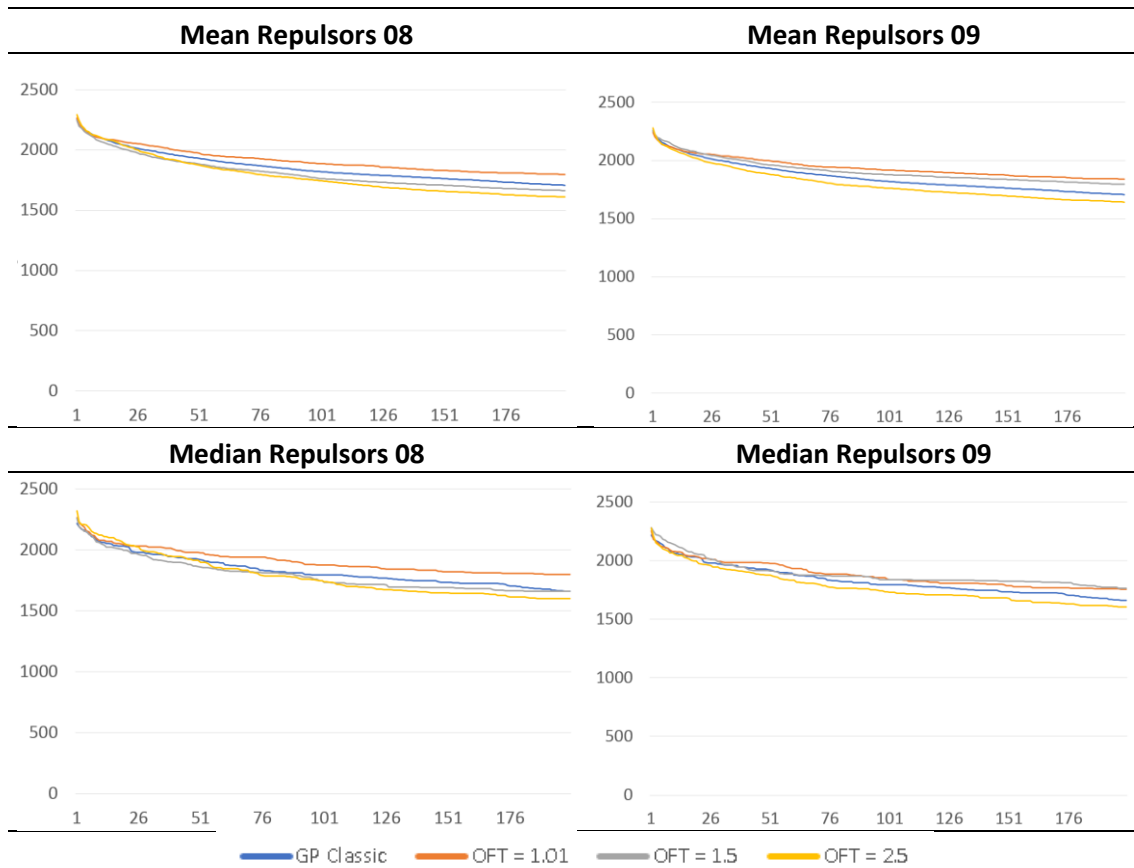


Figure 72: LD50, training set graphical centrality comparison.

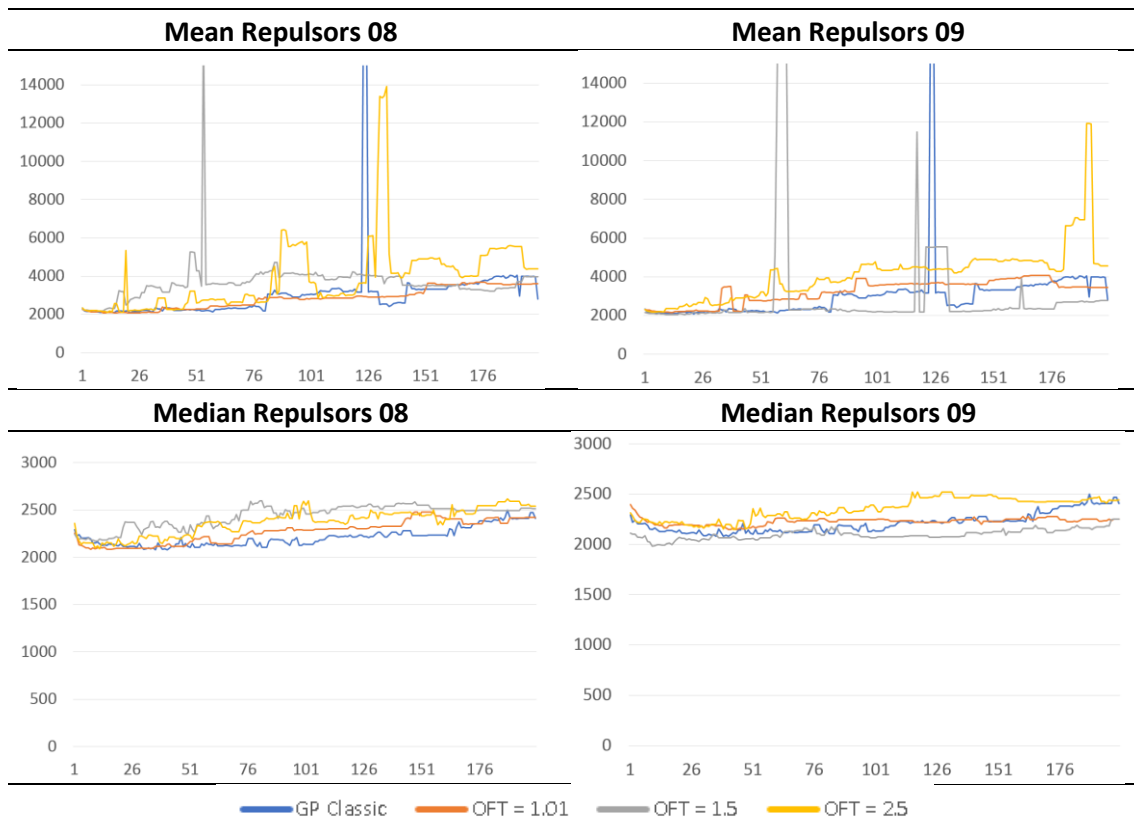


Figure 73: LD50, test set graphical centrality comparison.

7.2.9.4. Individual Conclusions for LD50 toxicity dataset.

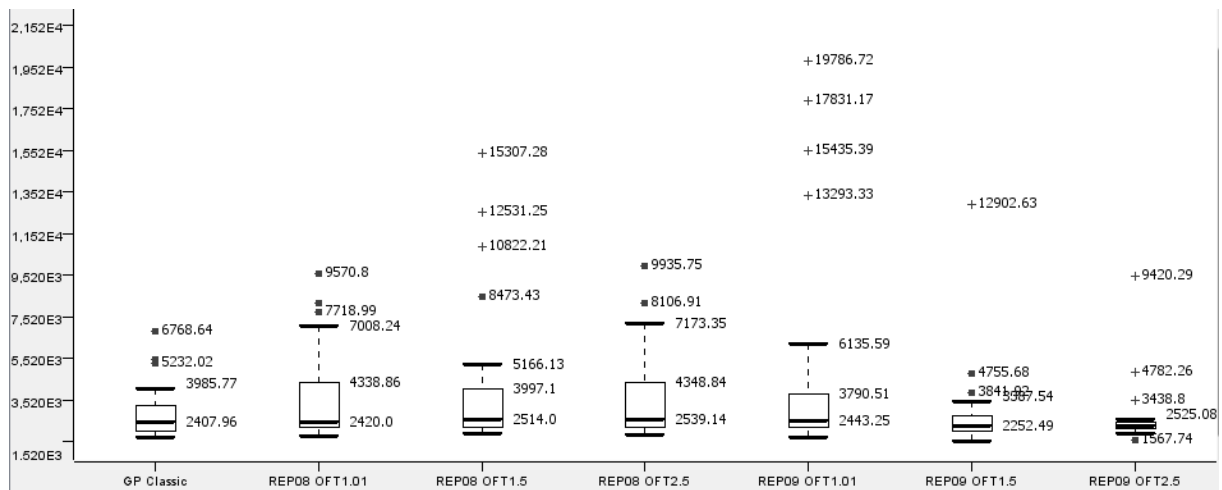


Figure 74: Boxplot of the best fitness on the LD 50 Toxicity test set.

Table 67 – Kruskal-Wallis Test for the LD50 dataset.

Method	Train Pv	Test Pv
GP Repulsors 08	0.012	0.913
GP Repulsors 09	0.038	0.620

The Generations graphs show a difference in behavior between all methods GP_REPULSORS_09 produced the most coherent results, as can be seen in the tight box plots, the Kruskal-Wallis test validates that there is a difference between the results of the methods used on the test set, but the learning behavior is different, that can be seen in the way that the learning curves are presented in figure 68

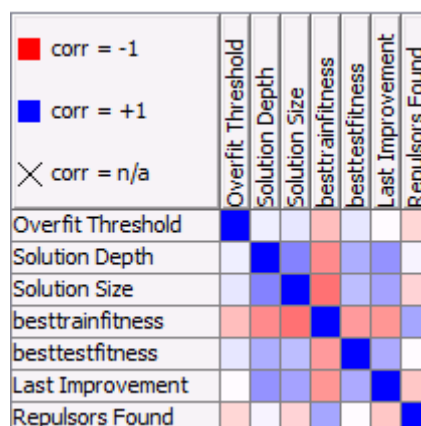


Figure 75: Correlation matrix for LD50 Toxicity dataset.

Correlations appear weak but still existing, interesting the fact that the solution size and depth are inversely correlated with the training fitness but positively correlated with the test fitness.

8. CONCLUSIONS

Standard GP takes populations as isolated entities when it comes to fitness evaluation, an individual is only compared to individuals inside the same population and no prior performance knowledge is transferred from one generation into another, in nature however, this is not the norm, and many species penalize characteristics that act as indicators of an undesired trait or behavior by out casting these individuals socially. Inspired by this, we implemented repulsor based GP. Repulsors are individuals that performed satisfactorily on the training set but performed poorly on a validation set. If found, these individuals will be added to a list, called repulsor list, on this implementation of repulsor based GP the Standard GP is extended by the insertion of an additional selection phase that will favor individuals whose behavior is different from the ones in the repulsor list.

The objective of this thesis was to study the behavior of repulsor based genetic programming and access the impact of the use of repulsors on the generalization ability of GP. Repulsors provide the algorithm with information on the results achieved by other individuals and affect the selection of others. To achieve this objective, we developed a program that implemented Standard GP and two repulsor based algorithms; this program was then run against several datasets with different characteristics and sizes. The results were then collected and analyzed.

Running the program against the data sets took a considerable amount of computer time, and resources, albeit there is plenty of duplication and many files that ended up being unused, the runs created take up more than 500 GB of space in 6.3 million files and use 2.5 GB of database disk space. The runs took about a week to complete using an 8-core hyper threaded Xeon microprocessor with 48GB of RAM that usually ran 8 to 10 GP instances at any given moment. Choosing parameter values also involved a lot of work and experimentation.

The results suggest that the use of repulsor based algorithms do have a noticeable effect on training and generalization abilities of GP.

Results have shown that in almost all datasets the repulsor based algorithms outperformed standard GP. In the training set, 7 out of 9 of the datasets achieved better results with repulsor based algorithms, indicating an improved learning ability. For the test set, 5 out of 9 got better results with Repulsor based GP, suggesting increased generalization capacity.

Datasets `fittable_fn05` and `fittable_fn07` were the only ones that had the best results with GP CLASSIC algorithm in both training and testing; this may be due to the simplicity of the functions that generated them.

With repulsor based GP we used an additional parameter, OFT, the overfit threshold defines when the algorithm should consider the solution to be overfitting. We verified that in practice, the lower the OFT, the more repulsors are found; this is a rather intuitive result that was expected, experimentation confirmed this hypothesis, subsequently, the Higher the OFT, the fewer repulsors are found.

The OFT appears to be an important parameter with repulsor based GP. In the learning phase with the use of repulsors the lower the OFT, the faster the algorithm seems to stop learning, this can be seen in the centrality measures graphical comparison presented on the detailed analysis of individual dataset results (section 7.2) where the learning curve seems to stabilize faster where the OFT is at a lower value, this does, however, translate to a lower median fitness of the solutions, implying that

setting the threshold too low and detecting too many repulsors may indeed hinder the performance of the algorithm.

Due to processing constraints, we had to limit the number of generations and the program size. On some datasets, the limit of 200 generations appears to be a limitation to the learning process as the learning curve does not look to have stabilized when reaching the parametrized limit. Most of the datasets produced individuals that grew to the maximum allowed size.

8.1. ANSWERS TO THE RESEARCH QUESTIONS

- Can the use of repulsors affect/limit the overfitting effect that occurs in genetic programming? If so, what kind of effect will it have?

Yes, the use of repulsors can and does affect the overfitting effect, the learning ratios change, and the results appear to be better in both learning and test sets. Changes can be observed in the learning curves, and the solutions seem better on the test sets. There was no observable difference in size and depth of individuals.

- Does the use of Repulsors have an impact on the ability to learn of a GP?

Yes, when using repulsor based algorithms, the training tends to stop earlier. The training results are also better in 7 out of 9 cases.

- Does the use of Repulsors affect the capacity to generalize?

Yes, in 5 out of the 9 analyzed datasets, repulsor based algorithms provided better results on the test set.

9. LIMITATIONS AND RECOMMENDATIONS FOR FUTURE WORKS

The major limitations found were in processing the datasets and storage capacity to store the detailed results, so, future work is to run the algorithm with looser constraints in both number of generations and most importantly individual depth. On the real life datasets we should also run the algorithms with different amounts of training data available. Since we were comparing techniques and datasets varying only some parameters to all of the tests (OFT and choice of algorithm), no parameter optimization was even tried, so it may be the case that a simple parameter tweak could provide better results than the ones we obtained.

An interesting addition would be the use of multi-objective optimization, where the repulsor distance would be evaluated at the same time as the fitness function. NSGA-II(Deb et al., 2002) appears as good candidate to begin this approach.

Extend and improve the overfit detection system; the current system only detects overfitting from the current generation comparing it to the previous one, so we could analyze not only the previous generation but also more generation behind it. Furthermore, we only test the best individual from the generation; we should extend this test to other individuals. In the Overfitting detection system, a way of detecting false positives would also be useful.

Evaluation of syntactical repulsors. All the work here was developed on semantic results; it would be interesting to analyze each individual and locate the functional part or parts that overfits.

Combine all the above improvements.

Combine repulsors with other techniques.

Since at the finish of each run we end up with a list of repulsors, it makes sense to pass this list to a new run, providing the run of prior knowledge of repulsors, somewhat mimicking the passing of knowledge between populations on a society.

10.BIBLIOGRAPHY

- Becker, L. A., & Seshadri, M. (n.d.). Comprehensibility & Overfitting Avoidance in Genetic Programming for Technical Trading Rules Lee A . Becker & Mukund Seshadri Department of Computer Science Worcester Polytechnic Institute Abstract.
- Berthold, M. R., Cebon, N., Dill, F., Gabriel, T. R., Kötter, T., Meinl, T., ... Wiswedel, B. (2009). KNIME - The Konstanz Information Miner. *SIGKDD Explorations*, 11(1), 26–31. <https://doi.org/10.1145/1656274.1656280>
- Briscoe, E., & Feldman, J. (2011). Conceptual complexity and the bias/variance tradeoff. *Cognition*, 118(1), 2–16. <https://doi.org/10.1016/j.cognition.2010.10.004>
- Cortez, P., Cerdeira, A., Almeida, F., Matos, T., & Reis, J. (2009). Modeling wine preferences by data mining from physicochemical properties. *Decision Support Systems*, 47(4), 547–553. <https://doi.org/10.1016/j.dss.2009.05.016>
- Darwin, C. (1859). On the origins of species by means of natural selection. *London: Murray*, 247. <https://doi.org/10.1126/science.146.3640.51-b>
- Deb, K., Pratap, A., Agarwal, S., & Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2), 182–197. <https://doi.org/10.1109/4235.996017>
- Goncalves, I., & Carreiras, M. B. (2012). Random Sampling Technique for Overfitting Control in Genetic Programming. *Control*, 218–229.
- Grus, J. (2015). *Data Science from Scratch*. (O'Reilly, Ed.) (1st ed.). O'Reilly.
- Hutchison, D., & Mitchell, J. C. (2003). *Genetic Programming. EuroGP*. https://doi.org/10.1007/3-540-68339-9_34
- John R. Koza. (n.d.). Home Page of John R. Koza. Retrieved from <http://www.genetic-programming.com/johnkoza.html>
- Keijzer, M. (2003). Improving Symbolic Regression with Interval Arithmetic and Linear Scaling. *Genetic Programming Proceedings of EuroGP2003*, 2610, 70–82. https://doi.org/10.1007/3-540-36599-0_7
- Koza, J. R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*.
- Koza, J. R. (1994). Introduction to genetic programming. In *Advances in Genetic Programming* (pp. 21–42). <https://doi.org/doi:10.1145/1388969.1389057>
- Langdon, W., & Banzhaf, W. (2008). A SIMD Interpreter for Genetic Programming on GPU Graphics Cards. *Genetic Programming*, 4971, 73–85.
- Lichman, M. (2013). UCI Machine Learning Repository. Retrieved May 21, 2017, from <https://archive.ics.uci.edu/ml/datasets.html>
- McConnell, S. (2004). *Code Complete: A Practical Handbook of Software Construction*. (M. Press, Ed.) (2nd ed.). Microsoft press.
- Mouffok, O. (2014). Using Genetic Algorithms in Integer Programming for Decision Support, 5(1), 55–63.

- Munoz, A. (2014). Machine Learning and Optimization. *Courant Institute of Mathematical Sciences*. Retrieved from https://www.cims.nyu.edu/~munoz/files/ml_optimization.pdf
- O'Neill, M., Vanneschi, L., Gustafson, S., & Banzhaf, W. (2010). Open issues in Genetic Programming. *Genetic Programming and Evolvable Machines*, 11(3–4), 339–363. <https://doi.org/10.1007/s10710-010-9113-2>
- Piotrowski, A. P., & Napiorkowski, J. J. (2013). A comparison of methods to avoid overfitting in neural networks training in the case of catchment runoff modelling. *Journal of Hydrology*, 476, 97–111. <https://doi.org/10.1016/j.jhydrol.2012.10.019>
- Poli, R., Langdon, W. B., & McPhee, N. F. (2008). *A Field Guide to Genetic Programming*. Wyvern. Retrieved from [http://www.essex.ac.uk/wyvern/2008-04/Wyvern April 08 7126.pdf](http://www.essex.ac.uk/wyvern/2008-04/Wyvern%20April%2008%207126.pdf)
- Sivanandam, S. N., & Deepa, S. N. (2008). Introduction to Genetic Algorithms, (November), 193. <https://doi.org/10.1007/978-3-540-73190-0>
- Subramanian, J., & Simon, R. (2013). Overfitting in prediction models - Is it a problem only in high dimensions? *Contemporary Clinical Trials*, 36(2), 636–641. <https://doi.org/10.1016/j.cct.2013.06.011>
- Tuite, C., Agapitos, A., O'Neill, M., & Brabazon, A. (2011). Early stopping criteria to counteract overfitting in genetic programming. *Proceedings of the 13th Annual Conference Companion on Genetic and Evolutionary Computation - GECCO '11*, 203. <https://doi.org/10.1145/2001858.2001971>
- Vanneschi, L. (2010). Measuring Bloat , Overfitting and Functional Complexity, 877–884.
- Vanneschi, L. (2014). Improving genetic programming for the prediction of pharmacokinetic parameters. *Memetic Computing*, 6(4), 255–262. <https://doi.org/10.1007/s12293-014-0143-9>
- Vanneschi, L., & Gustafson, S. (2009). Using crossover based similarity measure to improve genetic programming generalization ability. *GECCO '09: Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation*, 1139–1146.
- Yeh, I.-C. (1998). Modeling of Strength of High-Performance Concrete Using Artificial Neural Networks. *Journal of Civil Engineering*, 28(12), 1797–1808. <https://doi.org/10.1017/CBO9781107415324.004>
- Zaji, A. H., & Bonakdari, H. (2015). Application of artificial neural network and genetic programming models for estimating the longitudinal velocity field in open channel junctions. *Flow Measurement and Instrumentation*, 41, 81–89. <https://doi.org/10.1016/j.flowmeasinst.2014.10.011>

11.APPENDIX

11.1. PROGRAM SOURCE CODE

The developed software source code is available at : <https://github.com/jcanelhas/repulsors01>

11.2. DATASET DESCRIPTIONS.

These datasets(Lichman, 2013) are available at <https://archive.ics.uci.edu/ml/datasets.html>. The descriptions in this appendix are direct copies of the web page information.

11.2.1. Combined Cycle Power Plant data set (CCPP).

Available in: <https://archive.ics.uci.edu/ml/datasets/Combined+Cycle+Power+Plant>

Abstract: The dataset contains 9568 data points collected from a Combined Cycle Power Plant over 6 years (2006-2011), when the plant was set to work with full load.

Source:

Pınar Tüfekci, Çorlu Faculty of Engineering, Namık Kemal University, TR-59860 Çorlu, Tekirdağ, Turkey
Email: ptufekci '@' nku.edu.tr

Heysem Kaya, Department of Computer Engineering, Boğaziçi University, TR-34342, Beşiktaş, İstanbul, Turkey
Email: heysem '@' boun.edu.tr

Data Set Information:

The dataset contains 9568 data points collected from a Combined Cycle Power Plant over 6 years (2006-2011), when the power plant was set to work with full load. Features consist of hourly average ambient variables Temperature (T), Ambient Pressure (AP), Relative Humidity (RH) and Exhaust Vacuum (V) to predict the net hourly electrical energy output (EP) of the plant. A combined cycle power plant (CCPP) is composed of gas turbines (GT), steam turbines (ST) and heat recovery steam generators. In a CCPP, the electricity is generated by gas and steam turbines, which are combined in one cycle, and is transferred from one turbine to another. While the Vacuum is collected from and has effect on the Steam Turbine, the other three of the ambient variables effect the GT performance. For comparability with our baseline studies, and to allow 5x2 fold statistical tests be carried out, we provide the data shuffled five times. For each shuffling 2-fold CV is carried out and the resulting 10 measurements are used for statistical testing. We provide the data both in .ods and in .xlsx formats.

Attribute Information:

Features consist of hourly average ambient variables

- Temperature (T) in the range 1.81°C and 37.11°C,
- Ambient Pressure (AP) in the range 992.89-1033.30 milibar,
- Relative Humidity (RH) in the range 25.56% to 100.16%
- Exhaust Vacuum (V) in the range 25.36-81.56 cm Hg
- Net hourly electrical energy output (EP) 420.26-495.76 MW

The averages are taken from various sensors located around the plant that record the ambient variables every second. The variables are given without normalization.

11.2.2. Wine Quality Red.

Available in: <https://archive.ics.uci.edu/ml/datasets/Wine+Quality>

Only the wine quality red set was used

Abstract: Two datasets are included, related to red and white vinho verde wine samples, from the north of Portugal. The goal is to model wine quality based on physicochemical tests (see [Cortez et al., 2009],).

Data Set Information:

The two datasets are related to red and white variants of the Portuguese "Vinho Verde" wine. For more details, consult: [Web Link] or the reference [Cortez et al., 2009]. Due to privacy and logistic issues, only physicochemical (inputs) and sensory (the output) variables are available (e.g. there is no data about grape types, wine brand, wine selling price, etc.). These datasets can be viewed as classification or regression tasks. The classes are ordered and not balanced (e.g. there are much more normal wines than excellent or poor ones). Outlier detection algorithms could be used to detect the few excellent or poor wines. Also, we are not sure if all input variables are relevant. So it could be interesting to test feature selection methods.

Attribute Information:

For more information, read [Cortez et al., 2009].

Input variables (based on physicochemical tests):

- 1 - fixed acidity
- 2 - volatile acidity
- 3 - citric acid
- 4 - residual sugar
- 5 - chlorides
- 6 - free sulfur dioxide
- 7 - total sulfur dioxide
- 8 - density
- 9 - pH
- 10 - sulphates
- 11 - alcohol

Output variable (based on sensory data):

- 12 - quality (score between 0 and 10)

11.2.3. Concrete Slump Test

Available in: <https://archive.ics.uci.edu/ml/datasets/Concrete+Slump+Test>

This test used only the third output.

Source:

Donor: I-Cheng Yeh Email: icyeh '@' chu.edu.tw Institution: Department of Information Management, Chung-Hua University (Republic of China) Other contact information: Department of Information Management, Chung-Hua University, Hsin Chu, Taiwan 30067, R.O.C.

Data Set Information:

The data set includes 103 data points. There are 7 input variables, and 3 output variables in the data set.

The initial data set included 78 data. After several years, we got 25 new data points.

Attribute Information:

Input variables (7)(component kg in one M³ concrete):

Cement

Slag

Fly ash

Water

SP

Coarse Aggr.

Fine Aggr.

Output variables (3):

SLUMP (cm)

FLOW (cm)

28-day Compressive Strength (Mpa)

11.2.4. Concrete Compressive Strength dataset.

Available in : <https://archive.ics.uci.edu/ml/datasets/Concrete+Compressive+Strength>

Data Set Information:

Number of instances 1030

Number of Attributes 9

Attribute breakdown 8 quantitative input variables, and 1 quantitative output variable

Missing Attribute Values None

Attribute Information:

Given are the variable name, variable type, the measurement unit and a brief description. The concrete compressive strength is the regression problem. The order of this listing corresponds to the order of numerals along the rows of the database.

Name -- Data Type -- Measurement -- Description

Cement (component 1) -- quantitative -- kg in a m3 mixture -- Input Variable

Blast Furnace Slag (component 2) -- quantitative -- kg in a m3 mixture -- Input Variable

Fly Ash (component 3) -- quantitative -- kg in a m3 mixture -- Input Variable

Water (component 4) -- quantitative -- kg in a m3 mixture -- Input Variable

Superplasticizer (component 5) -- quantitative -- kg in a m3 mixture -- Input Variable

Coarse Aggregate (component 6) -- quantitative -- kg in a m3 mixture -- Input Variable

Fine Aggregate (component 7) -- quantitative -- kg in a m3 mixture -- Input Variable

Age -- quantitative -- Day (1~365) -- Input Variable

Concrete compressive strength -- quantitative -- MPa -- Output Variable

11.2.5. Toxicity dataset.

LD50 dataset, used in the paper Improving genetic programming for the prediction of pharmacokinetic parameters. (Vanneschi, 2014)

Can be downloaded from the webpage: <http://personal.disco.unimib.it/Vanneschi/toxicity.txt>

11.3. RUN INFORMATION

General Information on the program runs.

11.3.1. Number of runs per dataset table.

		1.01		1.5		2.5	
	GPClassic	REP 08	REP 09	REP 08	REP 09	REP 08	REP 09
fittable_sin	30	30	30	30	30	30	30
fittable_fn05	30	30	30	30	30	30	30
fittable_fn07	30	30	30	30	30	30	30
fittable_fn08	30	30	30	30	30	30	30
wine qty red	30	30	30	30	30	30	30
CCCP	30	30	30	30	30	30	30
Concrete Slump	30	30	30	30	30	30	30
Concrete	30	30	30	30	30	30	30
LD50 Toxicity	30	30	30	30	30	30	30

11.3.2. Run data

Run data used for this dissertation is available in github.

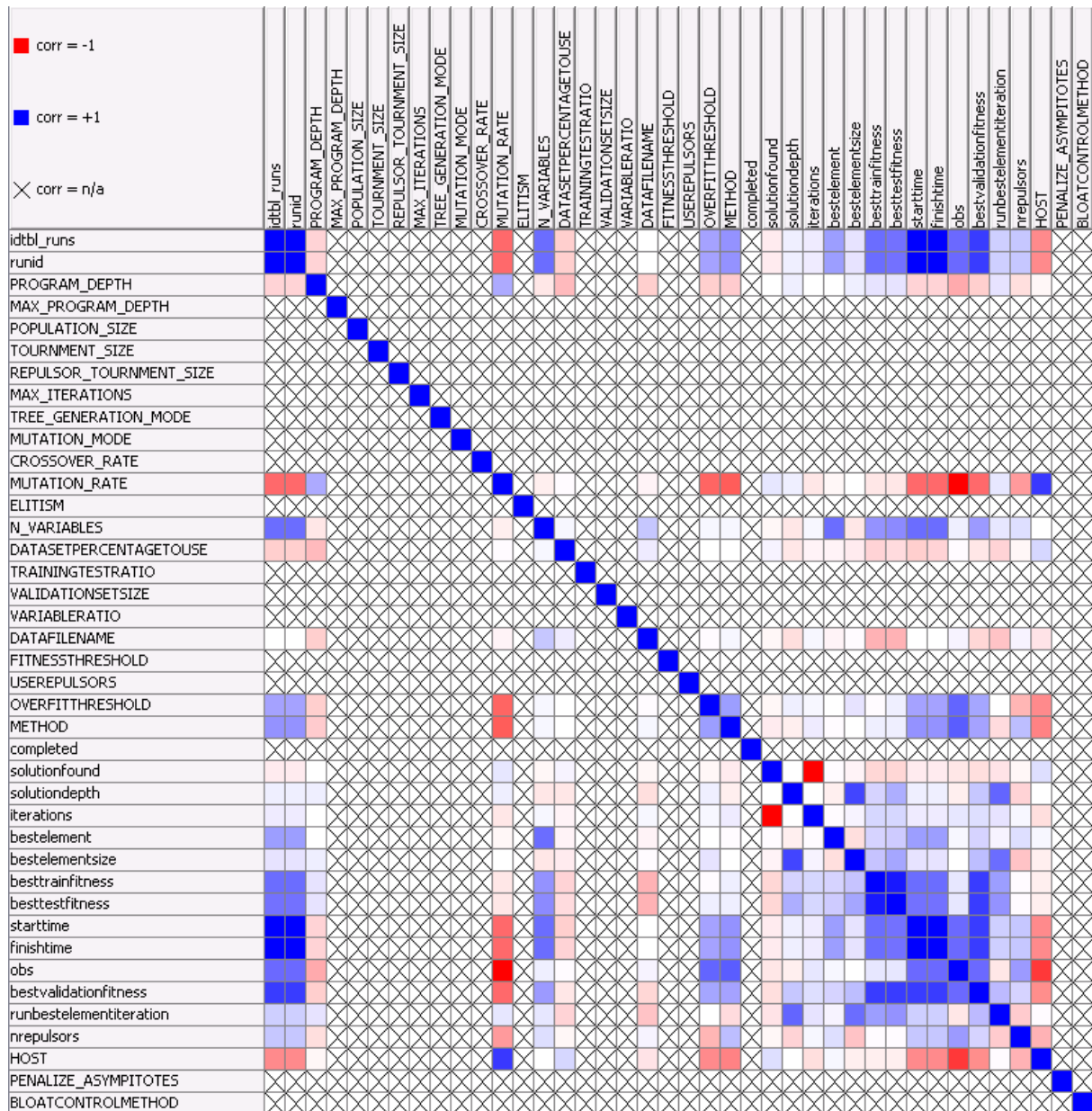
<https://github.com/jcanelhas/repulsors01>

11.4. KRUSKAL-WALLIS TEST DETAIL

DATAFILENAME	METHOD	H-Value - Test	p-value - Test	H-Value - Train	p-value - Train
datasets/CCPP/Folds5x2_pp.txt	GP Repulsors 08	4,996	0,544	2,253	0,895
datasets/CCPP/Folds5x2_pp.txt	GP Repulsors 09	7,987	0,239	9,000	0,174
datasets/Concrete_Data.txt	GP Repulsors 08	29,507	4,877E-05	34,513	5,355E-06
datasets/Concrete_Data.txt	GP Repulsors 09	22,139	0,001	30,762	2,814E-05
datasets/concrete_slump/slump_t	GP Repulsors 08	17,645	0,007	40,163	4,231E-07
datasets/concrete_slump/slump_t	GP Repulsors 09	17,628	0,007	33,760	7,484E-06
datasets/fittable_fn05.txt	GP Repulsors 08	12,434	0,053	13,129	0,041
datasets/fittable_fn05.txt	GP Repulsors 09	16,750	0,010	23,343	0,001
datasets/fittable_fn07.txt	GP Repulsors 08	3,899	0,690	1,283	0,973
datasets/fittable_fn07.txt	GP Repulsors 09	5,834	0,442	11,733	0,068
datasets/fittable_fn08.txt	GP Repulsors 08	10,213	0,116	1,650	0,949
datasets/fittable_fn08.txt	GP Repulsors 09	6,290	0,391	7,376	0,287
datasets/fittable_sin.txt	GP Repulsors 08	4,159	0,655	5,296	0,506
datasets/fittable_sin.txt	GP Repulsors 09	12,319	0,055	7,707	0,260
datasets/toxicity_clean.txt	GP Repulsors 08	2,076	0,913	16,421	0,012
datasets/toxicity_clean.txt	GP Repulsors 09	4,424	0,620	13,327	0,038
datasets/wine-quality/winequality	GP Repulsors 08	4,881	0,559	45,679	3,429E-08
datasets/wine-quality/winequality	GP Repulsors 09	2,943	0,816	59,891	4,736E-11

11.5. CORRELATION DETAILS

11.5.1. Correlation matrix for run summary.



11.5.2. Correlation Table by dataset.

		Overfit Threshold	Solution Depth	Solution Size	besttrainfitness	besttestfitness	Last Improvement	Repulsors Found
CCPP Folds	Overfit Threshold	1,000	0,055	0,169	0,010	0,074	0,039	-0,313
	Solution Depth	0,055	1,000	0,255	-0,089	-0,054	0,213	-0,239
	Solution Size	0,169	0,255	1,000	0,111	0,222	0,401	-0,400
	besttrainfitness	0,010	-0,089	0,111	1,000	0,851	0,071	0,064
	besttestfitness	0,074	-0,054	0,222	0,851	1,000	0,151	-0,059
	Last Improvement	0,039	0,213	0,401	0,071	0,151	1,000	-0,422
	Repulsors Found	-0,313	-0,239	-0,400	0,064	-0,059	-0,422	1,000
Concrete	Overfit Threshold	1,000	0,051	0,244	-0,368	-0,320	0,205	-0,390
	Solution Depth	0,051	1,000	0,183	-0,176	-0,149	0,194	-0,188
	Solution Size	0,244	0,183	1,000	-0,458	-0,357	0,406	-0,441
	besttrainfitness	-0,368	-0,176	-0,458	1,000	0,896	-0,333	0,571
	besttestfitness	-0,320	-0,149	-0,357	0,896	1,000	-0,302	0,501
	Last Improvement	0,205	0,194	0,406	-0,333	-0,302	1,000	-0,493
	Repulsors Found	-0,390	-0,188	-0,441	0,571	0,501	-0,493	1,000
Concrete Slump	Overfit Threshold	1,000	0,127	0,142	-0,201	-0,038	0,006	-0,308
	Solution Depth	0,127	1,000	0,325	-0,181	-0,096	0,203	-0,286
	Solution Size	0,142	0,325	1,000	-0,259	-0,140	0,313	-0,221
	besttrainfitness	-0,201	-0,181	-0,259	1,000	0,679	-0,262	0,620
	besttestfitness	-0,038	-0,096	-0,140	0,679	1,000	-0,212	0,417
	Last Improvement	0,006	0,203	0,313	-0,262	-0,212	1,000	-0,414
	Repulsors Found	-0,308	-0,286	-0,221	0,620	0,417	-0,414	1,000
fittable fn05	Overfit Threshold	1,000	-0,076	-0,056	0,065	0,086	-0,087	0,023
	Solution Depth	-0,076	1,000	0,949	-0,433	-0,241	0,593	-0,052
	Solution Size	-0,056	0,949	1,000	-0,394	-0,202	0,565	-0,097
	besttrainfitness	0,065	-0,433	-0,394	1,000	0,850	-0,268	0,037
	besttestfitness	0,086	-0,241	-0,202	0,850	1,000	-0,103	0,036
	Last Improvement	-0,087	0,593	0,565	-0,268	-0,103	1,000	0,077
	Repulsors Found	0,023	-0,052	-0,097	0,037	0,036	0,077	1,000
fittable fn07	Overfit Threshold	1,000	0,012	0,033	0,004	0,150	-0,261	-0,073
	Solution Depth	0,012	1,000	0,942	-0,785	-0,052	0,678	0,126
	Solution Size	0,033	0,942	1,000	-0,763	-0,015	0,666	0,057
	besttrainfitness	0,004	-0,785	-0,763	1,000	0,188	-0,588	-0,099
	besttestfitness	0,150	-0,052	-0,015	0,188	1,000	-0,177	-0,080
	Last Improvement	-0,261	0,678	0,666	-0,588	-0,177	1,000	0,128
	Repulsors Found	-0,073	0,126	0,057	-0,099	-0,080	0,128	1,000
fittable fn08	Overfit Threshold	1,000	0,064	0,059	-0,099	0,042	0,030	-0,158
	Solution Depth	0,064	1,000	0,830	-0,718	0,327	0,675	0,317
	Solution Size	0,059	0,830	1,000	-0,579	0,347	0,625	0,265
	besttrainfitness	-0,099	-0,718	-0,579	1,000	-0,344	-0,455	-0,285
	besttestfitness	0,042	0,327	0,347	-0,344	1,000	0,152	0,324
	Last Improvement	0,030	0,675	0,625	-0,455	0,152	1,000	0,110
	Repulsors Found	-0,158	0,317	0,265	-0,285	0,324	0,110	1,000
fittable sin	Overfit Threshold	1,000	-0,149	-0,078	0,151	0,147	-0,090	-0,153
	Solution Depth	-0,149	1,000	0,422	-0,315	-0,200	0,327	-0,132
	Solution Size	-0,078	0,422	1,000	-0,062	0,100	0,230	-0,070
	besttrainfitness	0,151	-0,315	-0,062	1,000	0,708	-0,196	-0,056
	besttestfitness	0,147	-0,200	0,100	0,708	1,000	-0,187	0,019
	Last Improvement	-0,090	0,327	0,230	-0,196	-0,187	1,000	-0,111
	Repulsors Found	-0,153	-0,132	-0,070	-0,056	0,019	-0,111	1,000
LD50 Toxicity	Overfit Threshold	1,000	0,069	0,102	-0,260	0,105	0,022	-0,169
	Solution Depth	0,069	1,000	0,485	-0,468	0,322	0,435	0,056
	Solution Size	0,102	0,485	1,000	-0,549	0,262	0,370	-0,182
	besttrainfitness	-0,260	-0,468	-0,549	1,000	-0,403	-0,416	0,352
	besttestfitness	0,105	0,322	0,262	-0,403	1,000	0,328	-0,028
	Last Improvement	0,022	0,435	0,370	-0,416	0,328	1,000	-0,222
	Repulsors Found	-0,169	0,056	-0,182	0,352	-0,028	-0,222	1,000
wine quality	Overfit Threshold	1,000	0,343	0,270	-0,321	0,142	0,075	-0,227
	Solution Depth	0,343	1,000	0,787	-0,687	-0,032	0,630	-0,622
	Solution Size	0,270	0,787	1,000	-0,633	0,086	0,636	-0,560
	besttrainfitness	-0,321	-0,687	-0,633	1,000	0,010	-0,492	0,609
	besttestfitness	0,142	-0,032	0,086	0,010	1,000	0,028	-0,007
	Last Improvement	0,075	0,630	0,636	-0,492	0,028	1,000	-0,549
	Repulsors Found	-0,227	-0,622	-0,560	0,609	-0,007	-0,549	1,000

11.6. DETAILED ANALYSIS ADDITIONAL INFORMATION.

Detailed information not

11.6.1. Fittable_sin dataset.

Table 68 – Mean number of repulsors found fittable_sin.

Algorithm	O.F.T.		
	1.01	1.5	2.5
GP_REPULSORS_08	28.567	6.133	1.6
GP_REPULSORS_09	50.167	5.767	2.133

Table 69 – Individual size fittable_sin.

Algorithm	Classic	Repulsors 08			Repulsors 09		
Measure / O.F.T.	N.A.	1.01	1.5	2.5	1.01	1.5	2.5
Mean	210.8	231.134	225.267	192.2	221.067	216.667	253.2
Median	187	193	200	158	168	184	186

Table 70 – Individual tree depth fittable_sin.

Algorithm	Classic	Repulsors 08			Repulsors 09		
Measure / O.F.T.	N.A.	1.01	1.5	2.5	1.01	1.5	2.5
Mean	10	9.9	9.7	9.6	9.867	9.834	9.6
Median	10	10	10	10	10	10	10

11.6.2. Fittable_fn05 dataset.

Table 71 – Mean number of repulsors found Fittable_fn05.

Algorithm	O.F.T.		
	1.01	1.5	2.5
GP_REPULSORS_08	70.233	36.767	13.233
GP_REPULSORS_09	58.567	16.867	4.667

Table 72 – Individual size Fittable_fn05.

Algorithm	Classic	Repulsors 08			Repulsors 09		
Measure / O.F.T.	N.A.	1.01	1.5	2.5	1.01	1.5	2.5
Mean	81.933	25.533	45.333	59.533	48.6	40.4	52.933
Median	76	9	24	18	8	9	21

Table 73 – Individual tree depth Fittable_fn05.

Algorithm	Classic	Repulsors 08			Repulsors 09		
Measure / O.F.T.	N.A.	1.01	1.5	2.5	1.01	1.5	2.5
Mean	8.1	6.033	7.133	6.267	5.5	5.933	6.6
Median	10	4.5	8.5	6	4	5	6.5

11.6.3. Fittable_fn07 dataset.

Table 74 – Mean number of repulsors found Fittable_fn07.

Algorithm	O.F.T.		
	1.01	1.5	2.5
GP_REPULSORS_08	80.633	10.233	5
GP_REPULSORS_09	77.733	13.433	6.4

Table 75 – Individual size Fittable_fn07.

Algorithm	Classic	Repulsors 08			Repulsors 09		
Measure / O.F.T.	N.A.	1.01	1.5	2.5	1.01	1.5	2.5
Mean	62.8	55.667	39.533	96.133	37.4	21.267	74.733
Median	51	28	24	38	23	8	74

Table 76 – Individual tree depth Fittable_fn07.

Algorithm	Classic	Repulsors 08			Repulsors 09		
Measure / O.F.T.	N.A.	1.01	1.5	2.5	1.01	1.5	2.5
Mean	7.2	7.033	6.633	6.633	6.3	5.467	7.8
Median	10	8.5	9.5	10	6.5	3.5	10

11.6.4. Fittable_fn08 dataset.

Table 77 – Mean number of repulsors found Fittable_fn08.

Algorithm	O.F.T.		
	1.01	1.5	2.5
GP_REPULSORS_08	19.367	4.867	0.1
GP_REPULSORS_09	21.233	4.333	1.3

Table 78 – Individual size Fittable_fn08.

Algorithm	Classic	Repulsors 08			Repulsors 09		
Measure / O.F.T.	N.A.	1.01	1.5	2.5	1.01	1.5	2.5
Mean	88.667	149.867	113.933	119.067	127.533	154.533	122.667
Median	59	97	88	77	83	111	101

Table 79 – Individual tree depth Fittable_fn08.

Algorithm	Classic	Repulsors 08			Repulsors 09		
Measure / O.F.T.	N.A.	1.01	1.5	2.5	1.01	1.5	2.5
Mean	7.133	8.867	8	7.167	7.767	9.667	8.567
Median	8.5	10	10	9	10	10	10

11.6.5. CCPP folds dataset.

Table 80 – Mean number of repulsors found CCPP folds.

Algorithm	O.F.T.		
	1.01	1.5	2.5
GP_REPULSORS_08	58.567	9.733	1.3
GP_REPULSORS_09	82.1	6.733	0.733

Table 81 – Individual size CCPP folds.

Algorithm	Classic	Repulsors 08			Repulsors 09		
Measure / O.F.T.	N.A.	1.01	1.5	2.5	1.01	1.5	2.5
Mean	194.067	147.4	184.2	218.533	161.067	237.067	198.933
Median	178	141	191	206	121	186	195

Table 82 – Individual tree depth CCPP folds.

Algorithm	Classic	Repulsors 08			Repulsors 09		
Measure / O.F.T.	N.A.	1.01	1.5	2.5	1.01	1.5	2.5
Mean	10	9.9	10	10	9.733	10	9.767
Median	10	10	10	10	10	10	10

11.6.6. Concrete slump dataset.

Table 83 – Mean number of repulsors found Concrete slump.

Algorithm	O.F.T.		
	1.01	1.5	2.5
GP_REPULSORS_08	61.567	6.6	0.767
GP_REPULSORS_09	79.633	3.367	0.533

Table 84 – Individual size Concrete slump.

Algorithm	Classic	Repulsors 08			Repulsors 09		
Measure / O.F.T.	N.A.	1.01	1.5	2.5	1.01	1.5	2.5
Mean	138.667	117.067	150.533	155.333	148.4	167.133	160.8
Median	134	107	145	142	141	165	148

Table 85 – Individual tree depth Concrete slump.

Algorithm	Classic	Repulsors 08			Repulsors 09		
Measure / O.F.T.	N.A.	1.01	1.5	2.5	1.01	1.5	2.5
Mean	10	9.7	10	10	9.833	9.867	10
Median	10	10	10	10	10	10	10

11.6.7. Concrete compressive strength dataset.

Table 86 – Mean number of repulsors found Concrete compressive strength.

Algorithm	O.F.T.		
	1.01	1.5	2.5
GP_REPULSORS_08	53.433	0.967	0.4
GP_REPULSORS_09	72.667	0.6	0.467

Table 87 – Individual size Concrete compressive strength.

Algorithm	Classic	Repulsors 08			Repulsors 09		
Measure / O.F.T.	N.A.	1.01	1.5	2.5	1.01	1.5	2.5
Mean	174.4	108.8	160	202.067	134.267	176.067	175.267
Median	152	95	156	185	117	170	182

Table 88 – Individual tree depth Concrete compressive strength.

Algorithm	Classic	Repulsors 08			Repulsors 09		
Measure / O.F.T.	N.A.	1.01	1.5	2.5	1.01	1.5	2.5
Mean	10	10	10	10	9.933	9.933	10
Median	10	10	10	10	10	10	10

11.6.8. Wine quality red dataset.

Table 89 – Mean number of repulsors found Wine quality red.

Algorithm	O.F.T.		
	1.01	1.5	2.5
GP_REPULSORS_08	79.467	16.333	3.267
GP_REPULSORS_09	76.067	11.167	1.967

Table 90 – Individual size Wine quality red.

Algorithm	Classic	Repulsors 08			Repulsors 09		
Measure / O.F.T.	N.A.	1.01	1.5	2.5	1.01	1.5	2.5
Mean	163.8	48.067	109.2	146.733	29.8	108.4	170.2
Median	144	8	118	138	7	115	159

Table 91 – Individual tree depth Wine quality red.

Algorithm	Classic	Repulsors 08			Repulsors 09		
Measure / O.F.T.	N.A.	1.01	1.5	2.5	1.01	1.5	2.5
Mean	10	5.2	8.4	10	4.067	8.933	10
Median	10	4	10	10	3	10	10

11.6.9. LD50 Toxicity dataset.

Table 92 – Mean number of repulsors found LD50 Toxicity.

Algorithm	O.F.T.		
	1.01	1.5	2.5
GP_REPULSORS_08	45.044	6.933	0.967
GP_REPULSORS_09	55.033	2.633	2.1

Table 93 – Individual size LD50 Toxicity.

Algorithm	Classic	Repulsors 08			Repulsors 09		
Measure / O.F.T.	N.A.	1.01	1.5	2.5	1.01	1.5	2.5
Mean	275.067	48.067	109.2	146.733	29.8	108.4	170.2
Median	144	8	118	138	7	115	159

Table 94 – Individual tree depth LD50 Toxicity.

Algorithm	Classic	Repulsors 08			Repulsors 09		
Measure / O.F.T.	N.A.	1.01	1.5	2.5	1.01	1.5	2.5
Mean	9.4	9.967	10	10	8.667	8.067	9.433
Median	10	10	10	10	10	10	10

