Universiteit Leiden
The Netherlands

Opleiding Informatica

Investigating Grammatical Evolution's Ability to Reconstruct Mathematical Functions

L.J. Schreurs (s1987747)

Supervisors:
Hao Wang

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

August 15, 2022

Acknowledgements

I want to thank my supervisor, dr. Hao Wang, for providing me with his guidance and support during this year long process. Due to the outbreak of COVID-19, I had certainly lost all energy to finish my thesis, but dr. Wang was incredibly patient. Our weekly sessions, in which we discussed many topics throughout the year, were very helpful and made me gain a lot more understanding for Grammatical Evolution.

I also want to extend my thanks to my family and friends for their support. They helped me gain motivation to study, worked as my rubber ducky when I was stuck on a problem and proofread my thesis. Special thanks to Elise, for the very much needed (mental) support, and to Jan, who certainly gave me some very helpful advice for the final stretch of the project.

Abstract

Grammatical Evolution (GE) is a Genetic Programming technique, that evolves a population of individuals, each with its own genotype. The genotype is evolved into a phenotype, the expression of its genes, making use of a supplied context-free grammar to restrict the search space and a genotype-to-phenotype mapping process using this grammar. The phenotype is then evaluated on its performance. Although GE has been succesfully applied in various scenarios, its performance of accurately learning the underlying expressions is not often questioned. This thesis examines the ability of GE to accurately capture the original form, the complexity, and the feasibility rate of the created expressions, using datasets for four different problems from the BBOB set: the Ellipsoidal, Rastrigin, Rotated Ellipsoidal, and Weierstrass problems. The results show that, using its current supplied parameters, GE is not capable of accurately capturing the original form of the underlying expressions very well.

Contents

1 Introduction

Evolutionary algorithms (EA) are a set of heuristic optimisation algorithms that use evolutionary processes similar to those found in nature. After initialising an initial population, it will determine the fitness of all individuals in a population, then apply operators such as crossover and mutation. After the fitness is evaluated again, this process will keep repeating over a number of generations. The objective of these algorithms is to find a solution that, although it may not be optimal, comes close to the correct solution in a reasonable amount of time. The advantages of EAs include their flexibility and the limited amount of time needed to find a satisfactory solution [1].

In this study we focus on an EA called Grammatical Evolution (GE) [2], a Genetic Programming (GP) [3] approach. The GP approach, and subsequently GE, operates on the structure of programs and functions. Where GE differs from GP is that it operates with the help of a supplied Context-free Grammar (CFG) in Backus-Naur Form (BNF), which is used to map a genotype, e.g., a list of integers, to a phenotype, a derivation tree. The fitness of the individual is decided by evaluating this phenotype. We can use GE to evolve a program for solving a specific task.

1.1 Research Question

Despite the successful application of GE systems in various scenarios [4, 5], its performance of accurately learning the underlying expressions is seldom questioned. Particularly, when solving the well-known symbolic regression problems, we are interested in its capability of reconstructing the target mathematical form of those problems. This thesis aims to answer this question using a set of widely-applied numerical benchmarking functions, namely the Black-Box Optimization Benchmarking (BBOB) problems [6]. In detail, we will investigate the following research questions:

RQ-1 Does the landscape (e.g., the contour lines) of the final expression from GE resemble the one of the target on a global scale?

RQ-2 Can a GE system capture the ruggedness in a high multi-modal problem given a limited number of points sampled thereon?

RQ-3 What is the complexity of the final solution produced by a GE system, e.g., the maximal depths of expression trees?

RQ-4 Given no prior knowledge of the set of input variables taken by a target problem, how likely is a GE system to identify the correct set of input variables?

1.2 Thesis overview

The subject of the thesis and the research questions are introduced in this section. Section 2 aims to provide a bird's-eye overview of GE and includes a short introduction on PonyGE2 [7], the software used to perform the experiments. Section 3 discusses which problems were used for benchmarks, the manner in which the dataset was generated, and the rationale behind the choices for the evolutionary parameters. Section 4 contains the experimental setup and the experiments. Section 5 contains the results for the experiments. Finally, Section 6 draws conclusions on the performance of

GE based on experiments results and discusses possible further research.

In addition to these sections, an Appendix has been added which includes a short list of acronyms and terms used throughout the text, the parameters and grammar used in the experiments, and the rest of the results for Section 5 as to reduce clutter.

2 Background

To gain a better understanding of the experiments that are carried out in Section 4, it is important to have at least some knowledge of how Grammatical Evolution (GE) works. This section aims to introduce GE from a high-level perspective. It also includes a short introduction to PonyGE2, an implementation of GE.

2.1 Grammatical Evolution

Grammatical Evolution [2] is a grammar-based variation of Genetic Programming (GP) [3]. GE can either make use of a Context-free Grammar (CFG) in Backus-Naur Form (BNF) form and a genotype-to-phenotype mapping to generate solutions or operate directly on the derivation tree, like in traditional GP. The grammar used in GE can be represented by the tuple $\{N, T, P, S\}$, where:

N is the set of non-terminals (e.g., $\langle exp \rangle$ and $\langle int \rangle$), which can be expanded into terminals or non-terminals

T is the set of terminals (e.g., x or $0 - 9$), which cannot be expanded further

P is a set of production rules (see Figure 1)

S is the start symbol and also a member of N ($\langle exp \rangle$ in the example grammar)

The genotype used to map a BNF grammar to a phenotype is a set of codons (`c`) in a linear representation (e.g., a list of integers or binary values). The value of each codon is mapped to a production rule (`r`) in a left-first order (pre-order traversal), although other types of mappers have been explored as well [8]. Exchanging codons between two individuals somewhere in the genotype can have a 'ripple effect' [9, 10], due to all codons to the right of the ripple site having to be reinterpreted. This change in the value of the codon can cause the mapping process to choose a different production rule subtree to follow, changing the form of the derivation tree being generated.

The phenotype of an individual is expressed as a derivation tree. This phenotype is evaluated to determine the fitness of an individual.

The mapping function used in the genotype-to-phenotype mapping process uses the *modulus* rule:

```
Rule = c % r
```

This rule states that the value of the codon (`c`) being read is divided by the number of rules in the current production rule (`r`). The remainder of this division will then be the rule (`Rule`) chosen to

$$
\begin{aligned}
\langle exp \rangle &::= \langle exp \rangle + \langle exp \rangle \\
&\mid \langle exp \rangle \text{ - } \langle exp \rangle \\
&\mid \langle exp \rangle * \langle exp \rangle \\
&\mid \langle exp \rangle \ / \ \langle exp \rangle \\
&\mid \langle int \rangle \\
&\mid \langle var \rangle \\
\langle int \rangle &::= 0 \mid \ldots \mid 9 \\
\langle var \rangle &::= x_0 \mid \ldots \mid x_n
\end{aligned}
$$

Figure 1: A simple BNF grammar.

be expanded further.

In the case of our simple grammar from Figure 1, there are 6 options `r` when considering the first production rule ($\langle exp \rangle ::= \ldots$). Taking a codon integer value of `7`, the value of `Rule` will be `7 % 6` or a remainder of `(1)` ($\langle exp \rangle - \langle exp \rangle$). This process will be repeated until either the set maximum depth or a terminal symbol in one of the branches of the derivation tree is reached. An example of this mapping process can be found in Figure 2.

Just like in GP, it is possible to directly manipulate the derivation tree (see [7], p. 7 (Derivation Tree Initialisation)). Direct manipulation of the derivation tree allows for more fine-tuned control when building the tree, such as limiting the depth to which a tree can grow.

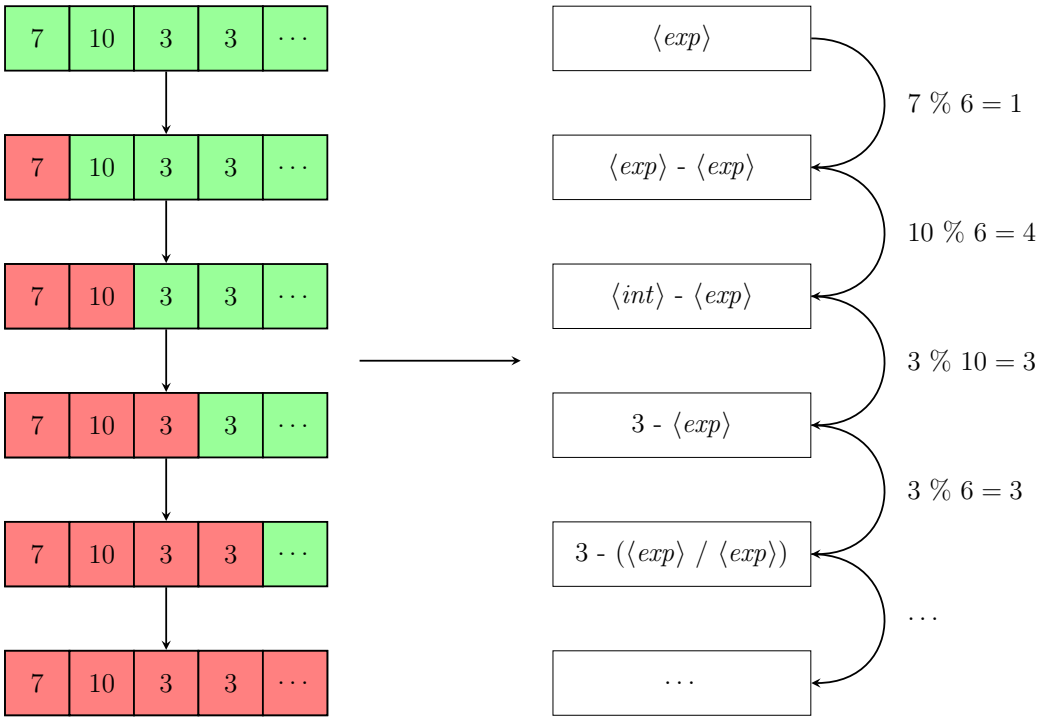


Figure 2: An example of the genotype-to-phenotype mapping using the example grammar. The genotype-to-phenotype mapping process works in a preorder fashion.

Grammatical Evolution uses processes similar to those found in nature to train a model. After the initialisation process, a population of individuals will evolve over a set amount of generations, according to evolution rules set in the parameters. These parameters determine not only how a population is initialised, the size of the population, or the number of generations, but will also determine the techniques used for selection, crossover and mutation during the evolution process. More on the choices for the parameters can be found in Section 3.

2.2 PonyGE2

There exist multiple implementations of GE in different languages. The experiments in this paper have been performed using PonyGE2 [7], an open-source Python-based implementation of GE.

PonyGE2 implements different options for the evolutionary operators commonly used in GE, such as for initialisation, mutation and crossover. All of the options available can be found in the PonyGE2 wiki[1].

PonyGE2 was chosen as our GE implementation of choice for its ease of use, in no small part due to its extensive documentation, implementation language, and its ease of extensibility of the code. This choice is supported by [11], who compared two different implementations of GE.

3 Methodology

When using Grammatical Evolution, it is important to take into account factors that can have an effect on its performance during the evolution process and its ability to generate effective solutions. These factors not only include hyper-parameters, such as the type of initialisation used and evolutionary parameters, but also include the maximum depth the derivation tree is allowed to reach and the structure of the grammar.

This section aims to explain these factors and how they have been taken into account during the design of the experiments.

3.1 Benchmark problems

The datasets used in the experiments were generated based on four functions defined in the Black-Box Optimisation Benchmarking (BBOB) Test Suite [6]:

Ellipsoidal (f_2): $\sum_{i=1}^{D}(10^6 \frac{i-1}{D-1} x_i^2) + f_{\text{opt}}$

Rastrigin (f_3): $10\left(D - \sum_{i=1}^{D}\cos(2\pi x_i)\right) + ||x||^2 + f_{\text{opt}}$

Rotated Ellipsoidal (f_{10}): $\sum_{i=1}^{D}(10^6 \frac{i-1}{D-1} x_i^2) + f_{\text{opt}}$

[1] https://github.com/PonyGE/PonyGE2/wiki

Weierstrass (f_{16}): $10\left(\frac{1}{D}\sum_{i=1}^{D}\sum_{k=0}^{11}1/2^k\cos(2\pi 3^k(x_i+1/2))-f_0\right)^3+f_{\text{opt}}$

We chose these functions based on the following considerations:

(1) these functions are widely studied and well-understood in the optimisation community;

(2) the form of these multimodal functions poses a major learning challenge for GE, due to its local fluctuations on their landscape.

In these functions, D represents the dimensionality of the function, x_i represents one of the input variable and f_{opt} represents a constant. This constant can be optimised by PonyGE2[2].

For each function, a separate training and test set were constructed using a simple loop in a small custom script and IOHExperimenter [12]. IOHexperimenter implements the functions defined in the BBOB Test Suite.

3.2 Dataset

Our data set consists of 50 000 points sampled uniform at random (u.a.r.) from the function domain $[-5,5]^2$, where the 2 is the *dimensionality*, using the zero-instance of the BBOB function. The meaning of problem instances can be found in [13]. We chose to test the GE system on the two-dimensional problem due to the following considerations:

(1) we can easily visualize the landscape (e.g., by means of contour plots) of the target and the learned functions, which provides an understanding of how GE performs on those BBOB function (which are usually deemed very difficult to optimise);

(2) The resulting expression of the function can be comprehended by the user. We evaluated each point in the data set with one of the selected objective function and the resulting function values were normalised to the unit interval, which will make the error metric more easily understandable to humans when interpreting the results. During evaluation of the performance, we took into account that using a finite number of samples, the extent to which a function is properly represented could be different to the original function.

After data generation, the dataset was split into a train and test set with a ratio of 80:20[3]. This ratio is considered standard for datasets in Machine Learning (ML). When evaluating the performance of the model on our test data, the metric used was the mean squared error (MSE).

3.3 Hyper-parameters

PonyGE2 has a list of hyperparameters that can be set by the user. These hyperparameters configure how the algorithm will behave during the learning process. An example of a hyperparameter is iniatialisation. Choosing what type of initialisation is used has an influence on the performance

[2] https://github.com/PonyGE/PonyGE2/wiki/Example-Problems#regression#optimisation

[3] 40 000 : 10 000 in our train and test set

of GE. Choices made for these parameters are explained in this section. A full list of parameter settings for all benchmark problems can be found in Appendix C.

3.3.1 Initialisation

Position Independent (PI) Grow [14] is a form of position independent initialisation. PI Grow will randomly choose a branch of the derivation tree to expand, instead of a left-first initialisation (of the derivation tree), and will grow at least one branch to the depth limit set by the user. The type of initialisation that directly controls the derivation tree is called 'Derivation Tree Initialisation' [7] and assures a larger amount of distinct derivation trees (read: larger genetic diversity) in the initial population.

We chose PI Grow for our experiments among all implemented initialisation methods in PonyGE2[4], due to its capability of initialising a diverse population and wide usability in the GE community. Although recent works exploring alternatives, e.g. [15], have reported excellent results on some test problems, we do not intend to implement and investigate each of these alternatives. The major research question here is not to find the most optimal hyperparameters, but to figure out GE's overall performance for the types of problems in our benchmark set.

3.3.2 Depth of the derivation tree

Both the maximum initialisation depth and maximum tree depth[5] are set to allow generated functions to grow deeper than the depth of the derivation tree of the original benchmark problems.

The maximum initialisation tree depth of 10 and the maximum tree depth of 17 were chosen to allow generated functions to not be limited by the depth of the original functions, but also to put a limit on the complexity of the inferred functions. An example of a derivation tree for the Ellipsoidal (f2) function, with its depth denoted by the numbers on the right side of the figure, can be found in Figure 3.

3.3.3 Mutation, Crossover and Selection

Since there is no intention of hyper-parameter tuning, we took the default setting of the mutation, crossover, and selection parameters in PonyGE2.

PonyGE2 uses the following options for these parameters:

Mutation Integer flip per codon

Crossover Variable onepoint crossover with a probability of 75%.

Selection Tournament selection with a tournament size of two

[4]https://github.com/PonyGE/PonyGE2/wiki/Initialisation

[5]The parameters for these values are `[MAX_INIT_TREE_DEPTH]` and `[MAX_TREE_DEPTH]` respectively.

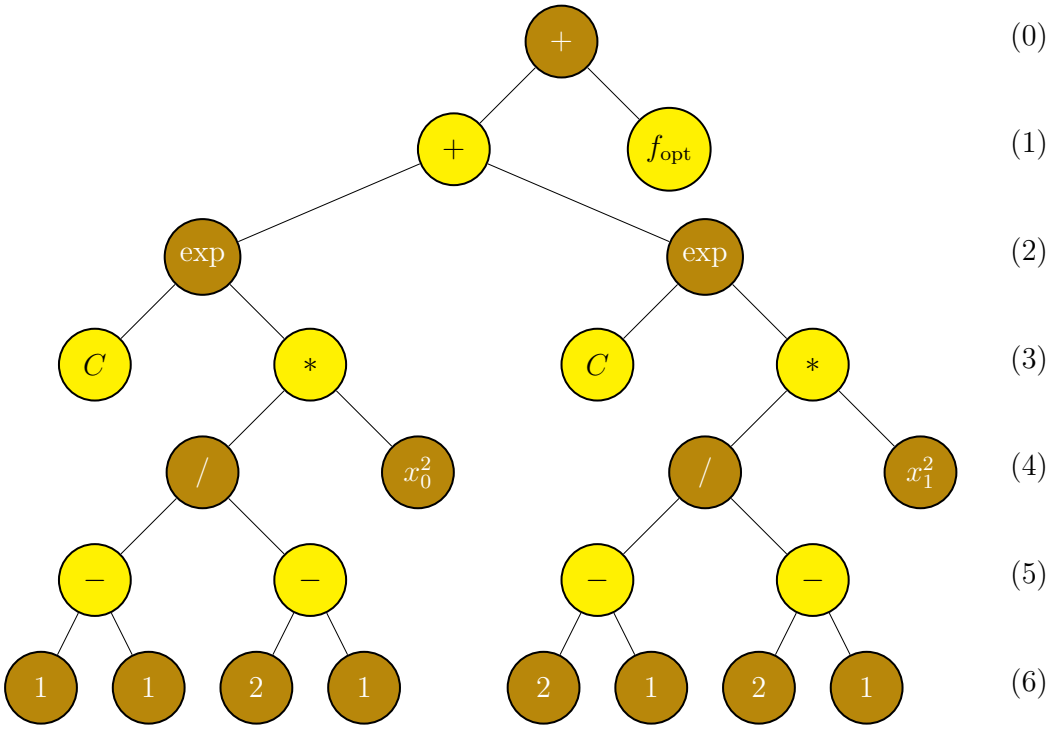


Figure 3: The derivation tree for the Ellipsoidal function (f_2) in the BBOB Test Suite with $D = 2$.

3.4 Context-free Grammar

The grammar used in GE is a context-free grammar (CFG) expressed in the Backus-Naur Form (BNF). In our experimentation, we used a simple recursive structure of the grammar to prevent possible bias for (non-)terminals when adding any additional symbols to the grammar (e.g., nth-root). This grammar covers the target mathematical forms provided in Section 3.1, without further optimisation of the grammar itself.

Since our grammar features a 10:4 split between non-terminal and terminal symbols, it is also called an *explosive* grammar [9, 16]. According to [16], the definition of an explosive grammar is "a grammar where, if there is a choice between non-terminals and terminal, a non-terminal expansion is more likely to be chosen". This explosiveness makes the derivation trees more likely to grow larger, not terminating until all codons in the genotype have been read or until the maximum depth has been reached, hence realising more complex candidate solutions.

3.5 Feasibility of derivation trees

In each experiment, we recorded the feasibility of the derivation trees in each run of PonyGE2. Here, by feasibility, we mean: "the candidate solution takes on the same set of non-constant terminal symbols as the target function".

Although we did measure the feasibility of all the formulas during the run, we did not take feasibility into account when evaluating the fitness of a generation of individuals due to the following considerations: (1) the final solution was generated without knowledge of the function

used to generate the function values in the dataset and is based on inference. GE can make well-performing models, with a low *MSE*, using only a single input variable; (2) Giving these 'invalid' individuals a low fitness during the evaluation process, would constrict the search space. An 'invalid' individual can evolve and in a matter of a few generations a 'valid' individual could be found again with an even higher fitness than those individuals evolved from 'valid' individuals.

4 Experiments

In order to give an answer to our research questions described in Section 1.1, we designed three experiments. Through these experiments, we aimed to gain a better understanding of the ability of GE to accurately reconstruct the target mathematical forms of our symbolic regression problems, which can be found in Section 3.1.

This section is split up into the following subsections:

(1) a visual comparison of contour plots, comparing the original BBOB function to the BEST, MEDIAN and WORST performing 'best' individuals inferred over 20 runs.

(2) a comparison of the mean of the average depths of each benchmark problem.

(3) a section on the feasible formulas found with GE, separated into:

  (i) a comparison of ratio of feasible to non-feasible solutions between inferred individuals for our four benchmark problems.

  (ii) a visual comparison of contour plots, comparing the original BBOB function to the BEST, MEDIAN and WORST performing 'best' *feasible* individuals inferred over 20 runs.

4.1 Experimental set-up

PonyGE2 facilitates easy data gathering and experimental setup through a simple experiments manager. This experiment manager can be used to start n runs at the same time through a setting in the parameter file [6]. Every run of PonyGE2 produces a list of stats in a tab-seperated values (TSV) format and a separate .txt file containing the best performing individual in that run. These best performing individuals in each run have been ranked from best to worst performance for the first and third experiment.

All plots in Section 5 were generated using `matplotlib` [17]. All contour plots were generated with values in the range $[-5, 5]$ for both x_0 and x_1, using a delta of 0.002 between each value in this range. All data for experiment 2 (Section 4.3) and experiment 3 (Section 4.4) was gathered using `pandas` [18].

[6] https://github.com/PonyGE/PonyGE2/wiki/Scripts#basic-experiment-manager

Parameter	Value
Number of experiment runs	20
Generations	100
Population	500
Initialisation	PI Grow
Max initialisation depth	10
Max tree depth	17

Table 1: List of parameters relevant for the experiments.

4.2 Experiment 1: Visual comparison

Through the first experiment, we attempted to gain insight into the ability of GE to infer the form of our four different benchmark problems. This was achieved by doing a visual comparison between the contour plots of the original BBOB functions with the BEST (rank 1), MEDIAN (rank 10) and WORST (rank 20) performing functions inferred by GE.

The contour plots provide a 2D representation of a 3D surface through lines joining points with an equal elevation (z-value). The ranking of the functions was done by taking the best solutions for each run, which are saved to a separate file by PonyGE2, and ordering them from lowest to highest mean squared error (MSE). In this ranking, feasibility was not taken into account.

The consideration for plotting these three solutions was as follows:

1. the MEDIAN performing individual gives the least biased view of the overall ability of GE to predict our target functions;

2. the WORST and BEST performing individuals can be used to gauge the minimum and maximum performance of individuals made by GE. The worst performing was especially important, because knowing how an algorithm performs at minimum was useful to gauge what performance can be expected for the other runs.

4.3 Experiment 2: Solution Complexity

The purpose of the second experiment was to determine the complexity of solutions inferred by GE for each benchmark problem. This was achieved by measuring the mean of the average depth per generation over 20 runs for each problem. Each non-terminal has at least one child node, but can have more depending on the type of mathematical operation. Each additional level of nodes adds another layer of complexity to the function.

The average depth per generation was read from the `stats.tsv` file in each respective run's results folder. The mean of the stats over 20 runs was then taken using the `pandas.Dataframe.mean` function.

4.4 Experiment 3: Feasibility of solutions

The third experiment consists of two parts:

(i) a comparison between the ratio of feasible to non-feasible solutions generated by GE for each benchmark problem.

(ii) a visual comparison between contour plots, similar to those in Experiment 1 (Section 4.2).

The purpose of part (i) was to see the how likely GE was to identify the number of input variables of the target problem without any prior knowledge of this problem. This was achieved by taking the mean of the feasibility of each generation over 20 runs. The initialisation step (generation 0) was not taken into account for this test.

The purpose of part (ii) was to see if, when comparing feasible solutions to each other, they were better at predicting the form of the problems than when feasibility was not taken into account. Again, the BEST, MEDIAN and WORST solutions were taken, choosing from the pool of *feasible* solutions found in 20 runs. The number of feasible solutions found for each benchmark problem can be found in Table 3.

5 Results

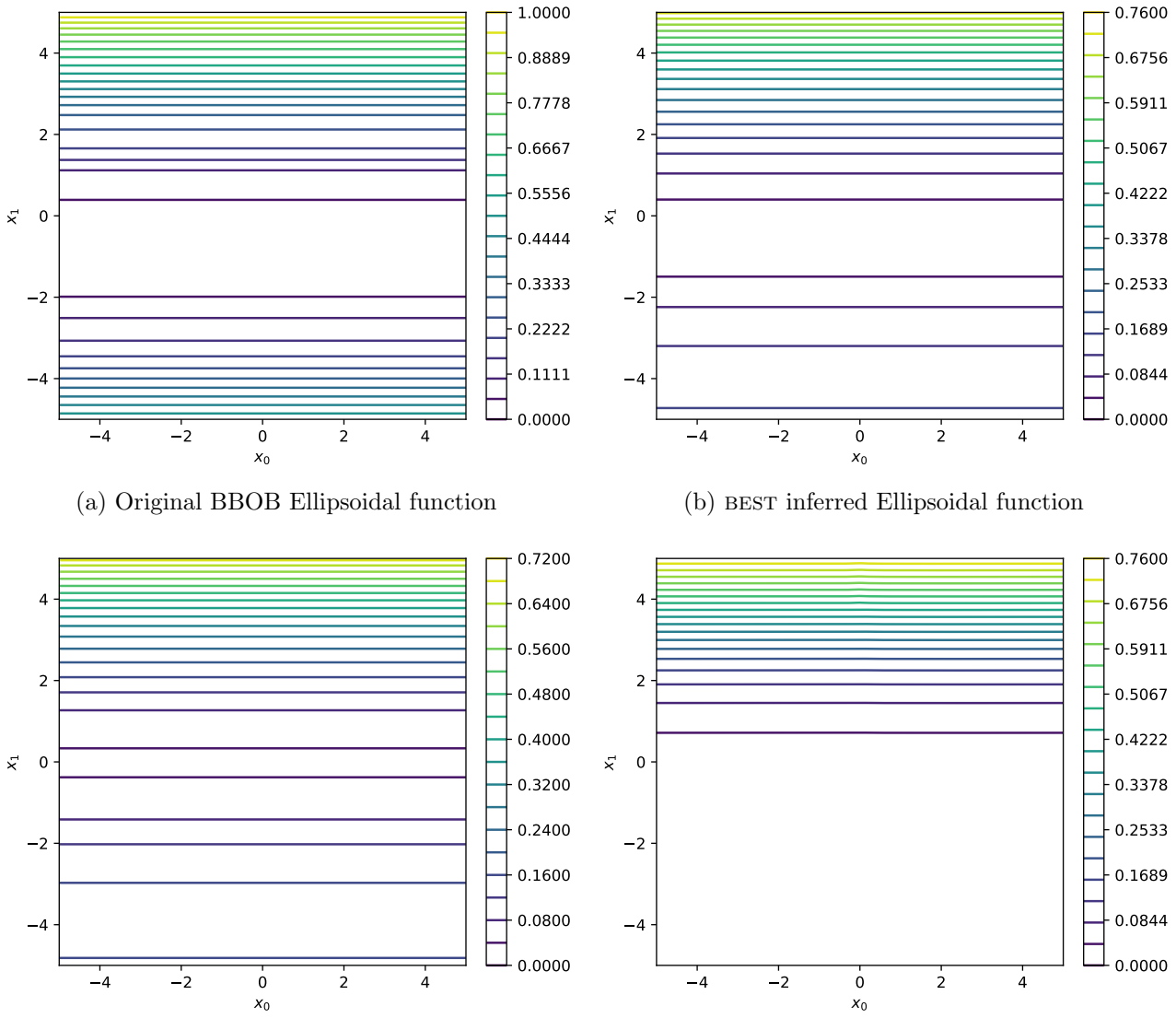
5.1 Experiment 1: Visual comparison

Ellipsoidal (see Figure 4) Although the contour lines in the upper part of each plot appears similar to those of the original function, the WORST individual misses the lower part of the landscape since the upper part has relatively higher function values, which seem to dominate the loss function. The BEST and MEDIAN individuals manages to capture the lower parts, but the range of the function value is significantly smaller than that of the original BBOB function, going from around 0.000 to around 0.2500 in the BEST and MEDIAN individuals and 0.000 to around 0.6700 in the original BBOB function between -5 and -2 on the x_1 axis.

Rastrigin (see Figure 8) The colour bar for the plot of the WORST individual shows a maximum height almost double that of the original BBOB function. Furthermore, this individual focuses too much on the upper part of the landscape, losing all the detail in the region under 4 on the x_1 axis.

The plots for the BEST and MEDIAN individuals show a form more akin to the plot of the original BBOB function, although they both do not reach the same maximum height as the original BBOB function. Of these plots, the plot of the MEDIAN individual has a more uniform shape and has seemingly captured the form of the original function better than the BEST individual, in which the higher function values dominate the loss function.

Rotated Ellipsoidal (see Figure 9) We have observed that the maximum height for the learned functions is between half to almost three times smaller than that of the original function. Furthermore, the rotation of the landscape is not learned in the WORST and MEDIAN functions. A rotation is learned in the BEST individual, which shows a discontinuity of the gradient at $x_0 = 0$.



(a) Original BBOB Ellipsoidal function

(b) BEST inferred Ellipsoidal function

(c) MEDIAN inferred Ellipsoidal function

(d) WORST inferred Ellipsoidal function

Figure 4: The contour plot of the original Ellipsoidal BBOB function can be compared to the BEST, MEDIAN, and WORST feasible functions inferred by PonyGE2 over 20 runs.

Although the maximum height is lower and rotation is missing in the plot of the MEDIAN individual, the overall form of this individual looks the most like plot of the original BBOB function. Both the BEST and WORST individuals have learned more contour changes in the upper part of the plot, which are not there in the plot of the original BBOB function.

Weierstrass (see Figure 10) The colour bars of all three learned functions show that both the minimum and maximum height of the original BBOB function have not properly been learned. Although a ridge shape slightly akin to that of the original BBOB function can be seen in parts of the plot of the BEST and WORST individual, the plot of the WORST individual seems to be a plateau.

5.2 Experiment 2: Solution complexity

The goal of Experiment 2 was to determine the complexity of solutions inferred by GE for each benchmark problem by measuring the mean of the average depth per generation over 20 runs for each problem.

Function name	*Mean depth*	*Standard dev.*
Ellipsoidal (f_2)	11.078	0.599
Rastrigin (f_3)	11.054	0.477
Rotated Ellipsoidal (f_{10})	11.130	0.508
Weierstrass (f_{16})	11.185	0.566

Table 2: The mean depth and the standard deviation over 20 runs of the average depth at generation 100.

The results of this experiment, which can be found in Figure 5, show that the initialisation is consistent for each function. Each function shows small differences between growth rate and standard deviation, but the complexity seems to grow at roughly the same rate per generation. In the last generation, the mean depth is comparable between all problems (see Table 2).

5.3 Experiment 3: Feasibility

5.3.1 Part I

The ratio of feasible functions in a population of 500 individuals grew at a slow pace for the Ellipsoidal (Figure 6(a)), Rastrigin (Figure 6(b)), and Rotated Ellipsoidal (Figure 6(c)). The ratio for the Ellipsoidal function experienced larger growth in the first 40 generations and then leveled out, while the ratio of feasible functions for the Rastrigin and Rotated Ellipsoidal function kept growing between the 0th and 100th generation.

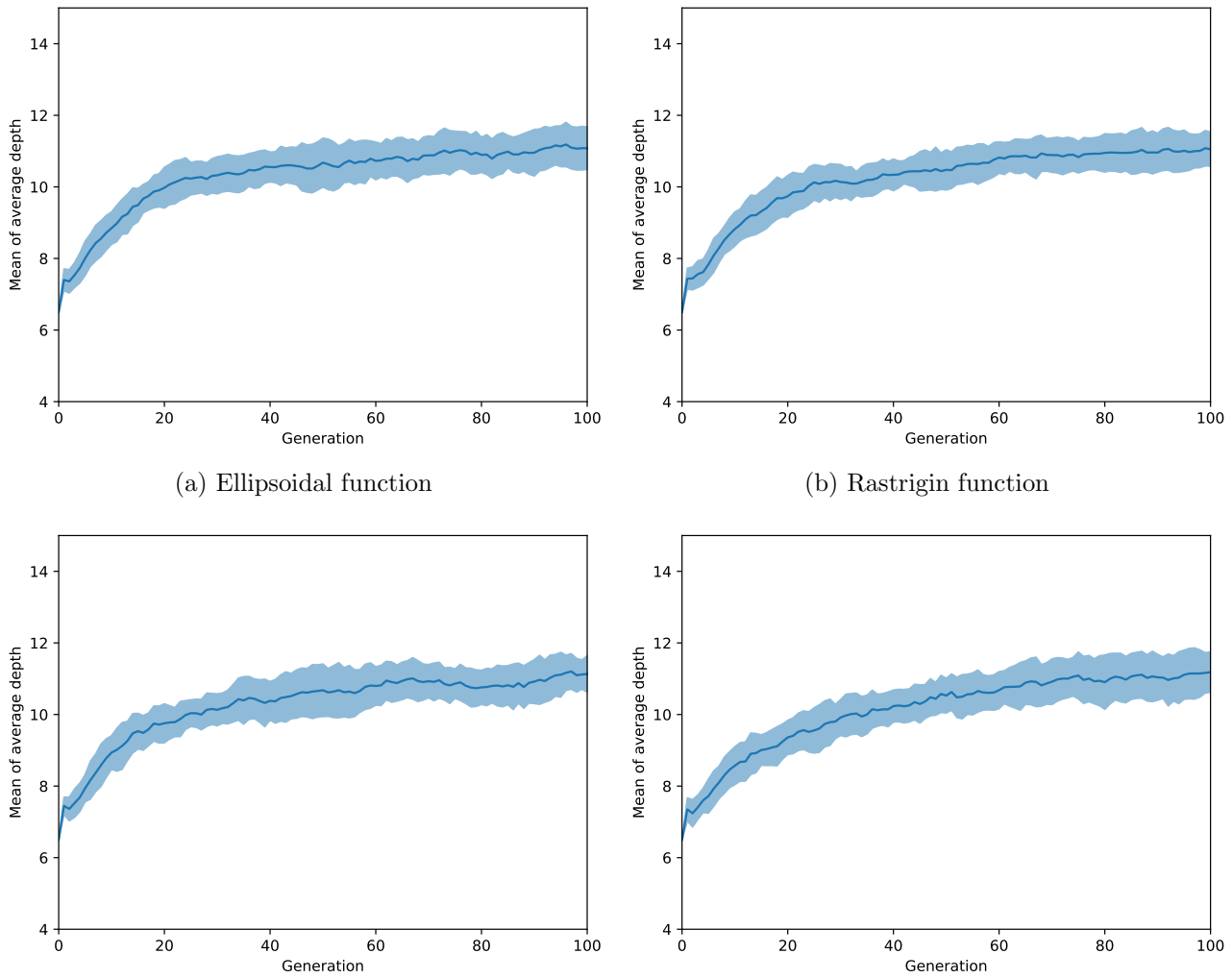
However, as can be observed in the figure, the ratio of feasible functions for the Weierstrass (Figure 6(d)) function grew slightly in the first 10 generations and then decreased thereafter.

Function name	*Nr. feasible best solutions*	*% feasible solutions*	*Standard dev.*
Ellipsoidal (f_2)	6	43.97	8.27
Rastrigin (f_3)	17	44.72	10.97
Rotated Ellipsoidal (f_{10})	14	47.13	6.34
Weierstrass (f_{16})	11	31.04	12.83

Table 3: The number of feasible best solutions and the mean percentage and standard deviation of feasible formulas for each function at generation 100.

5.3.2 Part II

Ellipsoidal (see Figure 7) The colour bar shows that the maximum height for the learned functions is around 25% smaller than that of the original BBOB function.



(a) Ellipsoidal function

(b) Rastrigin function

(c) Rotated Ellipsoidal function

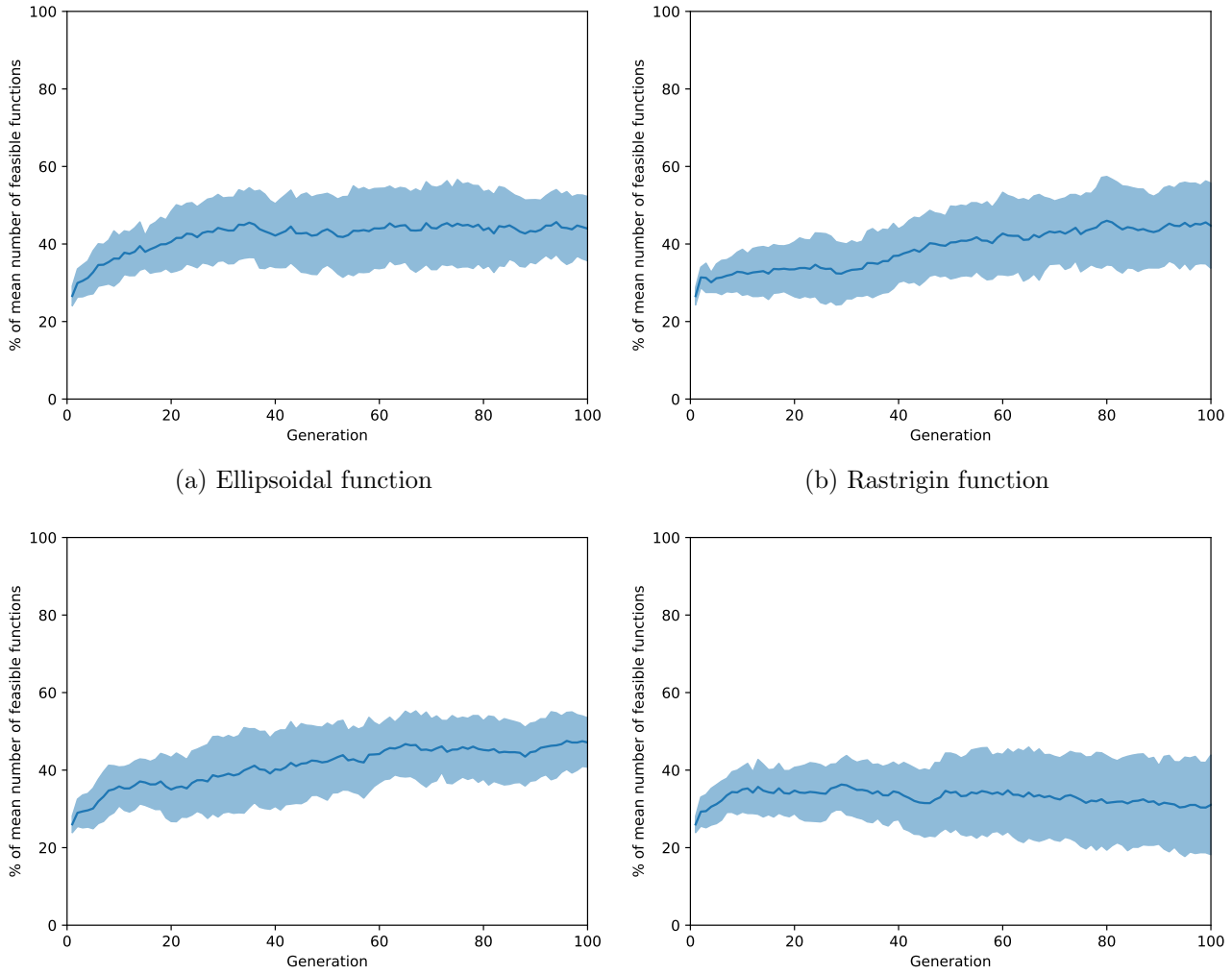
(d) Weierstrass function

Figure 5: The mean over 20 runs of the average depth per generation of a run. Through the depth, the complexity of a solution can be measured.

Furthermore, although the contour lines in the upper part of each plot for the inferred functions look similar to the original BBOB function, the lower part of the landscape in the BEST and MEDIAN does have changes in elevation, but not as large as those in the original function, with the MEDIAN individual having a slight rotation in the lower part. The WORST individual has relatively higher function values, which seem to dominate the loss function.

Rastrigin (see Figure 11) These feasible functions are the same as those described in Section 5.1.

Rotated Ellipsoidal (see Figure 12) The colour bars shows that the maximum height for the learned functions is around twice as small as the maximum height of the original BBOB function. The BEST individual is the same one as the one described in Section 5.1, possessing an increasingly large negative slope in the range -5 to 0 on the x_0 axis, with a sudden change at 0 to a smaller negative slope.



(a) Ellipsoidal function

(b) Rastrigin function

(c) Rotated Ellipsoidal function

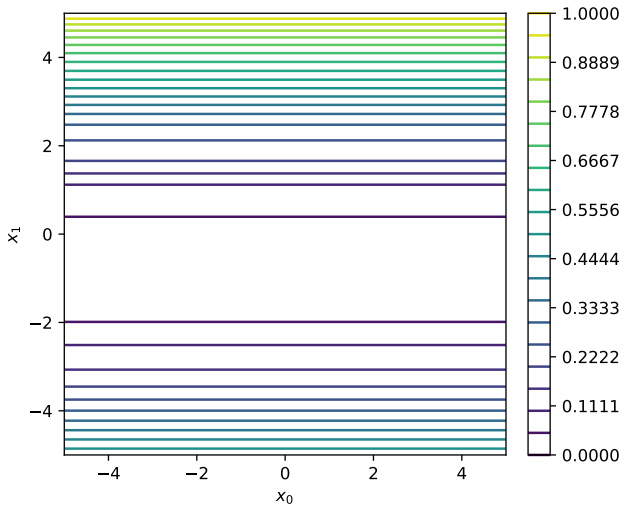
(d) Weierstrass function

Figure 6: The percentage of the mean number of feasible functions per generation over 20 runs of GE.

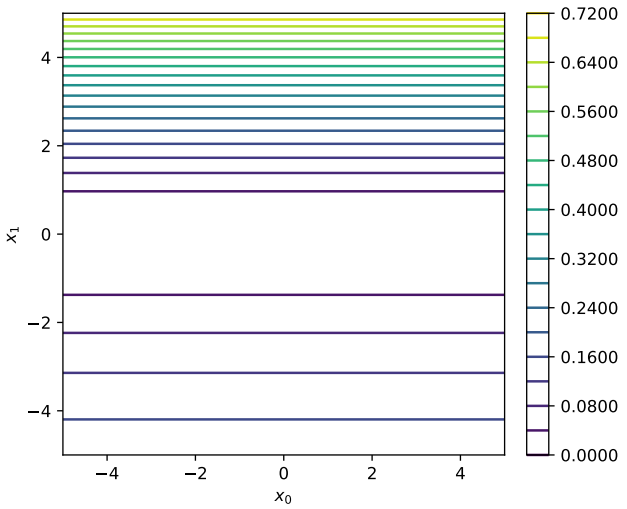
The MEDIAN individual looks more like the original BBOB function, but does not have any rotation. The contour lines in the lower parts of the landscape also have the shape of a wave, of which the amplitude seems to increase the further down on the landscape they are positioned.

The WORST individual has focused on the higher part of the landscape, where no large elevation are present in the original BBOB function. There is also a negative dip in the lower parts of the landscape with its deepest point as 0 on the x_0 axis, which increases in depth the lower the value of x_1 is.

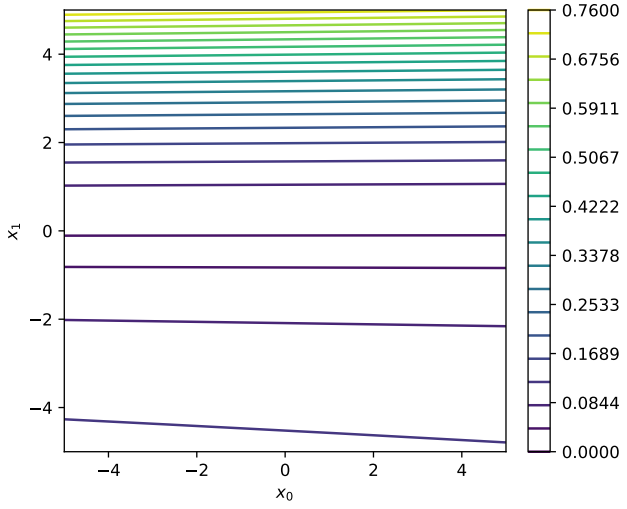
Weierstrass (see Figure 13) The colour bars of the three learned functions show that the minimum and maximum height of the original BBOB function have not properly been learned. Furthermore, the only individual that looks slightly similar to the original BBOB function is the BEST individual, having a slight rotation just above 0 on the x_1 axis.



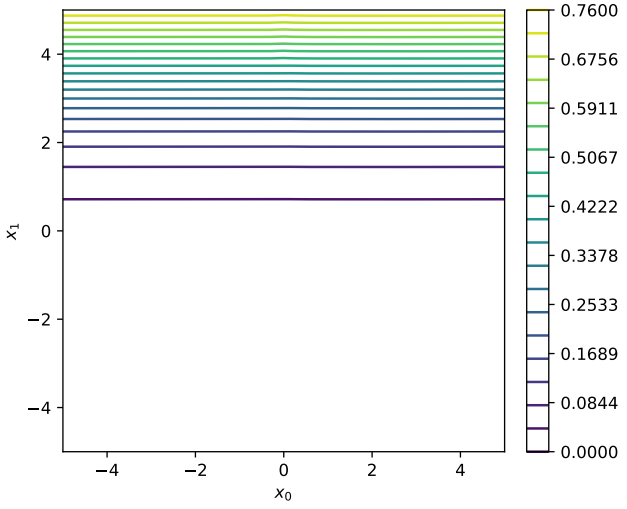
(a) Original BBOB Ellipsoidal function



(b) BEST inferred feasible Ellipsoidal function



(c) MEDIAN inferred feasible Ellipsoidal function



(d) WORST inferred feasible Ellipsoidal function

Figure 7: The contour plot of the original Ellipsoidal BBOB function can be compared to the BEST, MEDIAN and WORST **feasible** function inferred by PonyGE2 over 20 runs. Not all of the 20 functions generated by GE per run are feasible.

6 Conclusions and Future Research

6.1 Overview

In this work, we investigated the capability of GE to accurately reconstruct the target mathematical forms of our well-known symbolic regression problems. To this end, we designed three experiments to answer our four research questions proposed in Section 1.

Answer to RQ-1 First, we performed an experiment that compared a contour plot of the BEST, MEDIAN, and WORST performing individuals to a contour plot of the original BBOB function, to see if the landscape of the learned functions resembled those of the original. We observed that the same height as the original BBOB functions were not learned well for any function. Many of the learned functions seemed to have an overall shape similar to that of the original functions, but they missed a lot of the details (i.e., local landscapes) in the plots of the original functions. Many of the learned functions also seemed to have favoured only certain parts of the function in their plots. One of the best examples of this can be found in Figure 8(d).

Repeating this test in Experiment 3, comparing only feasible formulas to the original BBOB function, no improvement in similarities between the learned functions and the original BBOB function could be observed.

Answer to RQ-2 The first experiment was also used to gauge if GE could capture the ruggedness of a high multi-modal problem given a limited number of points sampled thereon. We observed that, especially for the Rastrigin and Weierstrass, GE was not able to correctly capture the ruggedness of the original function. GE was able to capture the form of the Ellipsoidal function and, although the rotation was not learned, it did manage to learn the overall form of the Rotated Ellipsoidal function.

Although the overall shape of the original Rastrigin function could be seen in the BEST and MEDIAN individuals, a lot of the smaller peaks and valleys found in the plot of the original function were not present in the plots of those learned functions.

GE was also not successful in learning the ruggedness of the Weierstrass function. Due to the rugged nature of this function, the learned functions were not successful in finding both minimum and maximum points of the original function.

Answer to RQ-3 We performed an experiment (Experiment 2) to determine the complexity of a formula by measuring the mean of the average depth of the derivation trees. Our observation was that no discernible difference could be found between the mean average depths for each problem, which were all around the same value (see Table 2).

Answer to RQ-4 Finally, we investigated how likely a GE system identifies the correct set of input variables. According to our findings, the empirical probability, taking into account the standard deviation, ranged from around 20% at the least to around 50% at the most, depending on the function that was learned.

To summarize the answers to our research questions, using our current setup, GE is not capable of accurately learning the underlying expression, in this case the mathematical form of well-known regression problems, very well.

6.2 Future work

Although our findings seem to suggest that GE is not capable of accurately learning the underlying expression very well, there are a number of ideas to be explored to improve the performance of GE in this regard. More research into limiting the complexity of expressions even further could also be done, to see if there is any significant drop in performance.

To limit the complexity of a function, there are a few options to consider. Firstly, a complexity value could be assigned to symbols in the grammar, allowing the user to select operators they want to take priority when building the tree. In our grammar, this could mean giving more complex operators, such as cos, sin, or tan, a smaller priority or bias.

Another direction to take is to research the effect of optimising the grammar design in our GE system. As described in [9], there are "specific design principles that can be applied when attempting to solve any problem using GE, which do not require domain knowledge". Applying these design principles could lead to improved performance for GE on our benchmark problems.

Finally, [19] proposes a way to optimise the hyper-parameters of a GE system, using an Efficient Global Optimization (EGO) algorithm. According to this study, the performance of GE in regression problems is significantly improved using this algorithm. This study also suggests doing more research into including the population size of GE in its tuning process.

References

[1] David B. Fogel. "Practical advantages of evolutionary computation". In: *Applications of Soft Computing*. Ed. by Bruno Bosacchi, James C. Bezdek, and David B. Fogel. Vol. 3165. International Society for Optics and Photonics. SPIE, 1997, pp. 14–22. DOI: 10.1117/12.279591. URL: https://doi.org/10.1117/12.279591.

[2] M O'Neill and C Ryan. "Grammatical evolution". eng. In: *IEEE transactions on evolutionary computation* 5.4 (2001), pp. 349–358. ISSN: 1089-778X.

[3] John R. Koza. "Genetic programming as a means for programming computers by natural selection". eng. In: *Statistics and computing* 4.2 (1994), p. 87. ISSN: 0960-3174.

[4] Anthony Brabazon and Michael O'Neill. "Diagnosing corporate stability using grammatical evolution". In: *Int. J. Appl. Math. Comput. Sci* 14 (Jan. 2004), pp. 363–374.

[5] Manuel Alfonseca and Francisco José Soler Gil. "Evolving a predator-prey ecosystem of mathematical expressions with grammatical evolution". In: *Complex.* 20 (2015), pp. 66–83.

[6] Ouassim Elhara et al. *COCO: The Large Scale Black-Box Optimization Benchmarking (bbob-largescale) Test Suite*. 2019. DOI: 10.48550/ARXIV.1903.06396. URL: https://arxiv.org/abs/1903.06396.

[7] Michael Fenton et al. "PonyGE2". In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. ACM, July 2017. DOI: 10.1145/3067695.3082469. URL: https://doi.org/10.1145%5C%2F3067695.3082469.

[8] David Fagan et al. "An Analysis of Genotype-Phenotype Maps in Grammatical Evolution". In: vol. 6021. Apr. 2010, pp. 62–73. ISBN: 978-3-642-12147-0. DOI: 10.1007/978-3-642-12148-7_6.

[9] Miguel Nicolau and Alexandros Agapitos. "Understanding Grammatical Evolution: Grammar Design". In: *Handbook of Grammatical Evolution*. Ed. by Conor Ryan, Michael O'Neill, and JJ Collins. Cham: Springer International Publishing, 2018, pp. 23–53. ISBN: 978-3-319-78717-6. DOI: 10.1007/978-3-319-78717-6_2. URL: https://doi.org/10.1007/978-3-319-78717-6_2.

[10] Michael O'Neill et al. "Crossover in Grammatical Evolution: The Search Continues". eng. In: *Genetic Programming*. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 337–347. ISBN: 3540418997.

[11] Yitan Lou. "The Paradox of Overfitting". MA thesis. Netherlands: Leiden University, 2019.

[12] Jacob de Nobel et al. "IOHexperimenter: Benchmarking Platform for Iterative Optimization Heuristics". In: *arXiv e-prints:2111.04077* (Nov. 2021). arXiv: 2111.04077. URL: https://arxiv.org/abs/2111.04077.

[13] Nikolaus Hansen et al. "COCO: a platform for comparing continuous optimizers in a black-box setting". In: *Optimization Methods and Software* 36.1 (Aug. 2020), pp. 114–144. DOI: 10.1080/10556788.2020.1808977. URL: https://doi.org/10.1080%2F10556788.2020.1808977.

[14] David Fagan, Michael Fenton, and Michael O'Neill. "Exploring position independent initialisation in grammatical evolution". In: *2016 IEEE Congress on Evolutionary Computation (CEC)*. 2016, pp. 5060–5067. DOI: 10.1109/CEC.2016.7748331.

[15] Miguel Nicolau. "Understanding grammatical evolution: initialisation". eng. In: *Genetic programming and evolvable machines* 18.4 (2017), pp. 467–507. ISSN: 1389-2576.

[16] Robin Harper. "GE, explosive grammars and the lasting legacy of bad initialisation". In: *IEEE Congress on Evolutionary Computation*. 2010, pp. 1–8. DOI: 10.1109/CEC.2010.5586336.
[17] Paul Barrett et al. "matplotlib – A Portable Python Plotting Package". In: Dec. 2005.
[18] Wes McKinney. "pandas: a Foundational Python Library for Data Analysis and Statistics". In: *Python High Performance Science Computer* (Jan. 2011).
[19] Hao Wang, Yitan Lou, and Thomas Bäck. "Hyper-Parameter Optimization for Improving the Performance of Grammatical Evolution". In: June 2019, pp. 2649–2656. DOI: 10.1109/CEC.2019.8790026.

Appendices

A Definitions

List of common acronyms

BNF Backus-Naur Form

CFG Context-free Grammar

GE Grammatical Evolution

GP Genetic Programming

ML Machine Learning

MSE mean squared error

List of common terms

codon A building block of the genotype that carries the genetic information for a single rule in the grammar. In GE, the codon is represented as an integer or a set of bits that can be translated to an integer.

context-free grammar A formal grammar in which a non-terminal can be expanded into all its different production rules, regardless of the context in which it is placed.

derivation tree Also called a parse tree, it is a graphical representation of the structure of a string that can be derived from applying the rules of a Context-free Grammar (CFG).

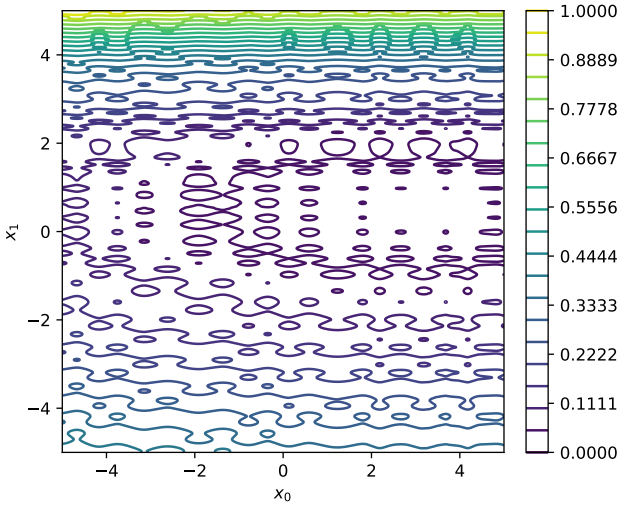
dimensionality The dimensionality of a mathematical function represents the number of input values needed to generate an output value y.

genotype The genetic makeup of an individual, which can be expressed as a phenotype through a genotype-to-phenotype mapping process. The genotype is a set of codons.

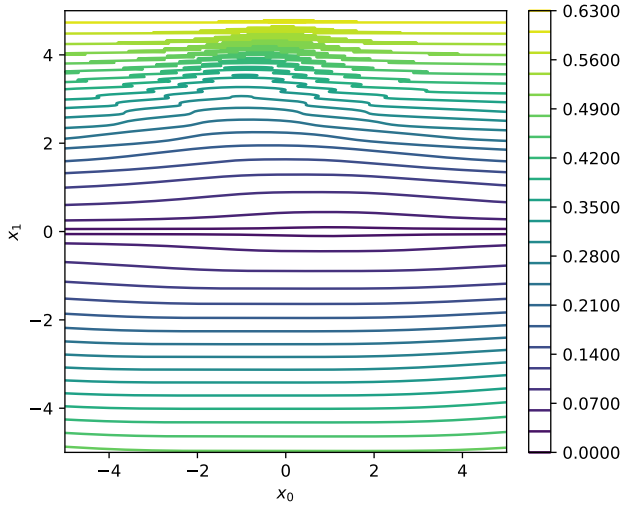
phenotype The visible characteristics of an individual, determined by the genotype of an individual.

B Figures

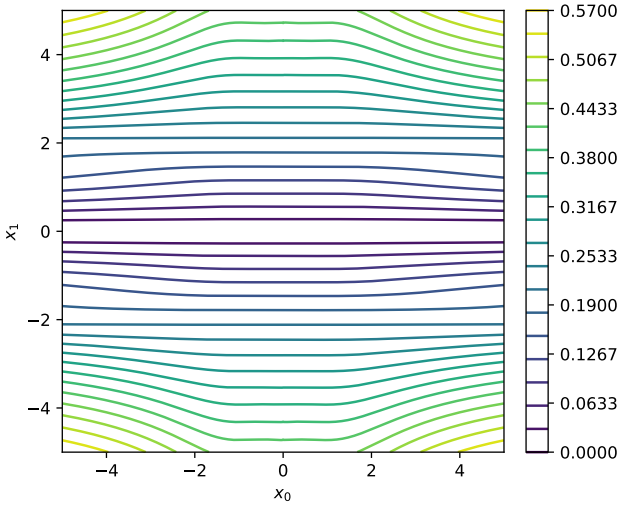
B.1 Experiment 1: Visual Comparison



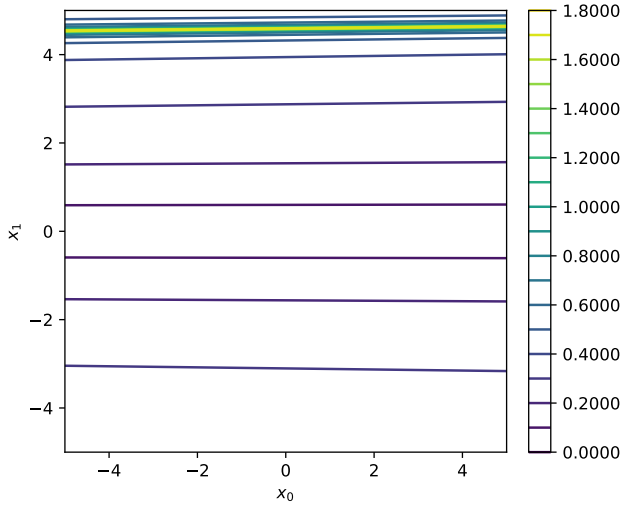
(a) BBOB Rastrigin function



(b) BEST inferred Rastrigin function

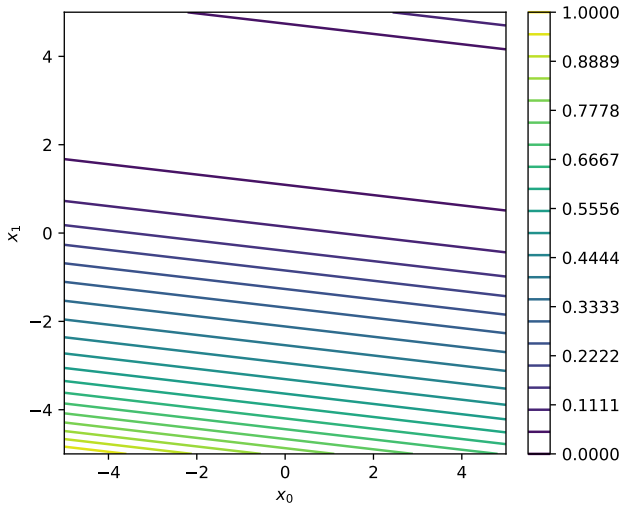


(c) MEDIAN inferred Rastrigin function

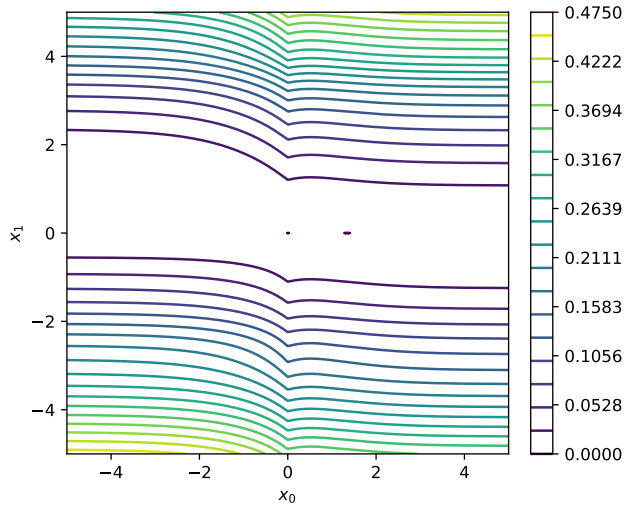


(d) WORST inferred Rastrigin function

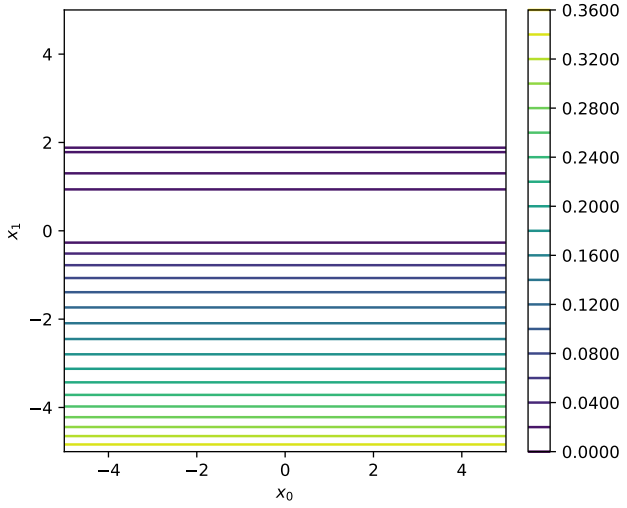
Figure 8: A set of contour plots for the Rastrigin function. The contour plot of the original BBOB function can be compared to the BEST, MEDIAN, and WORST function inferred by PonyGE2 over 20 runs.



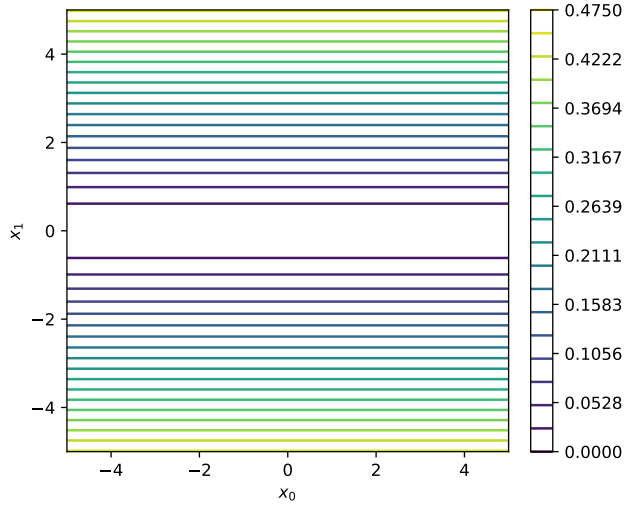
(a) BBOB Rotated Ellipsoidal function



(b) BEST inferred Rotated Ellipsoidal function

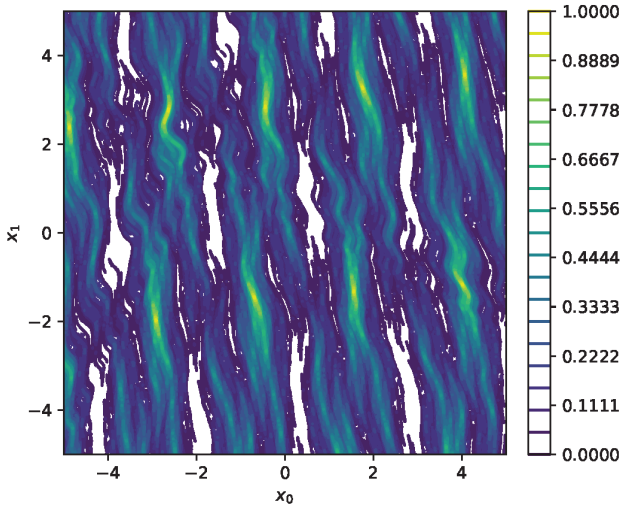


(c) MEDIAN inferred Rotated Ellipsoidal function.

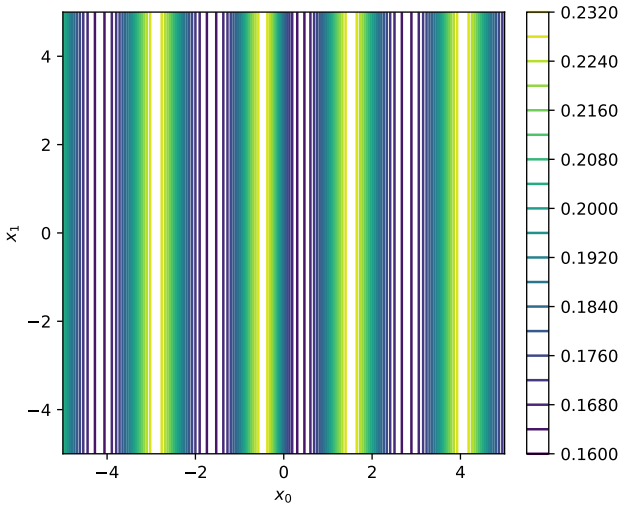


(d) WORST inferred Rotated Ellipsoidal function

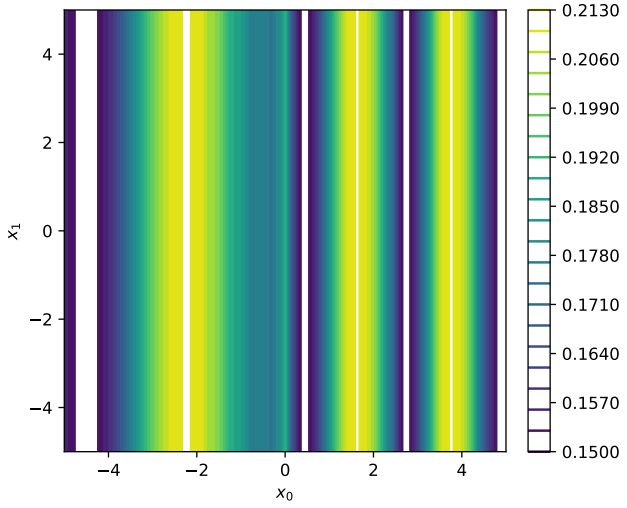
Figure 9: The contour plot of the original Rotated Ellipsoidal BBOB function can be compared to the BEST, MEDIAN, and WORST function inferred by PonyGE2 over 20 runs. Not all of the 20 inferred functions are feasible.



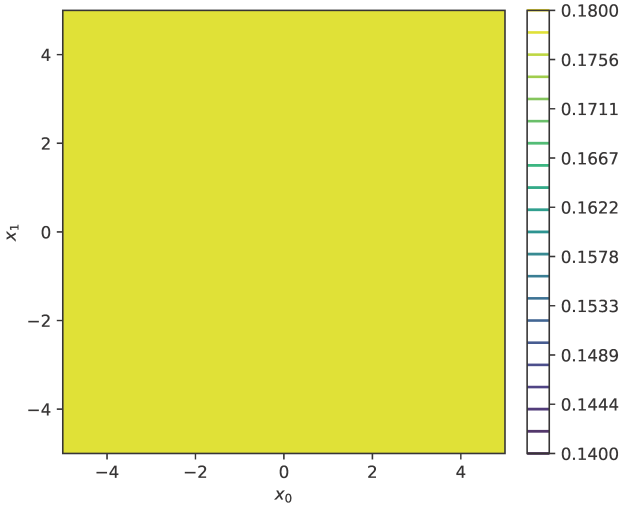
(a) BBOB Weierstrass function



(b) BEST inferred Weierstrass function



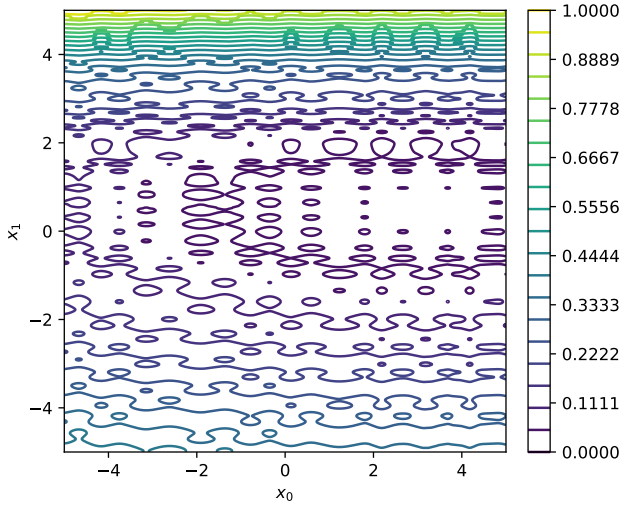
(c) MEDIAN inferred Weierstrass function



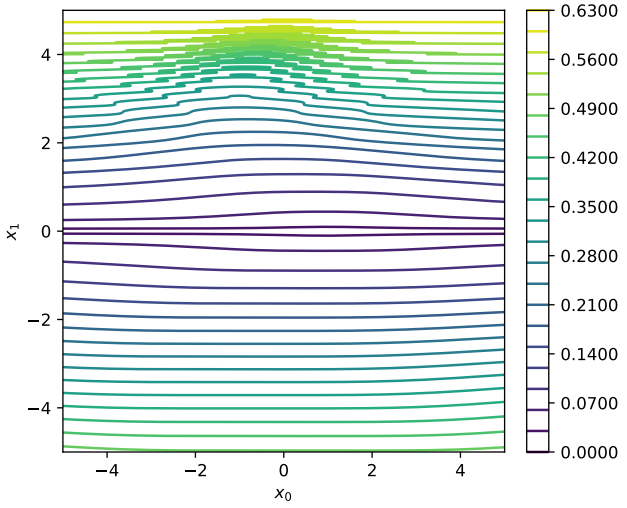
(d) WORST inferred Weierstrass function

Figure 10: The contour plot of the original Weierstrass BBOB function can be compared to the BEST, MEDIAN, and WORST function inferred by PonyGE2 over 20 runs. Not all of the 20 inferred functions are feasible.

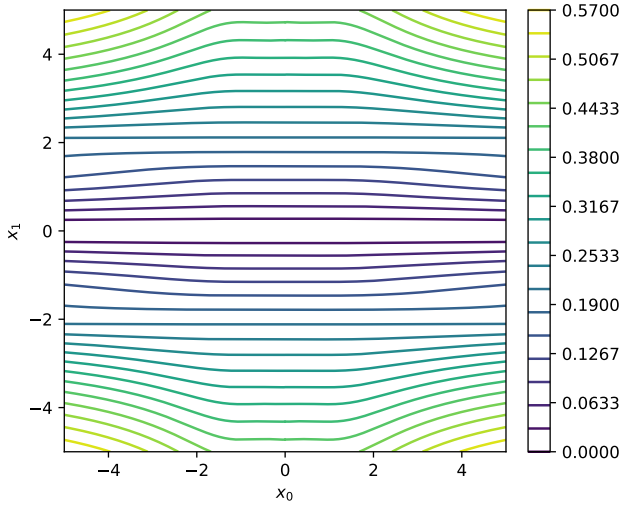
B.2 Experiment 3: Feasibility



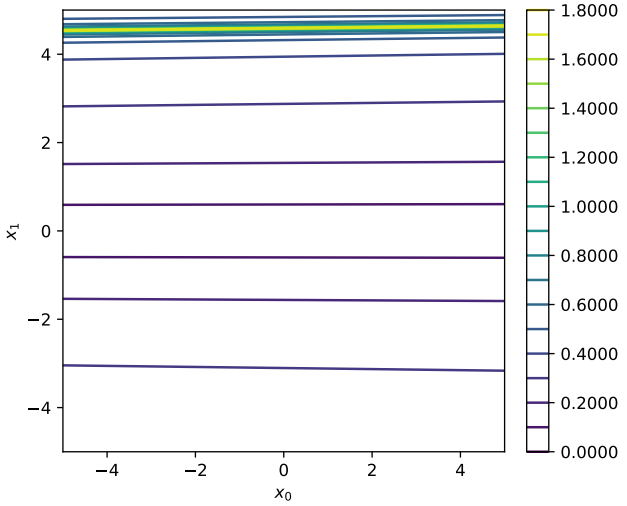
(a) Original BBOB Rastrigin function



(b) BEST inferred feasible Rastrigin function

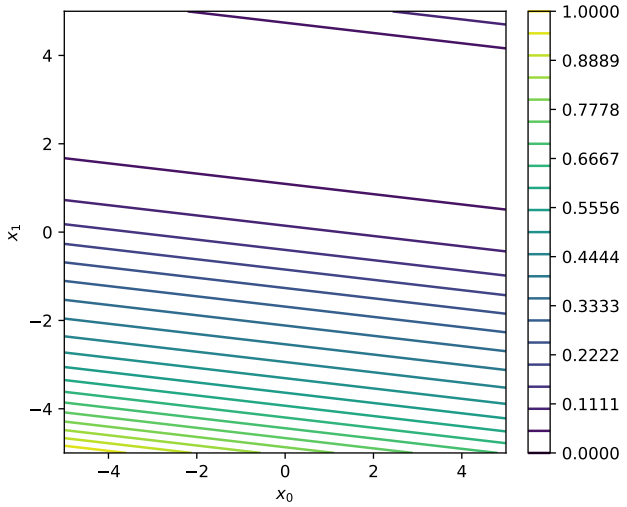


(c) MEDIAN inferred feasible Rastrigin function

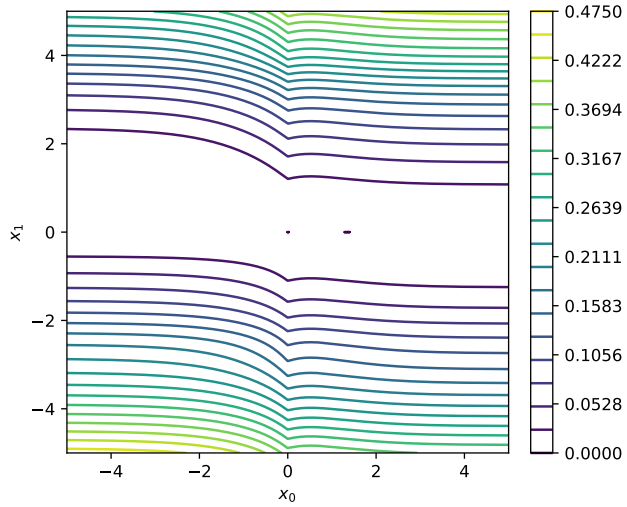


(d) WORST inferred feasible Rastrigin function

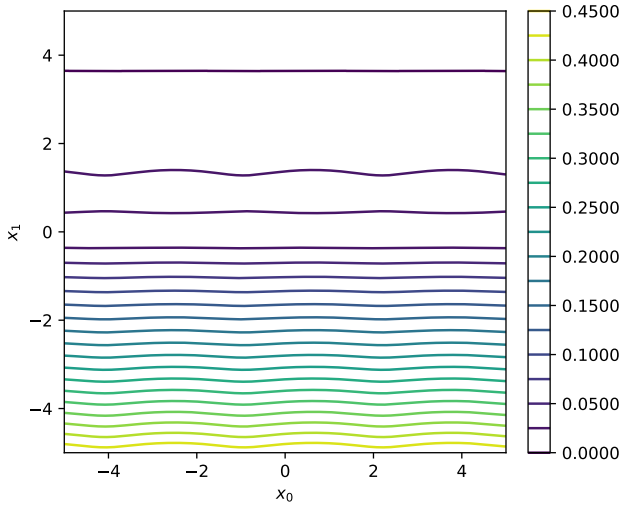
Figure 11: The contour plot of the original Rastrigin BBOB function can be compared to the BEST, MEDIAN, and WORST **feasible** function inferred by PonyGE2 over 20 runs. Not all of the 20 functions generated by GE per run are feasible.



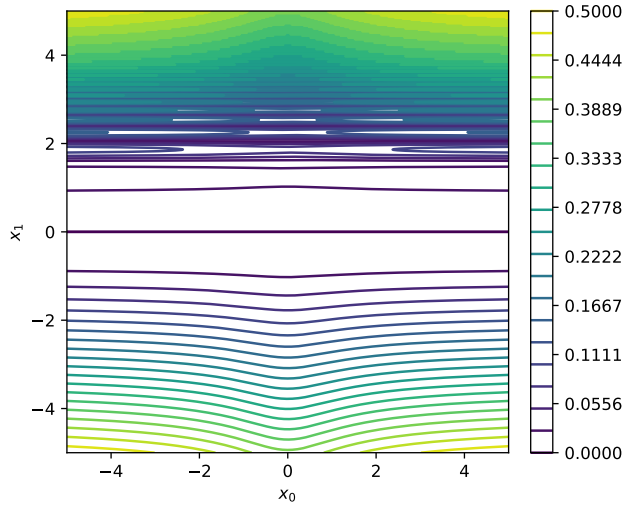
(a) Original BBOB Rotated Ellipsoidal function



(b) BEST inferred feasible Rotated Ellipsoidal function

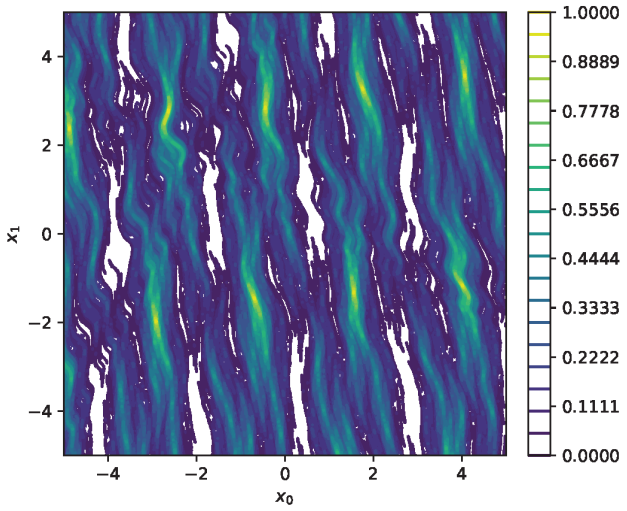


(c) MEDIAN inferred feasible Rotated Ellipsoidal function

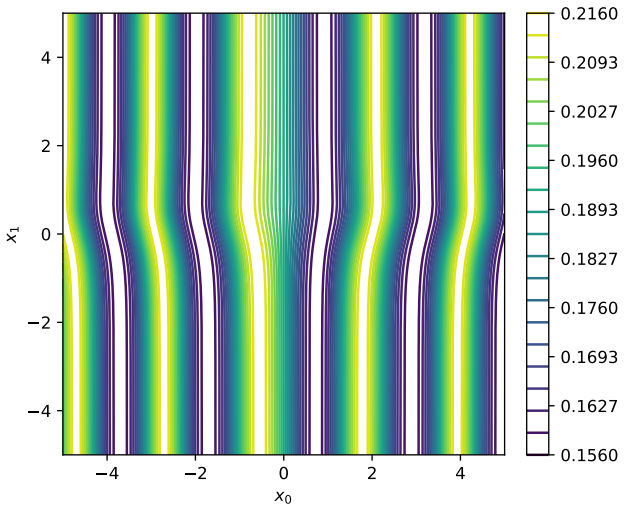


(d) WORST inferred feasible Rotated Ellipsoidal function

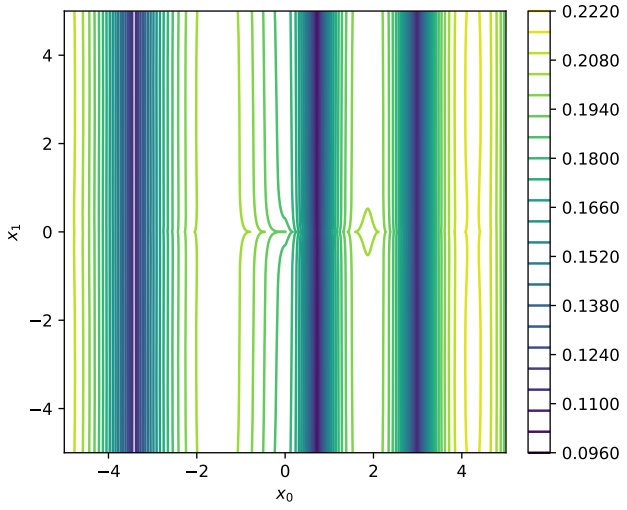
Figure 12: The contour plot of the original Rotated Ellipsoidal BBOB function can be compared to the BEST, MEDIAN, and WORST **feasible** function inferred by PonyGE2 over 20 runs. Not all of the 20 functions generated by GE per run are feasible.



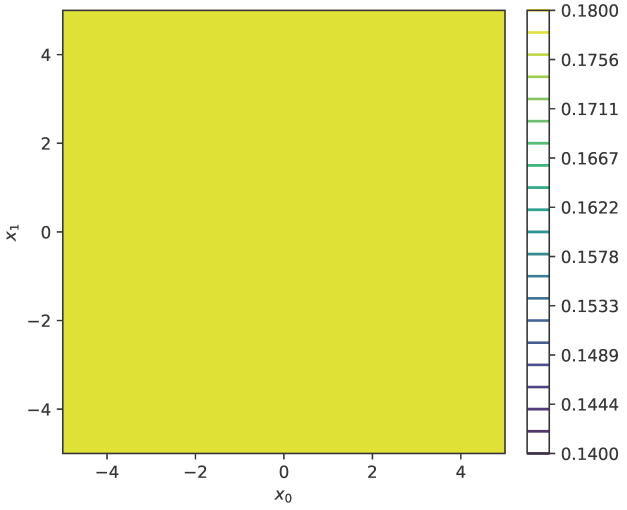
(a) Original BBOB Weierstrass function



(b) BEST inferred feasible Weierstrass function



(c) MEDIAN inferred feasible Weierstrass function



(d) WORST inferred feasible Weierstrass function

Figure 13: The contour plot of the original Weierstrass BBOB function can be compared to the BEST, MEDIAN, and WORST **feasible** function inferred by PonyGE2 over 20 runs. Not all of the 20 functions generated by GE per run are feasible.

C Parameters

```
CACHE:                  True
CODON_SIZE:             100000
CROSSOVER:              variable_onepoint
CROSSOVER_PROBABILITY:  0.75
DATASET_TRAIN:          [...]
DATASET_TEST:           [...]
DEBUG:                  False
ERROR_METRIC:           mse
GENERATIONS:            100
MAX_GENOME_LENGTH:      500
GRAMMAR_FILE:           [...]
INITIALISATION:         PI_grow
INVALID_SELECTION:      False
MAX_INIT_TREE_DEPTH:    10
MAX_TREE_DEPTH:         17
MUTATION:               int_flip_per_codon
POPULATION_SIZE:        500
FITNESS_FUNCTION:       supervised_learning.regression
REPLACEMENT:            generational
SELECTION:              tournament
TOURNAMENT_SIZE:        2
VERBOSE:                False
OPTIMIZE_CONSTANTS:     True
EXPERIMENT_NAME:        [...]
RUNS:                   20
```

D Grammar

```
<e>     ::= np.sin(<e>) |
            np.cos(<e>) |
            np.tanh(<e>) |
            np.exp(<e>) |
            psqrt(<e>) |
            plog(<e>) |
            <e>+<e> |
            <e>-<e> |
            <e>*<e> |
            aq(<e>,<e>) |
            <c><c>.<c><c> |
            <c> |
            x[0] | x[1]

<c>     ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```