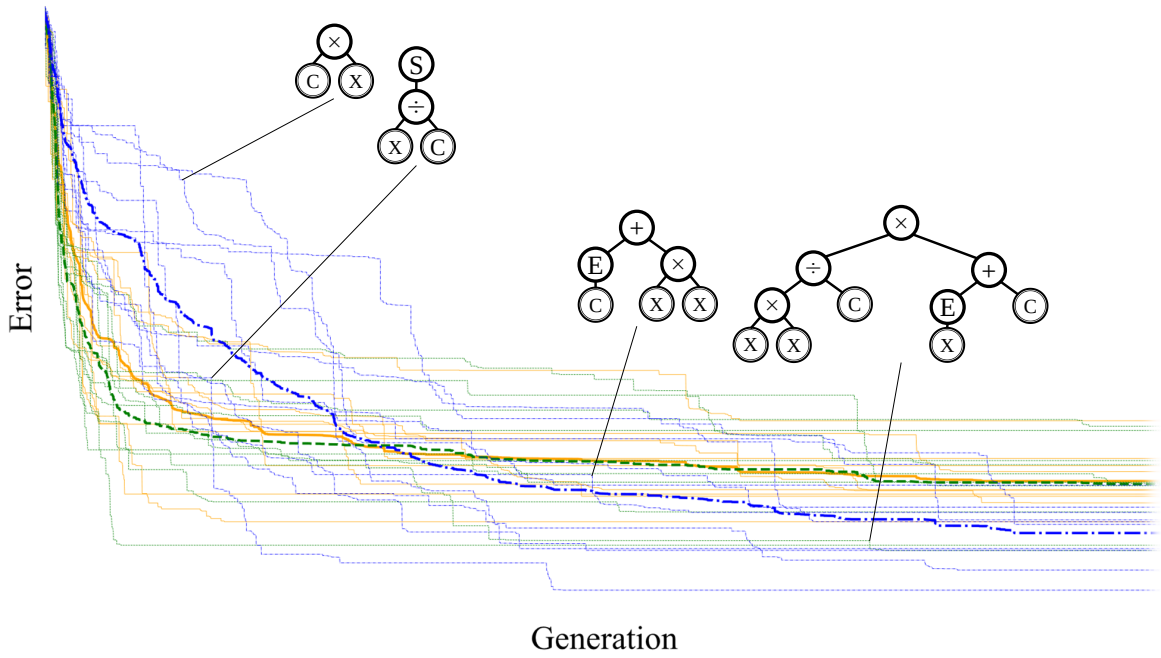
CHALMERS
UNIVERSITY OF TECHNOLOGY



Extracting Interpretable Equations from Data

Studying and developing bloat control and sampling techniques in genetic programming for symbolic regression

Master's Thesis in Applied Physics

ADRIAN ERIKSSON
FILIP FROSTELIND

Department of Physics
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2020

Master's thesis 2020:05

Extracting Interpretable Equations from Data

Studying and developing bloat control and sampling techniques in genetic programming for symbolic regression

ADRIAN ERIKSSON
FILIP FROSTELIND

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Physics
Chalmers University of Technology
Gothenburg, Sweden 2020

Extracting Interpretable Equations from Data
Studying and developing bloat control and sampling techniques in genetic programming for symbolic regression

ADRIAN ERIKSSON
FILIP FROSTELIND

Supervisor: Ignacio Mancha, i3tex AB
Examiner: Kristian Gustafsson, Department of Physics, University of Gothenburg

Master's Thesis 2020:05
Department of Physics
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Illustration of error progression from 30 genetic simulations with visual examples of individual solutions at different stages of the simulation.

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2020

Extracting Interpretable Equations from Data
Studying and developing bloat control and sampling techniques in genetic programming for symbolic regression
ADRIAN ERIKSSON & FILIP FROSTELIND
Department of Physics
Chalmers University of Technology

Abstract

Expressing relationships within data using mathematical formulas lies at the centre of scientific discovery. In recent years symbolic regression has been proposed as a tool for discovering relationships in data in order to convey information regarding system dynamics. In this study an algorithm for symbolic regression using genetic programming is developed and tested on a number of iconic equations from physics (50 equations from the Feynman lectures on physics) and breathing data from a ventilator. Two important features of genetic algorithms are investigated, namely non-disruptive bloat control and sampling methods for breeding. The results shows that the developed algorithm performs on-par with cutting-edge commercial genetic programming software and that interesting features can be extracted from input data. Further, an enhanced implementation of substitution with an approximate terminal (SAT) is performed with promising results, reducing bloat without hindering adaptation. Lastly, the impacts of roulette, linear rank and Boltzmann selection are investigated, and we find that they all produce similar results, but with different strategies regarding exploration and exploitation.

Keywords: Genetic Programming, Symbolic Regression, Stochastic Optimisation, Machine Learning, Bloat Control, Boltzmann Sampling, SAT-GP, TOPSIS

Acknowledgements

Firstly, we would like to thank i3tex AB for giving us the opportunity to do our thesis with them. We want to thank several people at i3tex for showing interest in our project, but in particular our supervisor Ignacio Mancha, for coming up with the idea of the thesis and for supporting throughout the project. We would also like to thank Kristian Gustafsson for being our examiner, for showing interest in our research and for giving valuable feedback.

Lastly, we want our families, friends and loved ones to know how much your continuous support has meant. Not only have you encouraged us during our thesis, but you have been there and supported us during our whole time at Chalmers. Without you we would not be where we are today. We will forever be grateful of your support and love.

Adrian Eriksson and Filip Frostelind, Gothenburg, May 2020

Contents

List of Figures

List of Tables

1
Introduction

Expressing relationships within data using mathematical formulas lies at the centre of scientific discovery. Consequently, scientific discoveries are often attributed to the formulation of mathematical descriptions, such as Newton's formulation of the laws of motion [1] or Maxwell's unifying equations within electromagnetism [2]. Formulas represent the predictive capability of a theory and is thus a cornerstone for theoretical confirmation and application. This connection between theory and formula allows a theory to provide rationale for the structure of its corresponding set of equations. A plain example is the damped harmonic oscillator, which is described by a formula consisting of two distinct terms. The first term describes to the force related to the extension of the oscillator and the second term describes the frictional contribution, according to,

$$F = -kx - cv.$$

The ability to convey insight from theory to formula has evoked the thought of applying this inherent connection in reverse – to gain theoretical understanding from a formula capable of predicting some feature in an arbitrary system. This could prove to be relevant in the era of machine learning and artificial intelligence when our predictive capability often transcend our theoretical understanding. However, a common problem with methods within artificial intelligence is that they lack the ability to produce interpretable models, severely limiting our ability to extract information from the model itself. Artificial neural networks for instance often include thousands of parameters, all contributing to the final model [3]. Yet, there are methods which circumvent this problem, one of which is symbolic regression.

The objective in symbolic regression is to find a mathematical expression that describes a given data set the best, based on accuracy and complexity (the given number of operations, functions, variables and constants in an expression). The optimal solution is acquired by probing the space of all included operators, functions, constants and variables. This can be achieved using different methods with genetic programming being a common approach [4]. Genetic programming is a technique that avoids brute force optimisation by applying processes analogous to mechanisms within natural population genetics. In a genetic algorithm a mathematical expression is seen as an individual representing a possible solution. By grouping individuals, creating a population, a probabilistic search based on natural selection and adaptation can be performed. The search is carried out over a number of generations, successively altering the population by selection, inheritance, mutation and recombination. Without any prior knowledge of the system genetic programming

can be used for symbolic regression in order to distil system describing mathematical expressions [5, 6].

Although symbolic regression using genetic programming can be used in order to produce a mathematical formula with great predictive capability there are an infinite number of formulas describing the same relationship. Thus, complicating the connection between expression and theory. It is therefore of great interest to specify which characteristics of an equation that makes it insightful and useful for conveying information regarding theory. The genetic algorithm must hence incorporate some definition of an insightful expression, promoting the existence of these expressions within the population.

1.1 Background

The original notion of evolving programs was first introduced by Turing when he proposed the idea of learning machines in 1950 [7]. Turing imagined an initial simple program consisting of hereditary material that can mutate. By evaluating the program based on some judgement of the experimenter, desirable changes can be promoted while undesirable ones punished, thus the program evolves. However, due to the obvious technological limitation at the time Turing never implemented this. Instead, it was Holland who popularised genetic programming with his work in the early 1970s, conceptualising the field [8, 9]. Following Holland, in the 1990s Koza produced a series of books on more practical applications of genetic programming, showcasing the potentials of the method and formulating the principles and terminology that is still used today [4]. Since then the field of genetic programming has progressed dramatically.

Because of the lack of computational power, genetic programming was initially used for simple problems, often applied to small systems with limited data points. This is no longer the case, today genetic programming is utilised on a variety of problems with ranging complexity and intricacy [10]. This change can be attributed to the increase in computational power as well as an overall increased interest in machine learning algorithms. Consequently, genetic programming has been used for a multitude of different applications ranging from modelling of biochemical reactions [11] to predicting stock prices [12].

In this work genetic programming is used for symbolic regression which was originally done by Koza but has since been developed by numerous authors. Notably, Schmidt and Lipson who used genetic algorithms to obtain conservation laws for physical systems without supplying a target value, only evaluating the internal derivatives of the input variables – highlighting the potential of genetic algorithms to be used for identification of system behaviour [5].

1.2 Problem Description

The objective of this thesis is to create a model performing symbolic regression using genetic programming. The model should be built from the ground up and be based on state-of-the-art implementation techniques within the field of genetic programming, generating mathematical expressions conveying relationships in data. The mathematical expressions should have high predictive capability while remaining interpretable for the user. The model will be tested on 50 equations from different areas of physics exhibiting a variety of features common in mathematical relationships. In addition, the algorithm will be used on data from a ventilator containing respiratory logs from a patient in order to deduce possible connections between time, pressure, tidal volume and air flow.

Further, the aim is to focus on two aspects of genetic algorithms; innovative implementation of non-disruptive bloat control; and studying effects of different sampling techniques on algorithm performance and characteristics. Non-disruptive bloat control refers to handling uncontrolled expansion of expression sizes without harming the performance of the evolutionary process.

1.3 Scope and Delimitations

This thesis will focus on the application of genetic programming for symbolic regression rather than the details of implementation and optimisation of the algorithm. Although a genetic program is created from scratch the details of the exact implementation will not be presented. This is because of the great variety of implementation methods, requiring a significant amount of effort to explain and motivate, which would redirect the focus of the report. As a result the created algorithm has non-optimal implementation schemes, resulting in a somewhat slower program than one with more optimal implementation schemes. Further, because of the shear amount of articles related to both bloat control and sampling techniques the implemented methods regarding these two topics have been selected because of their demonstrated ability to increase algorithm performance – acknowledging the fact that there are many other possibilities for implementing both non-disruptive bloat control and effective sampling.

2

Theory

This chapter is intended to provide a theoretical background to symbolic regression, stochastic optimisation and in particular genetic programming. It covers the fundamentals of genetic algorithms and highlights some important concepts, components as well as challenges within genetic programming.

2.1 Symbolic Regression

Symbolic regression (SR), also referred to as symbolic function identification, is a type of regression analysis that searches the space of mathematical expressions to provide a model that fits a data set best in terms of accuracy and simplicity [4]. In other words the objective is to find a symbolic expression that most accurately describes the relation between input and output data, and consist of the least number of mathematical operations, constants and variables. The latter is referred to as minimal model complexity.

One key difference between SR and many other regression analyses is the fact that no particular model is provided as a starting point for the algorithm. Instead, initial expressions are randomly combined using key mathematical building blocks such as numerical constants, mathematical operators and variables [4]. This limits the impact of possible human bias and/or lack of system knowledge. Thus, in SR both the model structure and parameters are optimised without any *a priori* assumptions, in contrast to more common regression methods such as linear regression, which mainly focuses on optimising model parameters with a predetermined model structure.

The search space in SR is typically much larger than the search space of other regression analyses because of the addition of model structure. Hence, SR algorithms are often more computationally demanding than other methods where the model structure is predefined. In order to reduce the computational load of a SR algorithm the number of included mathematical building blocks can be heuristically reduced using educated guesses, reducing the dimension of the probed search space. Although this will spoil the no *a priori* assumption it can be useful given that the system under consideration is understood. There is thus a trade-off between computational load and bias when preparing a SR algorithm.

2.1.1 Interpretable Models in Machine Learning

The recent advancements within machine learning (ML), particularly in artificial neural networks, have increased the ability to accurately describe, previously unknown, relationships in data. However, one common drawback when using neural networks is that they develop models with complex structures, structures that are difficult to interpret. A neural network can contain thousands of parameters, all contributing to describing the given relationship. Because of this complexity many deep learning approaches have been criticised for being *black box* models; the functions they provide are too complicated for humans to comprehend due to their number of parameters and how they often work recursively, making them hard to evaluate and understand [3].

The criticism of black box ML has led to the distinction between two different approaches within ML, namely explainable and interpretable ML. Explainable ML refers to black box models which are explained in hindsight, and interpretable ML refers to models specifically designed in order for the resulting models to be interpretable. While black box models tend to outperform interpretable models in terms of accuracy, this is not true for all systems and scenarios [3]. Symbolic regression is an example of the latter, where the purpose is to identify a symbolic function with the lowest complexity in order to describe system relations. The symbolic function can both be used when trying to describe the system mathematically but may also provide the user with insight on how the system works.

2.1.2 A Search for Parsimony

SR is a multi-objective optimisation method regarding model accuracy and complexity. The complexity of a model in SR is determined by the number of operators and operands contained by the model, and finding the most simple model is referred to as finding the *most parsimonious* model. A guideline for parsimony can be derived from the principle of Occam's razor which states that, "one should not increase, beyond what is necessary, the number of entities required to explain anything" [13].

In SR and other ML approaches the principle of parsimony tells us that if there are several models producing the same result, one should choose the simplest one. When models produce the same results this is a simple procedure, but one may also account for the parsimony in models with different results and penalise the more complex ones, allowing both the accuracy and parsimony of the model to influence the result [13]. The simplest interpretation of parsimony in SR is to define the most parsimonious model as the model that contains the lowest number of constants, variables, mathematical operators, etc., that describes the data.

2.2 Stochastic Optimisation

Stochastic optimisation refers to a collection of different methods for either minimising or maximising an objective function which incorporates randomness [14].

Classical optimisation methods can be used to solve a wide variety of optimisation problems, in particular if the problem is known to be convex [15]. Methods like gradient descent optimisation works well on convex problems but will fail in other problems due to the possibility of getting trapped in local optima. Stochastic optimisation methods seek to avoid getting trapped in local optima by introducing some stochastic functionality, and have during the last few decades become an essential tool for science, engineering, statistics, computer science and business [14]. A common stochastic optimisation method is stochastic gradient descent which has introduced a stochastic component compared to normal gradient decent. Instead of looking at the gradient of the whole data set only one or a few random data points are chosen for computation of the gradient, which has proven to be an effective approach [16].

In optimisation problems local optima are solutions which are optimal within a neighbouring set of candidate solutions. The difference between local and global optima is that the global optimum is the optimal solution among all possible solutions. While introducing some stochastic functionality to an optimisation method might reduce the possibility of getting trapped in local optima, there is no guarantee. This phenomenon is referred to as *premature convergence* and is present in many types of optimisation problems, such as SR [15]. In SR premature convergence corresponds to getting stuck at some symbolic expression where neighbouring solutions are worse but there exists other structurally different expressions with better performance.

Besides introducing randomness to optimisation methods there are several other techniques that can be utilised to avoid getting stuck in local optima. Because of the vast applicability of optimisation some of these techniques are more well-suited than others, depending on the problem. For SR using genetic programming one of these techniques is the island model [15], explained in section 2.6 after the introduction of genetic programming in section 2.3.

2.3 Genetic Programming

Genetic programming (GP) is a search method based on natural evolution – a form of artificial evolution that operates on data structures within the computer. It is an empirical modelling method for system modelling, emulation, monitoring and control, with applications in a wide range of problems, including; identification of physical conservation laws [5]; optimising release rates of pharmaceutical tablets [17]; designing high performance mechanical structures [18]; as well as predicting material properties of organic compounds [19]. GP can provide insight regarding the underlying physical mechanisms and produce parsimonious models due to the incorporation of both expression complexity and accuracy in the evaluation process [20]. Furthermore, genetic algorithms have the ability to identify necessary variables to capture the system behaviour with limited *a priori* assumptions.

A genetic algorithm progressively produces more accurate solutions by allowing a group of possible solutions, a population, evolve over generations. A solution is

called an individual and contains the structure of some solution to a specific problem. The structure depends on the situation, however when using GP for symbolic regression it is most common to represent each individual as an expression tree, containing the mathematical formula. By selecting the most fit individuals to progress to the next generation and introducing various minor and/or major changes to their structure, the population can evolve, resulting in more adapted individuals [21]. The changes are performed by genetic operators, mainly replication, mutation and crossover, each performing different operations in order to progress the population. In the following sections GP will be discussed in greater detail, highlighting the different components in a genetic algorithm.

2.3.1 The Conventional Genetic Algorithm

A conventional genetic algorithm has four major design features which has to be determined before running simulations [4]. These are,

1. representation and data structures of an individual,
2. fitness measure,
3. parameters for controlling the algorithm, and
4. how to compile the result and criterion for terminating the run.

Each design feature has implications on the algorithm's performance. The representation of an individual must allow for a complete mapping of the search space and the implementation method of this structure has major implications on the speed and memory usage of the program. An example implementation for symbolic regression is the use of expression trees. Expression trees can be implemented using various data structures and has the benefit of enabling simple implementation of genetic operations [22].

The fitness measure is the metric for which the evaluation of each individual is based on. It is case specific, non-unique, and has to be capable of evaluating all individuals. A common fitness measure for SR is the root-mean-square error although there are numerous other alternatives, that does not only account for the accuracy of the model but also the complexity. The definition of the fitness measure will influence the selection and in turn which individuals that will progress the evolutionary process.

The main parameters for controlling the algorithm are population size and maximum number of generations, together these parameters determine how well the simulation will perform. Population size determines the amount of genetic material that is available at any instance and the total number of generations determines how long the population is allowed to adapt to the system. Other parameters for controlling the algorithm include ratios for the genetic operators (commonly replication, mutation and crossover), as well as different internal settings for each of the components. In a non-conventional genetic algorithm several intermediate components might be present, adding additional parameters. Also, the genetic operators

might have different functionality depending on their implementation, resulting in extra parameters. The optimal settings of an algorithm depends on implementation and system. Common parameter optimisation techniques include trials of different combinations in order to find the best settings. There are also other parameter optimisation methods which avoids large number of tests [23].

The simplest termination criterion for a genetic algorithm is to proceed until the maximum number of generations is reached. This criterion enables maximum control for the user regarding computational time and ensures that the best solution is found, given that the maximum number of generations is sufficiently large for the problem. However, if the fitness metric is defined in such a way that an individual can attain a perfect score, this can also be used as a termination criterion. When terminating the adaptation process the program should compile the result gathered from the simulation and present it to the user. This usually includes presenting the Pareto front of the run. The Pareto front includes the best solution (individual with best fitness) for all complexities, giving a broader perspective of the result compared to a single best solution [20]. It is explained further in section 2.5.1.

Given that the four major design features of the genetic algorithm have been dealt with the program can be executed. The program itself consists of three main components, initialisation, genetic iteration and termination. An illustration of this is shown in figure 2.1, including the three components. The purpose of each component is:

1. Initialisation includes creating a randomly generated population of individuals as a starting point for the algorithm. The population should be generated in such a way that it has an abundance of genetic material and high genetic diversity in order to speed up the search process and to avoid non-optimal solutions.
2. Genetic iteration includes the actual probing of the solution space using genetic programming and consists of four parts:
    i. Fitness evaluation refers to calculating the score measure for each individual.
    ii. Termination refers to evaluation of the termination criterion, if the criterion is true, the genetic process is terminated, otherwise it continues.
    iii. Selection refers to selecting individuals for creating the next generation, this is done using a sampling method usually based on selecting individuals more fit than others to proceed to the next generation.
    iv. Genetic operators refers to different operations that are used in order to create offspring for the new generation. Genetic operation includes; replication, copy an individual; mutation, apply a random change to an individual; crossover, swap parts of individuals with each other.
3. Result includes summarising the genetic iteration and preparing the result based on the user's requirements.

Determining each design feature and parameter of a genetic algorithm is a challeng-

ing task and will impact the end result. A common difficulty is balancing exploitation and exploration [20]. Exploitation refers to optimisation of a small number of solution structures, maximising their fitness, creating optimal individuals given the small number of structures. In contrast, exploration refers to searching for solutions on a large number of structures, without fully optimising them. Neither exploitation nor exploration can be utilised alone in order to get accurate and reliable results. Instead the algorithm should balance these two tactics in order to perform well. A typical approach is for the algorithm to initially favour exploration while increasing exploitation with increasing generations. This increases the probability of the optimal solution structure to be found while ensuring that it has the correct parameters as the simulation is complete.

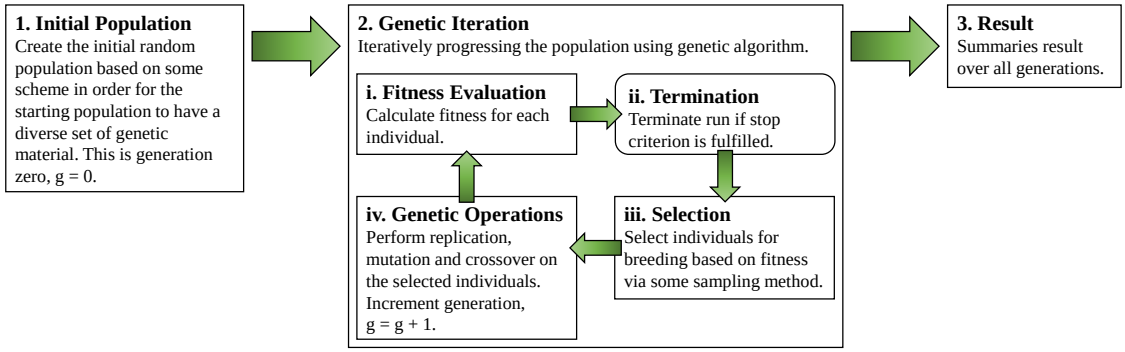


Figure 2.1: Schematic illustration of a conventional genetic algorithm. The three main components is the initialisation of the population, the genetic iteration and the termination/result. The genetic iteration consists of four internal parts which the iteration cycles each generation.

2.3.2 Representation and Data Structures

In any self-improving system there is at least one structure that undergoes adaptation. In GP it is the population and its individuals that adapts throughout the simulation, where each individual in every generation represents a point in some search space. Compared to other techniques genetic methods involve a parallel probing of the search space including all individuals in the population. The individual structures that undergo change are hierarchically structured computer programs [4]. Their internal structure can dynamically change throughout the genetic process and their size, shape and content govern how well they perform.

Although an individual can exhibit complex structures it is composed out of simple building blocks, analogous to the human genome showcasing extraordinary complexity using only a handful of relatively simple molecules. The building blocks of an individual is referred to as basis functions and terminals, and provide the syntax for creating and changing individuals. A basis function operates on a number of arguments referred to as the operator's arity. A unary operator (accepting 1 argument) has arity 1, a binary operator (accepting 2 arguments) has arity 2, etc. The

included basis functions are called the function set and may include: arithmetic operators ($+$, $-$, $\div$, ...), mathematical functions (cos, exp, log, ...), Boolean operators (*AND*, *OR*, ...), and any other general or domain specific operators. A terminal is an operand, accepting no arguments, and is typically a variable or a constant. Variables are inputs, sensors, detectors or state variables describing some feature in a system. Constants are numerical constants or other similar instances of constants. The included terminals are called the terminal set. The function and terminal sets determines the search space of a genetic algorithm. Hence, these sets must include the basic building blocks in order to be capable of expressing an accurate solution, referred to as sufficiency of the function and terminal sets. There is no definite or unique approach to this, in some situation the inclusion is inherent to the problem statement and in some it is not [4]. Including additional primitive functions and terminals has a degrading effect on the algorithm, however, in cases where it is not clear what the minimal sufficiency is, it is preferable to include more than needed to avoid missing potential solutions.

2.3.2.1 Expression Tree Representation

An individual in GP is typically represented by a tree structure [24]. In SR this can be done using an expression tree. In an expression tree each leaf is an operand (terminal) and each internal node is an operator (function). In turn all subtrees are syntactically valid subexpressions. This makes expression trees suitable for representing individuals.

Expression trees can be implemented in numerous ways depending on the situation. The implementation has an impact on how the algorithm performs, for instance, some implementations might allow for faster fitness evaluation while others results in more efficient memory usage. The optimal implementation depend on the problem and setup, and selection of proper data structures can improve the performance of the algorithm [22, 25].

In figure 2.2 an example expression tree is illustrated. The tree consists of 6 nodes, 3 terminals and 3 functions. The tree is a representation of the formula $\sin(x) \times (x \div 4.2)$, where x is some variable. The conversion from tree to formula is not one-to-one, and depending on the traversal method the notation of the given formula differs. A postorder traversal will result in postfix notation, a preorder traversal will result in prefix notation and an inorder traversal will result in common infix notation. It often computationally beneficial to use either post- or prefix notation.

2.3.2.2 *Ramped Half-and-Half* Initialisation

The initial structures in GP plays an important role for introducing genetic diversity into the population before the genetic search is carried out. A proper initialisation can improve the speed of which the algorithm finds solutions significantly [4]. When using expression trees one popular initialisation method is *ramped half-and-half* initialisation. This method utilises a combination of two methods for generating

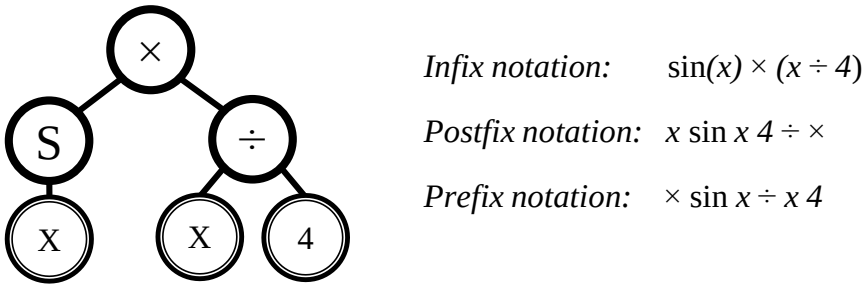


Figure 2.2: Expression tree representation of a simple mathematical expression with three notations depending on the traversal method used to evaluate the tree.

individuals, the grow and full method. The grow method refers to generating an individual by randomly filling the children of all nodes until every leaf is a terminal. This will result in an unbalanced tree. This can either be done randomly or with a maximum allowed depth at which nodes can only become terminals, limiting the size of the tree. The full method instead creates a balanced tree by only selecting functions until a certain depth. An illustration of trees generated using grow and full methods is shown in figure 2.3. The *ramped half-and-half* method divides the population size into equal sized portions for every depth between one and a user-specified maximum, and for each depth half of the individuals are generated using the grow method and half using the full method. This ensures a variety of sizes, shapes and contents of individuals in the initial population.

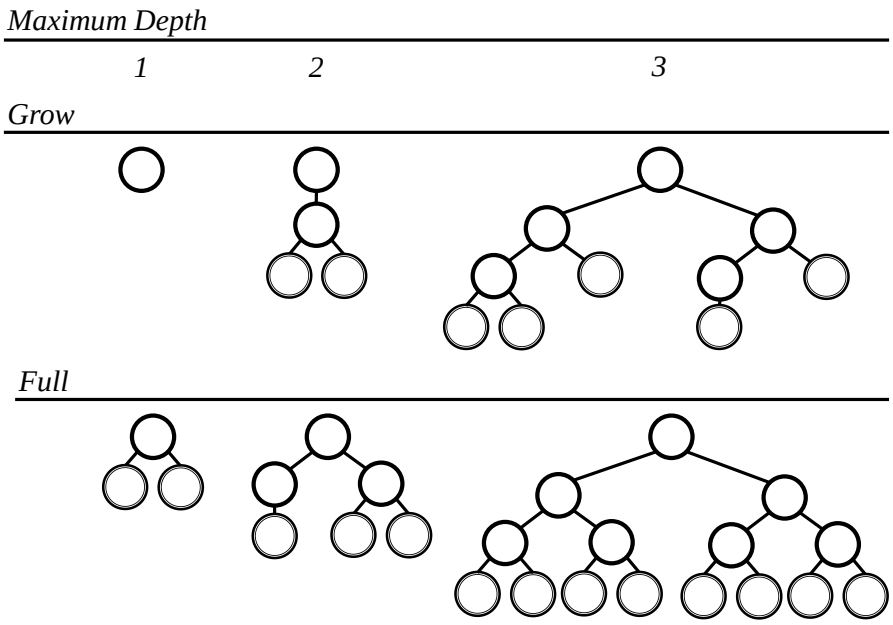


Figure 2.3: Example trees generated using grow and full method. The figure shows three examples for both grow and full methods corresponding to 1, 2 and 3 as maximum depth. These generation methods are used in order to create the initial population in genetic algorithms.

2.3.3 Fitness

Evaluating fitness is the most important procedure in GP [26]. The fitness measure determines the accuracy of the solution and guides the selection of individuals for subsequent generations. Calculating fitness is also the most computationally demanding step.

Fitness is case specific, meaning that for different problems different methods for determining an individual's fitness is needed. The most common approach for fitness is creating an explicit fitness measure for each individual, assigning each individual with a scalar measure by some well-defined evaluation procedure. This measure of fitness is called the raw fitness, r, and is the natural fitness for the given problem, e.g. for the travelling salesman problem it is the distance travelled after visiting each vertex. If the raw fitness is defined as the error, it is calculated for individual i in a population of size M in generation g as,

$$r(i,g) = \sum_{j=1}^{N} |S(i,j) - T(j)|, \tag{2.1}$$

where $S(i,j)$ is the returned value of individual i for fitness case j and where $T(j)$ is the target value for fitness case j, given N fitness cases. However, due to the definition of raw fitness as the natural measure for a specific problem the better value of fitness may either be smaller or larger.

In addition to raw fitness there is standardised fitness, s, adjusted fitness, a, and normalised fitness, n. The standardised fitness restates the raw fitness so that a lower value is better than a larger. The purpose of standardised fitness is, as the name suggests, to create a standardised fitness measure which has the best value equal to zero. If the raw fitness already obtains these two features the raw fitness is equal to the standardised fitness. Otherwise, the reversal from raw to standardised fitness is,

$$s(i,g) = r_{\max} - r(i,g), \tag{2.2}$$

where $r_{\max}$ is the maximum possible value of the raw fitness. It customary but not a absolute necessity to use standardised fitness, however it simplifies the comparison between individuals. Adjusted fitness is a further optional expression of the fitness and is calculated from the standardised fitness according to,

$$a(i,g) = \frac{1}{1 + s(i,g)}. \tag{2.3}$$

The adjusted fitness lies between 0 and 1, where an individual with larger values of adjusted fitness is better, reversed compared to the relation in standardised fitness. The benefits of using adjusted fitness is that it exaggerates differences for individuals close to the optimal solution, which is important in later generations, when the distinction between good and very good individuals must be done. For instance, given a standardised fitness ranging from 0 to 100, two individuals with (standardised) fitnesses 93 and 95 have the corresponding adjusted fitnesses 0.011 and 0.010,

whereas two individuals with fitnesses 1 and 3 have adjusted fitnesses 0.5 and 0.25. Thus, making a similar difference between two individuals more prominent as they become more adapted.

Normalised fitness is used when it is necessary for an individual's fitness to represent the probability of being selected for creating the next generation – such is the case for several selection methods, which will be discussed in the next section, section 2.3.4. The normalised fitness is calculated simply by,

$$n(i, g) = \frac{a(i, g)}{\sum_{j=1}^{M} a(j, g)}, \tag{2.4}$$

dividing each individual's fitness with the sum of all individuals' fitness.

2.3.3.1 Explicit Fitness Measure

When using GP for SR a common explicit fitness measure is root-mean-square error (RMSE). RMSE is defined as,

$$\text{RMSE} = \sqrt{\frac{\sum_{i=1}^{N}(\hat{y}_i - y_i)^2}{N}}, \tag{2.5}$$

where $\hat{y}_i$ and y_i are predicted and target values for instance i out of total N instances, respectively. RMSE provides a solid and well-studied measure for evaluating a models performance in regards to some target values.

2.3.4 Selection

Selection occurs once every iteration in the genetic algorithm and is the process of deciding which individuals that will be used for creating the subsequent generation. Selection is performed based on the fitness and should prioritise individuals with higher fitness over those with lower. However, simply selecting a few of the best individuals will result in highly fit individuals with low genetic diversity, which will hinder the progress of a population. Because of this a genetic algorithm must balance the selection process between exploitation and exploration. This is done by using sampling methods. In the following sections the five most prominent selection methods are explained, namely elitism, roulette, rank, tournament and Boltzmann selection.

2.3.4.1 Selection Pressure

Selection pressure is an important concept within selection in GP, and refers to the sampler's interpretation of the fitness [27]. Given two individuals with different fitnesses, the selection pressure determines how much this difference will influence the probability of selecting any of these individuals. A high selection pressure implies that the sampler will be inclined to select individuals based on fitness, selecting high scoring individuals much more frequently than low scoring ones. A low selection pressure implies the opposite, that the sampler will select individuals more randomly, reducing the influence of the fitness. Selection pressure is an important concept

because of its implications on exploitation and exploration. A high selection pressure will promote exploitation whereas a low selection pressure will promote exploration.

2.3.4.2 Elitism Selection

Elitism selection is the most straight forward approach to selection. In elitism selection the highest scoring individuals are passed down to the next generation. Elitism selection is highly exploitative but is used in order to preserve a small percentage of the best individuals over the course of the evolutionary process. This prevents individuals with good potential from falling out the population and preserves the current best solution, improving the overall performance of the algorithm. It is standard practice in GP to use elitism for a small amount of the selection.

2.3.4.3 Proportional Roulette Wheel Selection

In proportional roulette wheel selection, or simply roulette selection, individuals are selected based on their fitness. The selection can be imagined as spinning a roulette wheel with one segment for each individual where each segment is proportional to the corresponding individual's fitness. This implies that individuals with higher fitness will be selected more often than those with lower, mimicking the effect of survival of the fittest. The probability of selecting an individual is equal to their normalised fitness. The benefits of roulette selection is that sampling of individuals will be done on the whole population with all individual being subject for breeding. However, roulette selection is prone to premature convergence and to hinder genetic diversity [28]. Early outstanding individuals introduces a bias in the search which might steer the search into a genetic trap. Meaning that early sub-optimal solutions might dominate the population, leaving little room for alternative solution to evolve.

2.3.4.4 Linear Rank Selection

Rank selection tries to combat the deficiencies of roulette selection by eliminating the impact of each individual's absolute fitness value. In rank selection the probability of selecting an individual is based on its rank, where rank is the placement of the individuals fitness in ascending order. In linear rank selection the probability of selection is linearly dependent on the rank. The probability of selecting individual i is thus,

$$p(i) = \frac{rank(i)}{M(M-1)}, \tag{2.6}$$

where M is the number of individuals. This formulation of probability of selection prevents a few individuals from dominating the population in the early generations, promoting exploration over exploitation. However, in contrast to roulette selection, linear rank selection might instead cause the population to converge much slower due to the resulting small differences in probability of selection [29]. There are in addition to linear ranking other rank-based selection schemes which interpret the rank differently resulting in different sampling characteristics. One example is

exponential rank selection, which introduces an exponential based ranking scheme that is highly exploitative in contrast to linear ranking.

2.3.4.5 Tournament Selection

In tournament selection a small number of individuals compared to the total population are chosen at random. These individuals then compete against each other, and the one with the highest fitness wins and is hence selected. The number of individuals selected for each competition is called tournament size and is commonly set to 2, referred to as binary tournament selection [28]. Tournament selection is also beneficial due to it being easy to implement and can be executed in parallel, improving the speed of selection. However, similar to linear rank selection, tournament selection is prone to degrading the convergence speed.

2.3.4.6 Boltzmann Selection

Boltzmann selection is a unique selection scheme compared to the previous mentioned ones because it changes the selection pressure throughout the search. In Boltzmann selection the sampling of individuals is controlled by a continuously varying temperature,

$$T = T_0(1-\alpha)^k, \quad k = \gamma + \beta\frac{g}{G}, \tag{2.7}$$

where T_0 is the initial temperature, g is the current generation, G is the maximum generations and α, β and γ are hyperparameters defining the profile of the temperature change. The selection is then performed by sampling a potential energy surface where the probability of selecting individual i, with fitness f_i is,

$$p(i) = \mathcal{C}\exp\Big(\frac{-(f_{\max} - f_i)}{T}\Big), \tag{2.8}$$

where T is the Boltzmann temperature, $f_{\max}$ is the maximum fitness and $\mathcal{C}$ is a normalisation constant,

$$\mathcal{C} = \Big(\sum_{j=1}^{M}\exp\big(\frac{-(f_{\max} - f_j)}{T}\big)\Big)^{-1}. \tag{2.9}$$

Boltzmann selection is more intricate than the previous mentioned selection tools and has more options as to how it should perform the sampling. In essence Boltzmann selection attempts to balance exploration and exploitation by lowering the temperature for every generation. As can be seen in equation 2.8 the temperature is inversely connected to the selection pressure. A high temperature results in low selection pressure whereas a low temperature results in high selection pressure. This leads to the selection pressure initially being low and continuously increasing. The nature of this change is determine by the hyperparameters; α and β determines how much each generation should differ from one another regarding selection pressure; and γ determines the initial selection pressure.

2.3.5 Genetic Operators

Genetic operators are used to modify individuals undergoing adaption in GP. The objective of genetic operators is to preserve and restructure existing as well as to introduce new genetic material. The three main genetic operators are replication, crossover and mutation. These operators, among others, are used collectively in order to change the individuals selected for breeding.

2.3.5.1 Replication

Replication is the simplest form of genetic operation and involves copying the structure of a parent individual [4]. The resulting child individual is thus an exact copy of the parent. Replication is performed in order to preserve well performing individuals and to prevent loss of functional structures due to other genetic operators or the selection scheme. Also, by replicating individuals they can serve as references for the modified individuals – meaning that all modified individuals in a new generation will be directly compared to the replicated individuals from the previous generation. This helps to prevent evolutionary dead-ends by preventing deteriorating modification from happening over consecutive generations. Replication is often used on a small number of individuals, often selected using elitism selection [27]. Thus, the highest scoring individuals will always remain in the population.

2.3.5.2 Crossover

Crossover mixes the structure of two parent individuals creating child individuals that include features from both parents. The basic crossover operation requires selection of two parent individuals. Then, for each parent structure a point is selected at random, uniformly distributed. Because of the tree structure's inherent properties each point is a subtree and can thus be swapped with one-another, creating two new individuals with mixed features. A schematic illustration of this is shown in figure 2.4. In the figure the two *initial individuals* have two highlighted subtrees, representing the subtrees to-be swapped. Hence, the *crossovered individuals* have had their subtrees replaced, creating two new individuals.

The main purpose of the crossover operation is to introduce major and minor changes based on already existing subtrees to the selected individuals for breeding. The idea is that the selected individuals in any generation contain structural fragments that are important for their performance. The crossover operator will stochastically restructure these individuals in order to find more accurate solutions by combining fragments of generationally well-performing individuals.

Crossover can be performed in numerous ways [30]. Beyond the standard crossover previously described, referred to as free crossover, there are also other, such as size-fair and homologous crossover. In free crossover there are no illicit crossover scenarios, meaning that any two subtrees of the parent individuals can be swapped. A terminal can be swapped with the root of a tree, creating offspring with possibly tangible deviations to their parents. In turn, free crossover allows tree structures to

grow uncontrollably from generation to generation, having the potential to drastically slowing down the program.

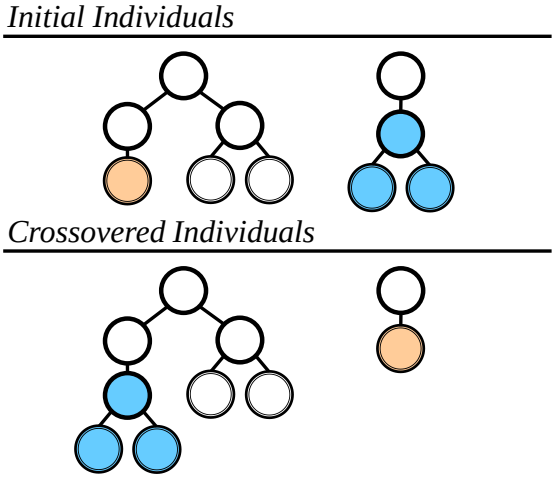


Figure 2.4: Schematic crossover. The initial individual represents the parents and the crossovered individuals are the resulting children of a basic crossover operation using the highlighted subtrees.

Size-fair (limited) crossover is similar to free crossover but limits the children's ability to grow [31]. This is enforced by only allowing subtrees to swap given that both subtrees are within a certain size difference, e.g. given a maximum size difference threshold of 1, two subtrees with sizes 2 and 4 are not allowed to be swapped, but subtrees with size 2 and 3 are allowed. Thus, controlling the growth of the offspring, resulting in less degradation of program speed while, however, preventing possible beneficial crossover operations from happening.

It is assumed that if an individual survives the selection process it must contain important structural fragments that can be used to create new individuals [31]. However, it is reasonable to assume that the context of which these fragments are executed will determine their worth – homologous crossover tries to acknowledge this fact. Homologous crossover is similar to size-fair crossover but instead of randomly selecting two subtrees with a maximum size difference, the selection is based on the closeness of the subtrees. The closeness of two subtrees is not uniquely defined but can be defined as the distance to the root, resulting in swapping subtrees at the same or close to the same depth.

2.3.5.3 Mutation

The mutation operator provides potentially new genetic material to the population. Mutation has two important functionalities [4]; to reintroduce diversity to a prematurely converging population; and to introduce changes at positions in individuals where other genetic operation can not access. The first functionality revolves around the often highly nonlinear search space of genetic algorithms and that two solution with similar fitness might have drastically different structures. A population can converge towards a solution with non-optimal structure, which can be hindered by

introducing random major and minor structural changes to the individuals. The second functionality is a consequence of how crossover operates. If for instance, for SR, a certain fragment is interpreted as $x_1 + x_2$ but should (for optimally reasons) be $x_1 - x_2$, mutation can be effective.

There are a multitude of different mutation operations, three of them are point, branch and constant mutation, the latter being specific for when there are numerical constants present (SR). Point mutation refers to randomly selecting a node in an individual and then replacing it with a randomly selected replacement node, without compromising the syntactic integrity of the individual. Branch mutation also requires selection of a random node in an individual but instead of only replacing the node the entire subtree is replaced by a new generated subtree. Constant mutation refers to random adjustments of leafs containing numerical constants in SR. A genetic program may include one, two or all of these mutation operations based on the specific problem.

2.4 Bloat in Genetic Programming

The dynamics of a genetic program is determined by the operations which are applied to its population. A common feature observed in genetic algorithms is that the average size of individuals in a population tend to grow over generations, sometimes uncontrollably [32]. Furthermore, this growth is not guaranteed to be accompanied by a corresponding improvement in fitness. This phenomenon is referred to as *bloat* and is defined as: *program growth without (significant) return in terms of fitness*, and is subject to extensive research within GP [32]. Not only has it been studied due to it being an interesting phenomenon but also due to how it affects the overall performance of the program. Because bloat causes individuals to grow bigger it is thereof causing each generation to be more computationally expensive. Additionally, by making individuals bigger the resulting symbolic expressions are harder to interpret and may also cause the program to exhibit poor generalisation [32].

The topic of bloat can be divided into two subtopics: what is causing and how to control bloat, which have both been studied extensively. Studying what is causing bloat in a genetic program is important in order to control it, but not absolutely necessary. It is also possible to control bloat without explicitly dealing with the cause, only with the bloat itself [33].

2.4.1 Causes of Bloat

There are several theories and explanations as to what is causing bloat. Two of the more established theories are fitness-causes-bloat theory (FCBT) and crossover-bias theory (CBT) [34]. In FCBT two features are assumed to be common in a genetic search:

1. there exists a many-to-one mapping to the fitness space from the syntactic space, and

2. for any fitness value there exist exponentially more large individuals than small individuals with the same fitness.

The latter feature suggests that if a particular fitness is desired (for example the optimal value of zero RMSE) there is a tendency towards larger/more bloated individuals during the genetic search simply due to the fact that there are more of them within the search space [34].

In CBT the focus is instead shifted towards the crossover operator. The bloat is assumed to be a product of applying subtree crossovers on the population. While the average tree size is not changed during a subtree crossover, a number of small trees are created. Smaller trees in SR are less likely to exhibit good behaviours due to their size, hence the following selection operations are biased towards favouring the larger individuals [34]. Over generations this is effectively increasing the average tree size in the population, causing bloat.

2.4.2 Bloat Control

Regardless if a genetic program has established the cause of bloat or not, it is vital that it is controlled. There are many different ways of doing this. A simple solution which was proposed as a direct response to bloat is to set a maximum depth for the expression trees and to discard all individuals produced in genetic operations which exceed this depth [4]. While being easy to implement this method is introducing bias from the user when deciding on the maximum depth, which could prevent the program from finding the correct solution given that it is more complex then what is allowed. Also, by hard-capping the maximum depth it can have negative effects on the genetic diversity and the distribution of tree sizes. There are today numerous alternatives for controlling bloat, some of these methods are presented in the following sections.

2.4.2.1 Parsimony Pressure

The parsimony pressure method is in theory a quite simple and straight-forward solution to prevent bloat in GP. Even though there exists more modern and advanced methods, parsimony pressure is still the most used and widely established method [32]. It is targeting bloat caused by fitness, described in FCBT, by adding a penalty to the fitness of each individual depending on tree size. The resulting fitness f_{pp} is described as,

$$f_{pp}(\mathbf{x}) = f(\mathbf{x}) \pm \alpha \cdot l(\mathbf{x}), \tag{2.10}$$

with a minus sign if large fitness is defined as good or a plus sign if the reverse is true. $f(\mathbf{x})$ is the unweighted fitness, α is a constant known as the *parsimony coefficient* and $l(\mathbf{x})$ is the size of the individual [32].

The parsimony coefficient is critical in the implementation. Maintaining control of the bloat requires a precise and careful choice of α. One of the main critiques

of this method is that the choice of α is often done by trial and error, and the fact that one choice might not work equally well for different problems [32].

The choice of α has been studied, and some alternatives in which it is no longer constant has been developed. The alternatives contains methods for keeping the average number of nodes at a constant number, making it scale linearly throughout a run or making it follow a sinusoidal behaviour during a run [32]. While the produced results from respective implementation may differ, they all control bloat, preventing uncontrolled population growth. By changing the value of α throughout the simulation it is effectively reducing the impact of user-bias towards solution-complexity.

2.4.2.2 Limited Crossovers

As mentioned in section 2.3.5.2 there are different types of crossover operations. Assuming that the crossover operation is causing bloat, as is suggested in CBT, using different types of crossovers can be a potential form of bloat control technique. By using something like size-fair crossover, also referred to as limited crossover, where two individuals are only allowed to swap subtrees with a given maximum difference in size, the amount of bloat can be reduced. However, while it controls the growth of offspring, it will also prevents possible beneficial crossover operations from happening [31]. A possible work-around is to use a mix of different types of crossover techniques [30]. The resulting benefit is that all crossover operations are possible while still limiting the probability of bloat-causing ones. It has been proven that a mix of size-fair, homologous and standard crossovers together with some additional techniques can produce good results fitness-wise, although while displaying the same exponential tree size growth as for the standard crossover operation [30]. Mixed crossover is hence by itself not a successful bloat controlling technique.

2.4.2.3 Substituting a Subtree with an Approximate Terminal

The previously mentioned methods for controlling bloat focuses on the causes of bloat, FCBT and CBT respectively. Another method, which does not focus on what causes but rather how to control bloat, is *substituting a subtree with an approximate terminal* (SAT or SAT-GP in short). This method is more recent than earlier mentioned ones (proposed in 2017) and has shown promising results both regarding controlling bloat and overall performance of the program [33].

The idea behind SAT-GP is to select a group of the k largest individuals in each generation and to substitute a random subtree within them with an approximate terminal, with similar semantics. It is implemented as follows [33]: for each of the chosen individuals to reduce, choose a random node in the individual as the current subtree T_1. The node can be any node within the individual. Then, choose a random terminal X from the terminal set of the genetic program. The values of terminal X in all N data points are denoted as $S = (a_1, a_2, ..., a_N)$ and the values from evaluating subtree T_1 are denoted as $S_1 = (b_1, b_2, ..., b_N)$. The subtree T_1 is then to be replaced by a new subtree $T_2 = \theta^* \cdot X$, where θ^* is the coefficient that minimises the squared Euclidean distance between $\theta \cdot S$ and S_1. This equals minimising

$f(\theta) = \sum_{i=1}^{N}(a_i \cdot \theta - b_i)^2$, hence θ^* is given by,

$$\theta^* = \min_{\theta} \sum_{i=1}^{N}(a_i \cdot \theta - b_i)^2. \tag{2.11}$$

By calculating the derivative of $f(\theta)$ and equalling to zero, the optimal value is given by,

$$\theta^* = \frac{\sum_{i=1}^{N} a_i \cdot b_i}{\sum_{i=1}^{N} a_i^2}. \tag{2.12}$$

Thus, T_1 can be substituted by the approximate tree T_2.

Choosing the number of largest individuals to reduce, k, is a delicate task and is usually based on some percentage of the population size [33]. This percentage must account for the increase in solution sizes over generations. Otherwise, too many or too few individuals may have their size reduced, resulting in either too aggressive or too mild bloat control. In addition to reducing bloat SAT-GP may also reduce overly complex individuals by simplifying certain subtrees. Hence, aiding the genetic program in finding more parsimonious solutions [33]. However, the substitution might also hinder certain solutions from fully developing.

2.5 Results in Genetic Programming

Genetic programs produce a large number of candidate solutions over the course of a simulation. The main idea is that solutions will adapt over generations, creating increasingly more fit individuals. However, because of the adaptive nature of the program there is no guarantee that a solution will remain in the population throughout the entire run, rather it is unlikely that a single solution will survive for more than a fraction of the total generations. This highlights an important challenge with interpretation of the results in GP – it is not sufficient to search the final generation for solutions. Hence, the gathering of results must proceed throughout the whole genetic simulation.

Given that the collection of results is performed over the whole simulation the next challenge revolves around selection of the *best* solution. This depends on the specific project but often involves deciding on how to weight solution performance to parsimony. In real applications when there is noise and unknown features included in the input data, choosing solely based on fitness will most likely not provide the most desirable solution. A common approach to these two challenges is to create a Pareto frontier for each simulation. The Pareto frontier and how to interpret it is presented in the following sections.

2.5.1 Definition of the Pareto Frontier

The Pareto frontier, also known as the Pareto front or the Pareto set, is the set of parameterisations (allocations) that are all Pareto optimal. Pareto optimality is a

notion of optimality defined for multi-objective optimisation, where the objective function is represented as a vector rather than a scalar [15]. With a set of data X and a set Y defined as $y = f(x), y \in Y, x \in X$, the Pareto frontier can informally be defined as a set of points which are most feasible (either minimise or maximise $f(x)$ depending on what is wanted), i.e. for each given $x \in X$ the best y-value is in the Pareto frontier.

A more formal definition of the Pareto frontier $P(Y)$ is: Consider a system with a function $f : \mathbb{R}^N \to \mathbb{R}^M$, a compact set X of feasible decisions in the metric space $\mathbb{R}^N$ and where Y is the feasible set of criterion vectors in $\mathbb{R}^M$ such that $Y = \{y \in \mathbb{R}^M : y = f(x), x \in X\}$. Assume that the preferred directions of criteria is known, i.e. if y should be minimised or maximised, such that a point $y' \in \mathbb{R}^M$ is preferred over another point $y'' \in \mathbb{R}^M$. Then the Pareto frontier is written as,

$$P(Y) = \{y' \in Y : \{y'' \in Y : y'' > y', y'' \neq y'\} = \emptyset\}, \tag{2.13}$$

where the $>$ implies that y'' is preferred over y'.

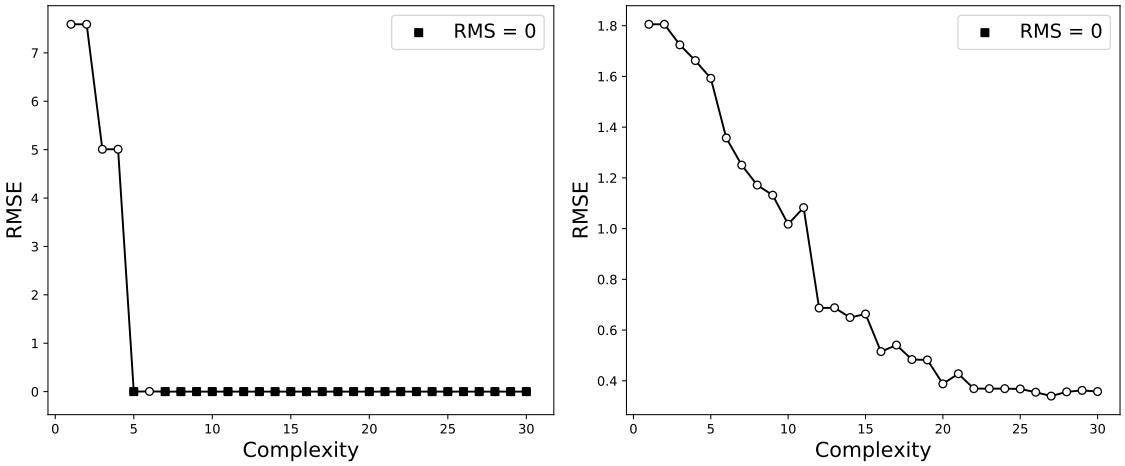
The Pareto frontier is a useful tool in several fields [35]. Since the frontier contains all Pareto optimal parameterisations it is used as a tool to study the trade-offs within a constrained set of parameters. It is for example useful when studying the error/deviation from a suggested symbolic function in symbolic regression versus its complexity.

2.5.2 Interpretation of the Pareto Frontier

Interpreting a Pareto frontier from a multi-objective optimisation problem is not as straightforward as from a conventional single-objective optimisation problem. This is because the Pareto frontier from a multi-objective optimisation problem contains several Pareto optimal solutions. In symbolic regression the accuracy and the complexity of a solution are both optimised. This is expressed as the error, for example RMSE, and the complexity in the Pareto frontier. Thus, the Pareto front for symbolic regression shows the most accurate solution for every complexity.

In symbolic regression some Pareto fronts are easier to interpret than others, visualised in figure 2.5. In figure 2.5a the optimal value for accuracy (RMSE) is exhibited by several solution complexities. A natural method for selecting the best solution is thereof to select the solution that has perfect accuracy with the lowest complexity. Thus, the most parsimonious solution with perfect accuracy is selected. However, the Pareto front in figure 2.5b does not provide any solution with perfect accuracy, making the selection of best solution more difficult.

There are many different methods used to facilitate the choice of best solution from the Pareto front. Results from previous works suggests that a prominent method is the technique for order of preference by similarity to ideal solution (TOPSIS in short) [36]. In TOPSIS the solution with the smallest Euclidean distance to the ideal solution and the largest Euclidean distance to the negative-ideal solution is chosen.



(a) **(b)**

Figure 2.5: Two examples of Pareto frontiers in symbolic regression. In 2.5a the RMS error is zero for complexity seven and higher, with the exception of complexity eight. In 2.5b none of the points in the Pareto frontier have an RMS error of zero.

The ideal solution is defined as a combination of the best value of each objective in the given Pareto set, i.e. the lowest RMSE and complexity. The negative-ideal solution is defined as the opposite, the combination of the largest RMSE and complexity.

TOPSIS is implemented as follows [36]: initially calculate a normalised objective matrix $\mathbf{F}$ with m (number of data points) rows and n (number of objectives) columns as,

$$F_{ij} = \frac{f_{ij}}{\sqrt{\sum_{i=1}^{m} f_{ij}^2}}. \tag{2.14}$$

A weighted normalised objective matrix v_{ij} is then constructed by multiplying each column in F_{ij} with a weight w_j,

$$v_{ij} = F_{ij} \cdot w_j, \;\; j = 1, 2, ..., n, \tag{2.15}$$

where the weight w_j is chosen by the user. The weights may be used if one or more objectives are deemed more important than others, but can be equal for all objectives if they are all deemed equally important. The element in vector $\mathbf{w}$ should sum up to one.

In the set J of minimisation objectives the ideal solution A^+ is determined as,

$$A^+ = \{min_i(v_{ij}), \;\; j \in J, \;\; i \in 1, 2, 3, ..., m\} = \left(v_1^+, v_2^+, ..., v_n^+\right), \tag{2.16}$$

and similarly the negative-ideal A^- as,

$$A^+ = \{max_i(v_{ij}), \;\; j \in J, \;\; i \in 1, 2, 3, ..., m\} = \left(v_1^-, v_2^-, ..., v_n^-\right). \tag{2.17}$$

With the ideal A^+ and negative-ideal A^- determined, the distance from each point in the Pareto set to respective ideal point can be calculated. The distances to A^+ are stored in S_+ as,

$$S_{i+} = \sqrt{\sum_{j=1}^{n}(v_{ij} - v_j^+)^2}, \quad i = 1, 2, ..., m, \tag{2.18}$$

and to A^- in S_- as,

$$S_{i-} = \sqrt{\sum_{j=1}^{n}(v_{ij} - v_j^-)^2}, \quad i = 1, 2, ..., m. \tag{2.19}$$

To lastly determine which of the solutions in the Pareto frontier to suggest, the closeness C of each solution is calculated and the smallest is chosen as,

$$\min_i \; C_i = \frac{S_{i+}}{S_{i-} + S_{i+}}. \tag{2.20}$$

2.6 Performance Enhancing Method for Genetic Programming

In section 2.2 the challenge of premature convergence within stochastic optimisation was presented. There are several methods for tackling this problem, however for symbolic regression using GP, methods for mitigating premature convergence often involve modification to the selection procedure. This is typically done by introducing some kind of mating restriction [15]. This can involve changing the parameters or changing the sampling method or, as was hinted in section 2.2, other methods such as the island model.

In the island model the whole population of N individuals is divided into N_s subgroups, each with a total of $N_g = N/N_s$ individuals. If no interactions are allowed between these subgroups, this is equal to running N_s independent simulations with a population size of N_g. However in the island model a certain amount of individuals may be allowed to migrate over to other populations and replace the worst individuals of that population. This is done in order to upset the evolutionary lineage by introducing other well performing but different solutions to the population. Since the model uses several different smaller populations it is less likely to get trapped in one specific local minimum, as each subgroup is independent of each other until migration occurs [15].

2.7 Sampling of Data Points

The quality of any analysis is dependent on the quality and abundance of input data. In many areas of engineering the amount of available data is large as a result of technological advances regarding collecting and organising data. Consequently, it is important to contemplate how and which parts of the input data should be used

in order to avoid bias and false predictions. In cases when dealing with large sets of data it is often necessary to sample subsets of the entire data set in order for the analysis to become viable to perform regarding computational load.

Many machine learning algorithms perform optimally when there is a large data set available to train on [16], meaning more data points enhance the accuracy of the algorithms. When large amounts of data is available the bottleneck that arise is the computational time required to use and/or train an algorithm. The time is dependent on the computational resources available and the computational complexity of the algorithm, which is dependent on the number of data points.

When there are more data points available than what can be used due to time and computational limitations, there are different ways of choosing which ones to include. The relative importance of experimental data points has been studied, concluding that some data points might be of more importance than other [37].

There are two main types of sampling methods: probabilistic sampling techniques and non-probabilistic sampling techniques [38]. The most simple probabilistic sampling technique is random sampling, where data points is simply chosen at random. It is easy to implement and to understand but the the main drawback of the technique is that it can not guarantee representation of the whole data set [38]. There are many other techniques which may increase representation of the whole data set, but each come with other drawbacks such as being harder to implement or introducing bias.

2.7.1 The Sampling Theorem

Another important aspect of sampling arises when it comes to sampling signals. As stated in the sampling theorem [39]:

Theorem 1 (The Sampling Theorem)
A continuous bandlimited signal is uniquely determined by its values at uniform sampling points if the sampling frequency is greater than twice the maximal frequency of the signal.

This frequency is also referred to as the critical frequency and is simply defined as

$$f_s > 2 \cdot B \tag{2.21}$$

where B is highest frequency of the signal. Hence the sampled points from a signal must be sampled with a rate of at least $2 \cdot B$ and for the sampling theorem to hold they must be sampled uniformly.

3

Development and Tuning of Genetic Program

This chapter revolves around implementation of concepts presented in chapter 2, presenting a detailed description of the finished algorithm. In addition, an enhanced version of SAT (substituting a subtree with an approximate terminal) is presented followed by a comparison of different bloat controlling techniques and how choice of sampling method impacts algorithm performance as well as the size distribution of individuals of the simulation. Lastly the implementation of TOPSIS (technique for order of preference by similarity to ideal solution) is presented with an added feature for ensuring optimal proposal solution.

3.1 General Outline

The overall algorithm is schematically illustrated in figure 3.1. It consists of a slightly altered version of the conventional genetic algorithm presented in figure 2.1 in section 2.3.1. The algorithm first initialises a population and then enters the main genetic loop, repeating fitness evaluation, selection, breeding and an enhanced version of SAT until termination, whereby the result is summarised. In the following sections each algorithm module is described in turn.

3.1.1 Initial Population

Individuals are created using expression trees. Each individual has an attributed size and a root node. The size and the root node correspond to the individual's complexity and bottom level node of its expression tree, respectively. The root node has 0, 1 or 2 children depending on the arity of its expression. The root's children in turn also have children, creating a data structure for representing expression trees. Creation of individuals is done using either grow or full methods with uniformly distributed function and terminal selection. The function set can consist of any subset of the following functions,

$$\mathcal{F}_{\text{set}} = \{+, -, \div, \times, \sin, \cos, \exp, \log, \text{sqrt}\}, \tag{3.1}$$

and the terminal set is determined by the input data according to,

$$\mathcal{T}_{\text{set}} = \{\pi, C, x_1, x_2, ...\}, \tag{3.2}$$

where C represents numerical constants and $x_1, x_2, \ldots$ are input variables. Numerical constants are uniformly initialised ranging from 0 to 5. Initialisation of all populations is done using ramped half-and-half initialisation. To account for uneven partitioning all residual individuals are generated using grow method with the maximum allowed depth.

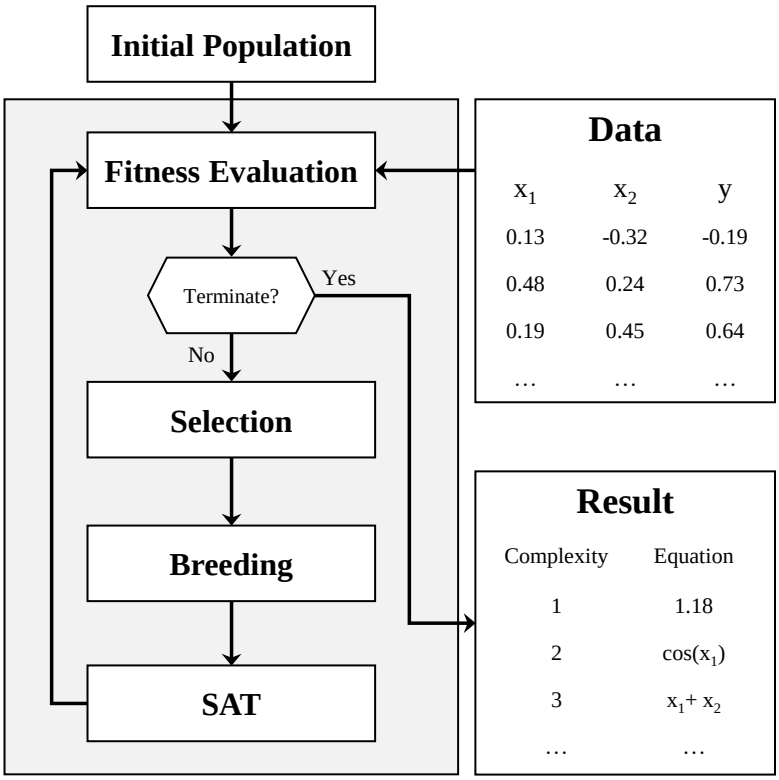


Figure 3.1: Schematic illustration of the final algorithm outline. Each box corresponds to an algorithm module responsible for some vital process.

3.1.2 Fitness Evaluation

Fitness evaluation is done by converting each expression tree to postfix notation, using postorder traversal, and evaluating the corresponding expression string. The implemented fitness measure is RMSE and evaluation is performed using the supplied data points from the input. The RMSE is a form of natural standardised fitness and is converted to adjusted fitness and then normalised over all individuals before selection. Individuals with expressions containing undefined operations such as division with zero, are given an adjusted fitness equal to zero, in order for them to avoid being selected for reproduction.

3.1.2.1 Termination

Termination of the genetic loop is done after a user-specific number of generations have been completed. The number of generations before termination varies depending on the purpose of the simulation. The gradient of the learning curve averaged

over several simulation runs can be used in order to judge the sufficiency of the number of generations. A flattened average learning curve indicates that the progression of solution accuracy has stalled, making additional generations redundant. If this is the case it is more beneficial to change other hyperparameters such as population size or ratios of genetic operators for the performance of the algorithm.

3.1.2.2 Data

The input data consists of input variables and corresponding target values. The number of data points used for evaluation of fitness has an important effect on the quality of the result but also the computational load. Thus, the number of fitness cases must account for both of these factors. Current implementation allows for up to 9 input variables.

3.1.3 Selection

Five selection schemes are implemented: elitism, roulette, linear rank, tournament and Boltzmann selection. The initial step of selection is to determine the number of individuals that has to be sampled for each genetic operation. This is determined by user-defined ratios and involves splitting the total number of individuals into three partitions, corresponding to replication, crossover and mutation. The partitions for crossover and mutation are sampled using the selected sampler while replication uses elitism selection.

3.1.4 Breeding

Breeding involves replication, crossover and mutation. Replication is done by simply copying individuals. Two crossover operations are implemented: free and size-fair crossover. Based on the number of individuals selected for crossover the user may define the ratios for free and size-fair crossovers. Three mutation operations are implemented: point, branch and constant mutation. Point mutation selects a random node and replaces it with a different node with the same arity in order to avoid syntactical errors. Branch mutation selects a random subtree and replaces it with a newly generated tree using grow method with maximum depth of 2. Constant mutation scales a numerical constant with scaling factor α according to,

$$\alpha = 10^{\beta}, \tag{3.3}$$

where β is uniformly distributed between 0 and 1, making the scaling factor assume values between 0.1 and 10, with equal probability of reducing and increasing a numerical constant's value.

3.1.5 Enhanced and Adaptive SAT

In SAT-GP a random subtree is substituted with a new small subtree consisting of a constant, θ, multiplied with a random terminal, X, from the terminal set. The choice of terminal is selected randomly from the terminal set and is followed by

finding the constant which minimises the Euclidean distance between the old and the new subtree.

An enhanced version of SAT is implemented. Instead of selecting a random terminal for substitution the selection of terminal is performed based on every terminal's optimal Euclidean distance to the to-be substituted subtree. The algorithm works as follows: for all available terminals, X_i, in the terminal set, $\mathcal{T}_{\text{set}}$, perform least square optimisation to find every terminal's respective optimal constant θ_i^* that minimises the Euclidean distance between $\theta_i^* \cdot X_i$ and the subtree selected for substitution. If X_i is a numerical constant the corresponding optimal constant θ^* is equal to the mean of the selected subtree. The algorithm then compares the Euclidean distance between each subtree $\theta_i^* \cdot X_i$ and the to-be substituted subtree and selects the optimal subtree for substitution. Thus, the best substitution is performed. If the optimal substitution is a numerical constant the substituted subtree $\theta^* \cdot C$ is reduced to C^*, where $C^* = \theta^*$. The selection of subtree for substitution is only applied to non-leaf nodes.

Selecting the ratio for largest individuals to reduce with SAT-GP, k, has to be done in such a way that it prevents bloat while still avoids too aggressive size-reduction. Because of the stochastic nature of a genetic algorithm the sizes of individuals will change, sometimes drastically, over a simulation. This implies that a certain value for k might not be suitable for an entire run. This motivated the development of a method for adaptive k.

The adaptive-k method starts with k at 1% and whenever the average tree size increases after one generation, k is increased with 0.25%. If the average tree size instead decreased, k is decreased by 0.5%. The method also contains a user-chosen cap which should prevent runaway individual sizes. If the average tree size passes this cap (such as 20, 30 or 40 average complexity) then k is instead increased by 1%. This adaptive method is developed to maintain a balanced increase/decrease of the average tree size. It does still allow for trees to grow bigger than the cap, but will over time try to simplify them more aggressively. In comparison to other bloat controlling techniques it does not affect choice of parents in breeding, put a hard cap on the tree sizes or affect which subtrees to swap in crossovers. However, the choice of cap-value may infer some bias but is necessary in order to avoid issues regarding computational load. The mentioned values of k are obtained by trial and error.

3.2 Studying Bloat Control & Samplers

In the following sections different bloat controlling techniques are studied as well as how different samplers impact the resulting error progression and size distribution of individuals. For this purpose a target function is constructed with the intent to provide a similar challenge as for a problem which might be encountered in real applications. The resulting function has close resemblance to Newton's equation for gravitational force with an added term according to,

$$f(\mathbf{x}) = C_1 \frac{x_1 x_2}{x_3^2} + C_2 \sqrt{x_4}, \tag{3.4}$$

where $x_1, x_2, x_4 \in \mathcal{U}\{0, 10\}$, $x_3 \in \mathcal{U}\{5, 15\}$, $C_1 = 13.10$, $C_2 = 2.68$ and using one unrelated variable $x_5 \in \mathcal{U}\{0, 10\}$. The function has in this form a complexity of 14 and should provide a comparable reference for future input functions. From this function 60 random points are generated and used as the data set in the following comparisons.

3.2.1 Bloat Controlling Techniques

Two of the theories trying to explain what is causing bloat in GP, presented in section 2.4, are crossover-bias-theory (CBT) and fitness-causes-bloat-theory (FCBT). Since bloat may be caused by one or several different phenomenons there are varying approaches to prevent bloat. Using size-fair crossovers is one way of targeting the cause of bloat as described in CBT, while parsimony pressure is a method targeting fitness as the cause of bloat as described in FCBT. There are also other techniques whose main purpose is to reduce bloat, ignoring what is causing it, such as SAT. In order to investigate different techniques for bloat control a number of tests are performed. Initially varying crossover operations are studied, followed by a comparison of different parsimony pressures and lastly two constant k-values for the enhanced implementation of SAT are compared to the adaptive-k method in figures 3.2, 3.3 and 3.4, respectively. The best performing methods in terms of bloat control are then studied further in figures 3.5 and 3.6. Between different runs the majority of parameters and settings are kept constant only varying the bloat controlling method. All parameters and settings can be found in table 3.1.

In figure 3.2 results from three different crossover settings are presented. Free refers to 100% of crossovers being unlimited, limited refers to 100% of crossovers being size-fair with a maximum allowed difference of 1 and mixed refers to mixing unlimited and limited crossovers with 50% of each. Free and mixed are both experiencing a large increase in average tree size, corresponding to the exponential tree growth mentioned in section 2.4 and are clear examples of bloat since the error progression does not improve accordingly. They do however result in slightly lower RMSE than limited. Regarding bloat control it is clear that only the limited crossover alone achieves any acceptable level of bloat control. Since mixed crossovers produces less bloat than free and similar RMSE, it is chosen as the standard for crossovers used for all further investigations of bloat controlling techniques, samplers etc.

Three different examples of parsimony pressure are presented in figure 3.3 corresponding to three different values of the penalty constant α. The penalty is described in equation (2.10) in section 2.4.2.1. For both $\alpha = 0.1$ and $\alpha = 0.01$ the average tree sizes are relatively constant, both controlling bloat well. For $\alpha = 0.001$ the average tree size increases initially but is more or less constant after 600 generations. Since the target function has complexity 14 the desired average tree size should be close-to or slightly higher than 14 for the search to successfully explore the function space around the correct complexity. For $\alpha = 0.1$ the average tree size is

Table 3.1: Program settings for investigation of different bloat controlling techniques. An additional cap terminating a run if the average number of nodes exceeds 1000 is also used for all runs.

Setting	Description
Population size	1000
Generations	1000
Initialisation	Ramped Half-and-Half with 9 levels of maximum depth
Function set	$\{+, -, \times, \div, \exp, \log, \sin, \cos, \sqrt{}\}$
Terminal set	$\{\pi, C, \mathbf{x}\}$
Fitness measure	RMSE
Selection	Roulette
Target Function	$C_1 \frac{x_1 x_2}{x_3^2} + C_2\sqrt{x_4}$
Data points	60
Replication	1%
Mutation	29% (point: 25%, branch: 25%, constant: 50%)
Crossover	70% (varying composition of free and size-fair)
Parsimony Pressure	Yes (varying α)
Enhanced SAT	Yes (varying k and with adaptive-k using cap 20)

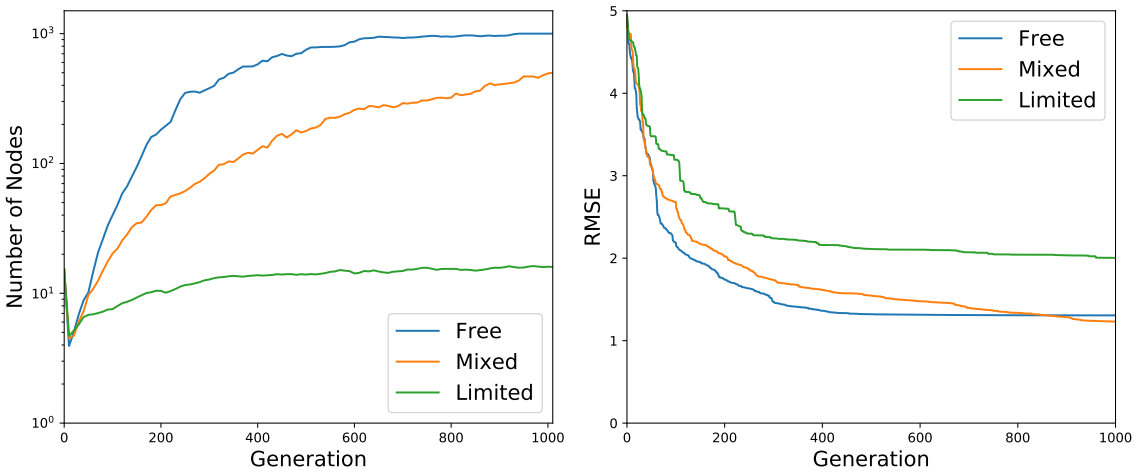


Figure 3.2: The average tree sizes and best solution over time for three different crossover settings. The data is the average of 10 independent runs with different random seeds.

around 9 for most generations, for $\alpha = 0.01$ it is approximately 15 and for $\alpha = 0.001$ it is between 50 and 70 for the last 400 generations. Hence, neither $\alpha = 0.1$ nor $\alpha = 0.001$ controls bloat as desired, while $\alpha = 0.01$ performs relatively well. One important thing to note is that the values for α are deduced by trial and error.

Equation (2.10), describing the parsimony pressure, implies that α may cause the penalty for complex individuals to have great impact on the resulting fitness f_{pp}. If the fitness measure is significantly smaller than the parsimony term the resulting altered fitness will be dominated by the parsimony pressure and vice versa. Since the magnitude of the unweighted fitness f may vary, there is therefore no guarantee that the studied values of α would affect average tree sizes equally for other target functions.

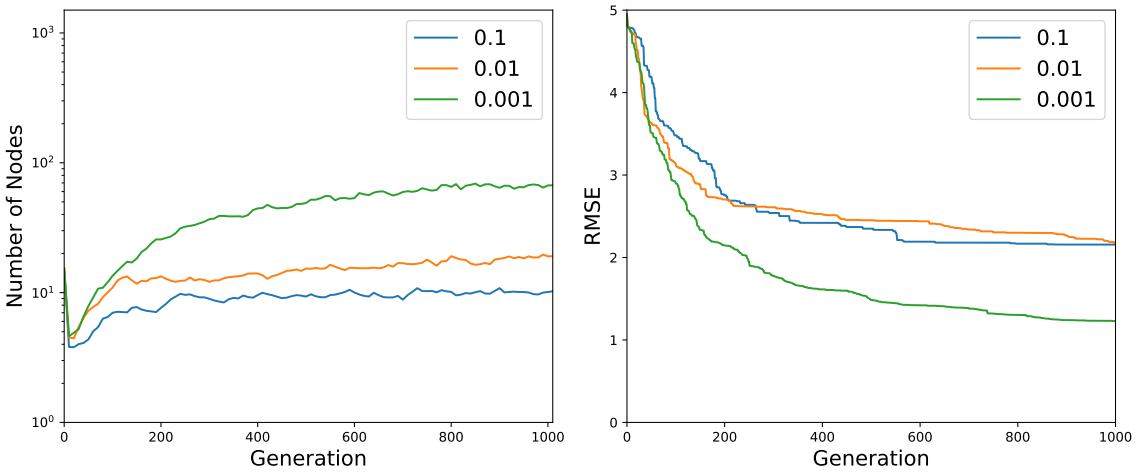


Figure 3.3: The average tree sizes and best solution over time for three different values parsimony penalty α. The data is the average of 10 independent runs with different random seeds. The crossovers are mixed; 50% free and 50% size-fair.

Figure 3.4 is displaying three cases of the enhanced SAT methods; two with constant k-values and one using adaptive-k method, described in section 3.1.5. For $k = 5\%$ the number of nodes decreases rapidly and is kept at a very low level, resulting in a high RMSE. For $k = 3\%$ the number of nodes appear to have some sporadic increases and decreases until generation 600 when a constant increase can be seen. For the adaptive k the number of nodes is kept close-to constant below 20 (since the SAT cap is set at 20). Both the adaptive and $k = 3\%$ perform well in terms of RMSE but only the adaptive one manages to control bloat as desired.

From figures 3.2, 3.3 and 3.4 it is concluded that the lowest RMSE values are achieved using mixed crossovers with a parsimony pressure constant of $\alpha = 0.001$ or with only free and mixed crossovers, which all have average tree sizes much larger than desired. This result is inline with the second feature in FCBT, described in section 2.4.1, which states that there exists exponentially more large individuals than small individuals with the same fitness. However, a solution with orders of magnitude higher complexity than the true complexity is not acceptable making these three setups unsuitable for further use. From the figures it is also concluded that mainly three of the total nine setups are successfully controlling bloat in a desired way: using limited crossovers, using a combination of mixed crossovers with parsimony pressure constant with $\alpha = 0.01$ and using a combination of mixed crossovers with adaptive-k for the enhanced SAT. To compare these three setups their average tree sizes and best solution over time are presented together with the addition of

areas showing the standard deviation in figures 3.5 and 3.6.

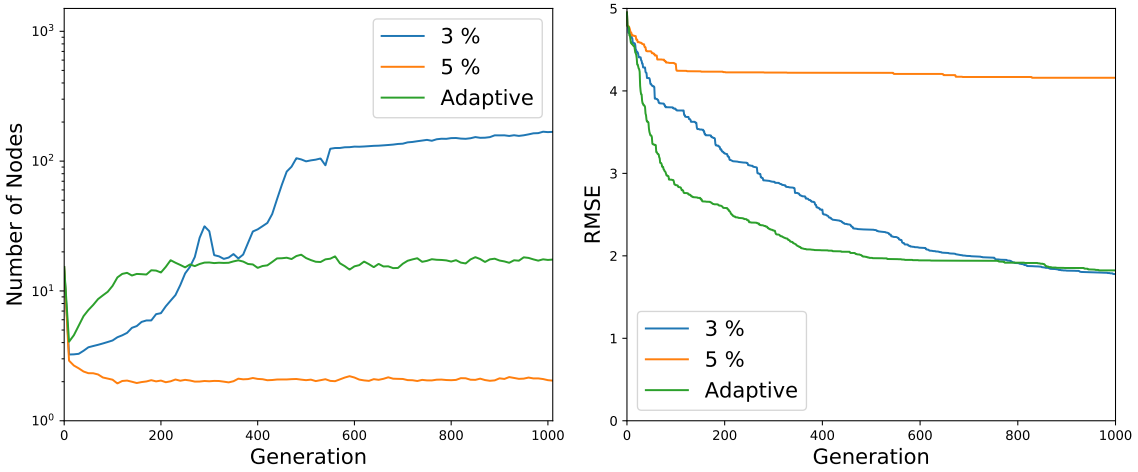


Figure 3.4: The average tree sizes and best solution over time for the enhanced SAT method, for two constant k values and using adaptive-k. The data is the average of 10 independent runs with different random seeds. The crossovers are mixed, 50% free and 50% size-fair.

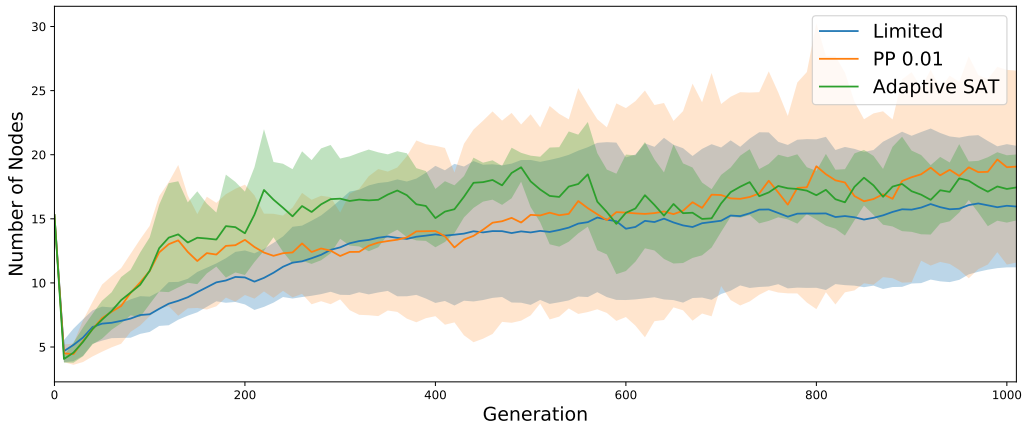


Figure 3.5: The average tree sizes for the three most well-performing methods in terms of bloat control. For each method the average of 10 independent runs with different random seeds are shown together with semi-transparent areas representing the standard deviation. The areas do not represent distribution of sizes within populations, only the average number of nodes between runs.

Both in figure 3.5 and 3.6 SAT using adaptive-k has smaller standard deviation during the majority of the generations in comparison to the other two setups. Not only is it controlling bloat in a more consistent way over multiple runs, it is also achieving lower RMSE solutions on average and with a smaller variance between runs. Additionally using only size-fair crossovers is limiting the genetic operations, while the chosen $\alpha = 0.01$ might not work equally well for other target functions. The enhanced SAT method with adaptive-k in combination with mixed crossovers is therefore chosen as the bloat controlling setup to be used for all further tests.

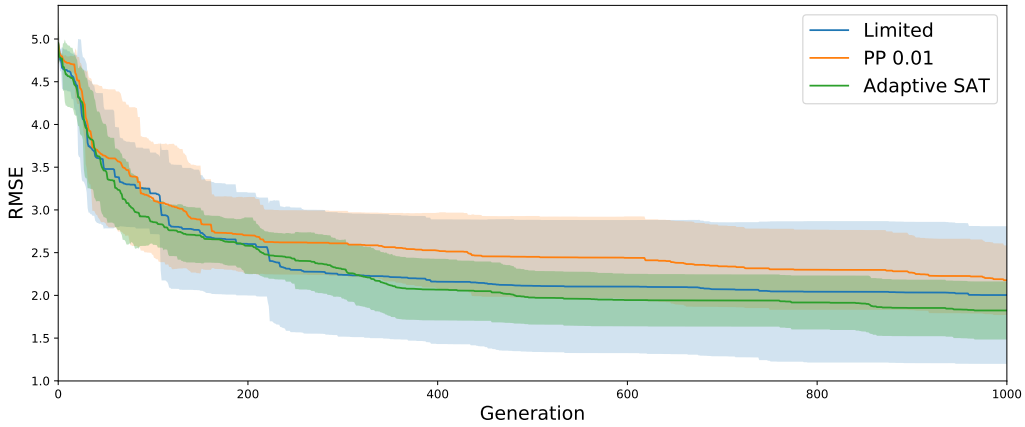


Figure 3.6: The best solution over time for the three most well-performing methods in terms of bloat control. For each method the average of 10 independent runs with different random seeds are shown together with semi-transparent areas representing the standard deviation.

3.2.2 Samplers

The sampler used for selection determines the balance between exploitation and exploration. As mentioned in section 2.3.4, there are numerous samplers which samples individuals for breeding in different ways. The difference between samplers is easy to understand although the impact they have on the population can be difficult to predict due to the complex dynamics resulting from the interactions of different components of the genetic algorithm.

In order to investigate how different samplers impact the progression of solution accuracy and size distribution of individuals, simulation are performed using different samplers. In the tests roulette, linear rank and three setups of Boltzmann selection are investigated. The motivation for choice of samplers and parameters for Boltzmann selection is further developed in appendix A. In short roulette is chosen due to its high selective pressure, linear rank due to its low selective pressure and Boltzmann due to the potential of having the selection pressure change throughout the simulation. Each selection method is tested on the same data for 50 simulation runs, with settings according to table 3.2. The result is presented in figures 3.7-3.11, which presents the error progression and complexity distribution of the cumulative result over all simulations for each sampling scheme. In each figure the left graph presents the error progression while the right graph shows the cumulative size distribution for all simulations. The intent of each simulation is not to find the correct solution but instead to highlight the dynamics of the different samplers. By running several simulation on the same sampling method certain sampler-specific features can be distinguished guiding the choice of sampling method for future simulations.

The result from roulette selection is shown in figure 3.7. The average RMSE at the end of a simulation is 1.25 with a somewhat flattened learning curve. The size distribution shows, compared to the upcoming selection methods, a scattered dis-

Table 3.2: Program settings for investigation of sampler's impact on error progression and size distribution of individuals.

Setting	Description
Population size	1000
Generations	2000
Initialisation	Ramped Half-and-Half with 9 levels of maximum depth
Function set	$\{+, -, \times, \div, \exp, \log, \sin, \cos, \text{sqrt}\}$
Terminal set	$\{\pi, C, \mathbf{x}\}$
Fitness measure	RMSE
Selection	Roulette, Linear Rank, Boltzmann ($T_0 = \{150/N, 250/N, 350/N\}$)
Target Function	$C_1 \frac{x_1 x_2}{x_3^2} + C_2\sqrt{x_4}$
Data points	60
Replication	1%
Mutation	29% (point: 25%, branch: 25%, constant: 50%)
Crossover	70% (free: 50%, fixed: 50%)
Parsimony Pressure	No
Enhanced SAT	Yes (with adaptive-k using cap 40)

tribution with two blurry peaks around complexities 1 and 35. There also seems to be a non insignificant amount of individuals with intermediate complexities between these two peaks. The size distribution over generations assumes its final shape after approximately 20% of the total simulation, showing only minor changes afterwards. The presence of intermediate solutions shows that the sampler is searching for solutions with different complexities.

Linear rank selection has the lowest average final RMSE of all samplers tested, 0.95. The result is presented in figure 3.8. Although linear rank selection exhibits the best result regarding average error it also has the most flattened learning curve out of all samplers, indicating that its adaptive momentum has reduced significantly towards the end of the simulation. This is true for all samplers as it becomes increasingly more difficult to improve upon the current best solution. The size distribution shows a distinct peak around complexity 30 and assumes its final form after approximately 20% of the simulation run. The size distribution indicates that the program is performing a meticulous search around the peak complexity.

The three setups of Boltzmann selection tested are using different values for the initial temperature, T_0, according to $150/N$, $250/N$ and $350/N$, where N is the population size. Values for parameters α, β and γ are presented in appendix A (see equation (2.7) for reference). The results are presented in figures 3.9-3.11. The

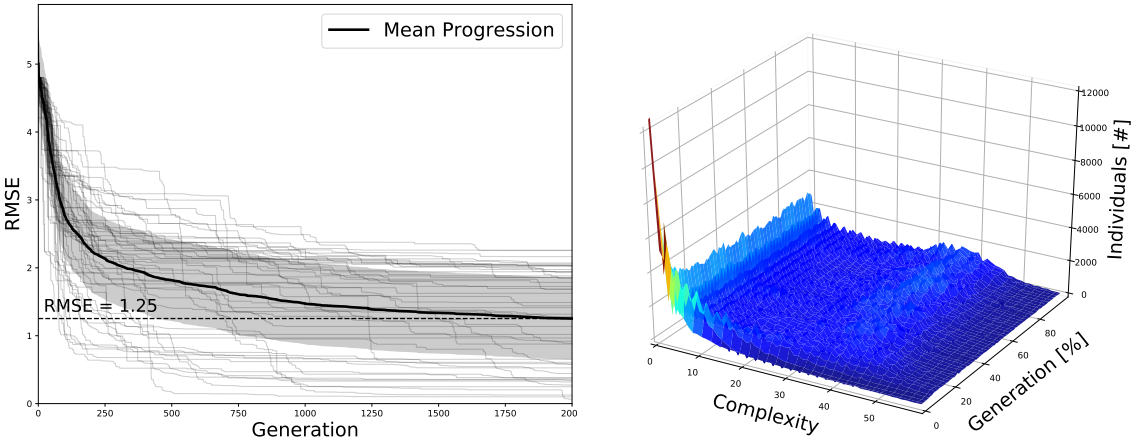


Figure 3.7: Error progression and complexity distribution for roulette selection on 60 data points from target function 3.4. The graphs show the accumulative results from 50 simulations each using a population size of 1000 running for 2000 generations.

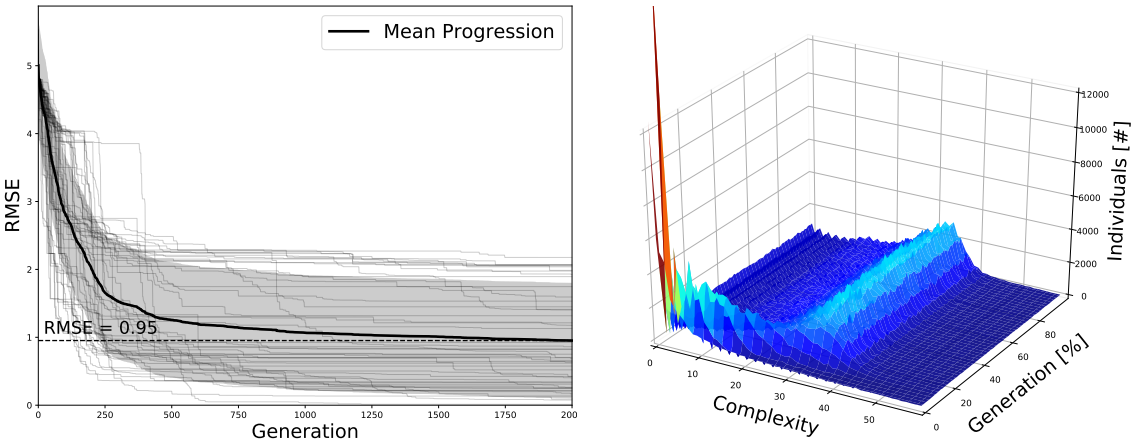


Figure 3.8: Error progression and complexity distribution for linear rank selection on 60 data points from target function 3.4. The graphs show the accumulative results from 50 simulations each using a population size of 1000 running for 2000 generations, settings are presented in table 3.2.

average final RMSE is lowest for $T_0 = 150/N$ and highest for $T_0 = 350/N$, implying that the lower initial temperature results in the best algorithm performance. In all three figures the two phases of exploration and exploitation are visible when examining the size distributions. Exploration occurs in early stages of the simulation when the selection pressure is low and the genetic diversity is high. This can be seen in the figures where individuals in early stages have lower complexities, which based on the second feature assumed in FCBT (section 2.4.1), indicates that the individuals are not selected based on fitness but rather genetic differentiation. The selection pressure is gradually increased as the Boltzmann temperature is decreased, successively promoting exploitation over exploration. Towards the end of all three simulations the size distribution have changed dramatically compared to the initial structure, exploiting the best individuals with larger complexities which, again ac-

cording to the second feature assumed in FCBT, is expected. The gradual change in temperature, resulting in the change from low to high selection pressure, also changes the appearance of the error progression. For Boltzmann the improvement in solution accuracy is more protracted compared to roulette and linear rank selection, implying that the initial process of the simulation is more focused on creating genetic diversity than finding the optimal solution.

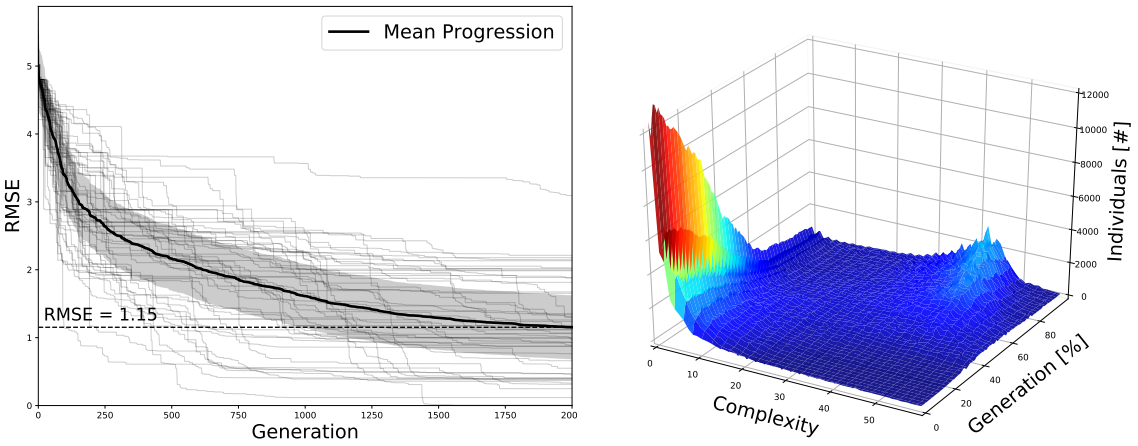


Figure 3.9: Error progression and complexity distribution for Boltzmann selection ($T_0 = 150/N$) on 60 data points from target function 3.4. The graphs show the accumulative results from 50 simulations each using a population size of 1000 running for 2000 generations, settings are presented in table 3.2.

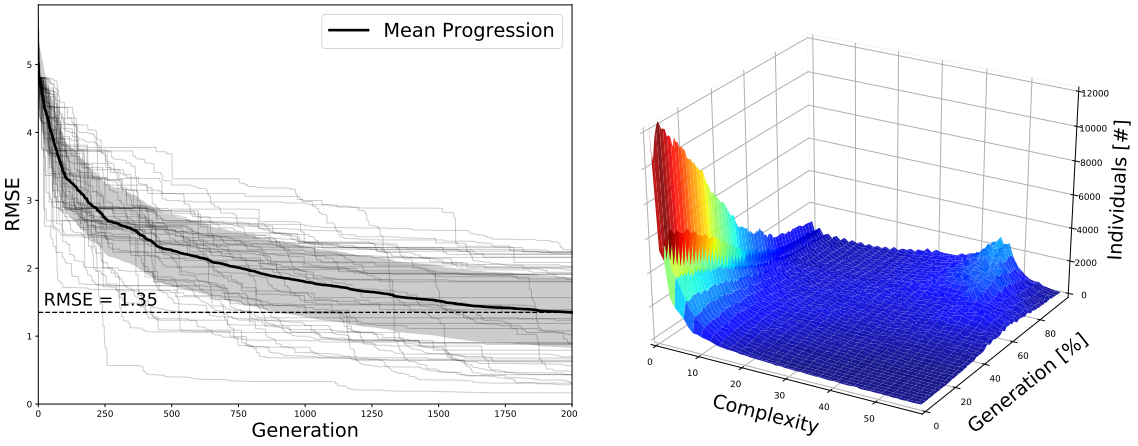


Figure 3.10: Error progression and complexity distribution for Boltzmann selection ($T_0 = 250/N$) on 60 data points from target function 3.4. The graphs show the accumulative results from 50 simulations each using a population size of 1000 running for 2000 generations, settings are presented in table 3.2.

The three tested selection schemes shows three different approaches to selection with similar results. An important distinction between Boltzmann and both roulette and linear rank selection is that in Boltzmann selection the number of maximum generations will affect exploration and exploitation. If the number of generations is

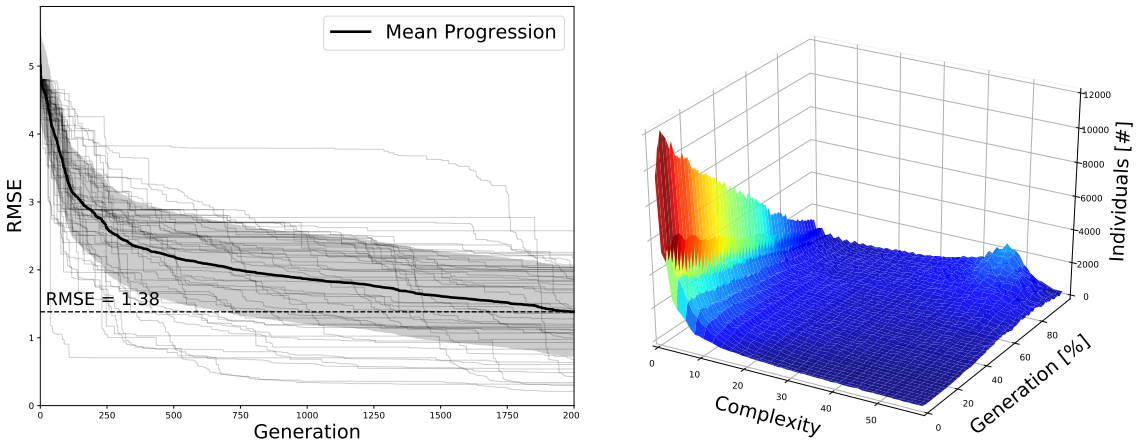


Figure 3.11: Error progression and complexity distribution for Boltzmann selection ($T_0 = 350/N$) on 60 data points from target function 3.4. The graphs show the accumulative results from 50 simulations each using a population size of 1000 running for 2000 generations, settings are presented in table 3.2.

increased the resulting exploration and exploitation phases will increase. However, for roulette and linear rank selection increasing the maximum allowed generations will simply extend the progression, leaving the previous progression unchanged. This is something which might change the performance of the samplers when applied using different settings.

The investigation on how roulette, linear rank and Boltzmann selection impacts the performance of the overall algorithm has shown similar results for all samplers regarding RMSE but not size distribution. This is an indication that the three different approaches are all viable but might have performance variations depending on the specific input data and algorithm settings. In the following results all simulations are performed using roulette, linear rank and Boltzmann selection with initial temperature $T_0 = 150/N$ (due to $T_0 = 150/N$ showing most promising results of the Boltzmann selection setups).

3.3 Choosing Solution in Pareto Frontier

The results generated in a multi-objective optimisation such as SR using GP are often presented in a Pareto frontier, which was presented and explained in section 2.5.1. One of the methods for choosing one of the solutions contained in the Pareto set is TOPSIS (technique for order of preference by similarity to ideal solution). In short one is creating a matrix $\mathbf{F}$ by normalising each input objective, here complexity and RMSE, and storing each vector as a column. Each column j is then multiplied by a weight w_j, which can be varied if one objective is considered more or less important. In the case of SR both error and complexity are of similar interest, hence the weight vector $\mathbf{w} = \frac{1}{2}$ (1,1).

Since the Pareto frontier consists of all Pareto optimal solutions, there is no *correct*

choice of solution. However, in the results from SR one of the two objectives may reach its optimal value; the RMSE may reach zero. An RMSE of zero corresponds to a solution which describe the dynamics of the system perfectly (ignoring possible overfitting), but might not be the most parsimonious one. To make sure these solutions are also presented to the user a small alteration of the TOPSIS algorithm is made, always presenting the most parsimonious solution with zero RMSE if there exists any. The rest of TOPSIS is implemented as described in section 2.5.1. An example of when TOPSIS would suggest another solution than the one with zero RMSE is shown in figure 3.12.

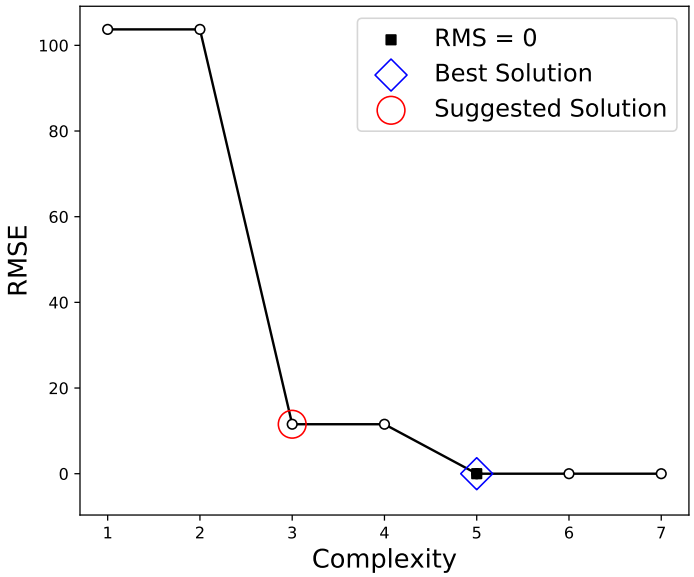


Figure 3.12: Example of a Pareto frontier where TOPSIS does not suggest the most parsimonious solution with zero RMSE, marked as *Best Solution*.

4
Results

In this chapter the results from two different tests are presented. The first test intends to evaluate the algorithm's ability to identify exact expressions based on input data from 50 equations from physics [40]. The result is then compared with two other algorithms' performance on the same equations. The second test includes simulations done with input data from a ventilator, where the objective is to identify a mathematical expression for the air flow using different subsets of the input data consisting of time, pressure and tidal volume.

4.1 Feynman Equations

To test how well the developed genetic program can perform symbolic regression (SR) a number of equations are needed. Also, to put the performance into context, the results should ideally be compared to the corresponding results from other programs.

There are infinitely many equations that can be used to test the algorithm. Equations are used in far more areas than just physics, but since the authors of this thesis are aspiring physicists, there are few examples more compelling than the equations from the famous Feynman lectures on physics [40]. In the lectures Richard Feynman derives a large number of equations from various areas of physics, some are simple while others are more complex. The content of the lectures vary a lot but are divided into three volumes; *I: Mainly mechanics, radiation and heat*; *II: Mainly electromagnetism and matter*; and *III: Quantum mechanics* [40]. Each volume consists of several chapters containing numerous equations. The denotation used in this thesis for equation 10 in chapter 15 of volume *II* is II.15.10.

The performance of two different methods for SR have previously been compared for 100 Feynman equations – comparing a newly developed physics-inspired method for SR, called AI Feynman[1] with one of the best commercial GP softwares, called Eureqa [41]. The data set used for the 100 equations is public and is used in this thesis.[2] It should be pointed out that the authors of AI Feynman have chosen which

[1]The model consists of a series of modules trying to exploit properties of data such as smoothness, symmetry, separability, compositionality, units and low order polynomial components. Common features in physics which are used to derive expressions.

[2]All data sets used for the Feynman equations are available at: `https://space.mit.edu/home/tegmark/aifeynman.html`

100 Feynman equations to perform the comparison on.

Because of the limited scope of this thesis the first 50 applicable equations in the mentioned comparison is used as test set to study the performance of the developed genetic program. In the following section the result from the study is presented, comparing the created algorithm with AI Feynman and Eureqa. Afterwards, the performances of roulette, linear rank and Boltzmann sampling are compared.

4.1.1 Comparison to AI Feynman and Eureqa

The program settings used for producing the results in this section are found in table 4.1. The result for each equation includes 30 total runs, 10 for each sampler (roulette, linear rank and Boltzmann). If one or several of the total 30 runs finds the correct equation (RMSE = 0 and lowest achievable complexity) the result is considered successful. The results from testing the developed program on the 50 different Feynman equations are presented in table 4.2, together with the results for AI Feynman and Eureqa. For every equation in the table a figure showing error progression and the Pareto frontier is found in appendix C.

Table 4.1: Program settings for testing the developed program on the 50 Feynman equations (and ventilator data). For each equation 10 independent runs with each sampler is executed, corresponding to using the island model without interactions.

Setting	Description
Population size	2000
Generations	5000
Initialisation	Ramped Half-and-Half with 9 levels of maximum depth
Function set	$\{+, -, \times, \div, \exp, \log, \sin, \cos, \text{sqrt}\}$
Terminal set	$\{\pi, C, \mathbf{x}\}$
Fitness measure	RMSE
Selection (parameters)	Roulette, Linear Rank, Boltzmann ($T_0 = 150/N$)
Target Function	50 Feynman equations
Data points	100
Replication	1%
Mutation	29% (point: 25%, branch: 25%, constant: 50%)
Crossover	70% (free: 50%, fixed: 50%)
Parsimony Pressure	No
Enhanced SAT	Yes (with adaptive-k using cap 20)

Table 4.2: The 50 Feynman equations tested. All equations are derived in the Feynman lectures on physics and follows the denotation of *Volume / Chapter / Equation Number*. AI-F denotes AI Feynman, a physics inspired method for SR. Eureqa is a commercially available GP software. Thesis denotes the program developed in this thesis. "Yes" indicates that the correct equation in terms of accuracy and parsimony has been found.

Feynman Eq.	**Equation**	**AI - F**	**Eureqa**	**Thesis**
I.6.20a	$f = e^{\frac{-\theta^2}{2}}/\sqrt{2\pi}$	Yes	No	Yes
I.6.20	$f = e^{\frac{-\theta^2}{2\sigma^2}}/\sqrt{2\pi\sigma^2}$	Yes	No	No
I.6.20b	$f = e^{\frac{-(\theta-\theta_1)^2}{2\sigma^2}}/\sqrt{2\pi\sigma^2}$	Yes	No	No
I.8.14	$d = \sqrt{(x_2-x_1)^2+(y_2-y_1)^2}$	Yes	No	No
I.9.18	$F = \frac{Gm_1m_2}{(x_2-x_1)^2+(y_2-y_1)^2+(z_2-z_1)^2}$	Yes	No	No
I.10.7	$m = \frac{m_0}{\sqrt{1-v^2/c^2}}$	Yes	No	No
I.11.19	$A = x_1y_1 + x_2y_2 + x_3y_3$	Yes	Yes	Yes
I.12.1	$F = \mu N_n$	Yes	Yes	Yes
I.12.2	$F = \frac{q_1q_2}{4\pi\epsilon r^2}$	Yes	Yes	Yes
I.12.4	$E_f = \frac{q_1}{4\pi\epsilon r^2}$	Yes	Yes	Yes
I.12.5	$F = q_2E_f$	Yes	Yes	Yes
I.12.11	$F = q(E_f + B\nu sin(\theta))$	Yes	Yes	Yes
I.13.4	$K = \frac{1}{2}m(v^2+u^2+w^2)$	Yes	Yes	Yes
I.13.12	$U = Gm_1m_2(\frac{1}{r_2}-\frac{1}{r_1})$	Yes	Yes	Yes
I.14.3	$U = mgz$	Yes	Yes	Yes
I.14.4	$U = \frac{kx^2}{2}$	Yes	Yes	Yes
I.15.3x	$x_1 = \frac{x-ut}{\sqrt{1-u^2/c^2}}$	Yes	No	No
I.15.3t	$t_1 = \frac{t-ux/c^2}{\sqrt{1-u^2/c^2}}$	Yes	No	No

Table 4.2 continued from previous page

Feynman Eq.	Equation	AI - F	Eureqa	Thesis
I.15.10	$p = \frac{m_0 v}{\sqrt{1-v^2/c^2}}$	Yes	No	No
I.16.6	$v_1 = \frac{u+v}{1+uv/c^2}$	Yes	No	No
I.18.4	$r = \frac{m_1 r_1 + m_2 r_2}{m_1 + m_2}$	Yes	Yes	Yes
I.18.12	$\tau = rFsin(\theta)$	Yes	Yes	Yes
I.18.16	$L = mrvsin(\theta)$	Yes	Yes	Yes
I.24.6	$E = \frac{1}{4}m(\omega^2 + \omega_0^2)x^2$	Yes	Yes	Yes
I.25.13	$V_e = \frac{q}{C}$	Yes	Yes	Yes
I.27.6	$f_f = 1/(\frac{1}{d_1} + \frac{n}{d_2})$	Yes	Yes	Yes
I.29.4	$k = \frac{\omega}{c}$	Yes	Yes	Yes
I.29.16	$x = \sqrt{x_1^2 + x_2^2 - 2x_1 x_2 cos(\theta_1 - \theta_2)}$	Yes	No	No
I.30.3	$I_* = I_{*0}\frac{sin^2(n\theta/2)}{sin^2(\theta/2)}$	Yes	No	No
I.32.5	$P = \frac{q^2 a^3}{6\pi\epsilon c^3}$	Yes	Yes	Yes
I.32.17	$P = (\frac{1}{2}\epsilon c E_f^2)(\frac{8\pi r^2}{3})(\frac{\omega^4}{(\omega^2-\omega_0^2)^2})$	Yes	No	No
I.34.8	$\omega = \frac{qvB}{p}$	Yes	Yes	Yes
I.34.10	$\omega = \frac{\omega_0}{1-v/c}$	Yes	No	Yes
I.34.14	$\omega = \frac{1+v/c}{\sqrt{1-v^2/c^2}}\omega_0$	Yes	No	No
I.34.27	$E = \hbar\omega$	Yes	Yes	Yes
I.37.4	$I_* = I_1 + I_2 + 2\sqrt{I_1 I_2}cos(\delta)$	Yes	Yes	Yes
I.38.12	$r = \frac{4\pi\epsilon\hbar^2}{mq^2}$	Yes	Yes	Yes
I.39.10	$E = \frac{3}{2}p_F V$	Yes	Yes	Yes
I.39.11	$E = \frac{1}{\gamma-1}p_F V$	Yes	Yes	Yes

Table 4.2 continued from previous page

Feynman Eq.	Equation	AI - F	Eureqa	Thesis
I.39.22	$P_F = \frac{nk_bT}{V}$	Yes	Yes	Yes
I.40.1	$n = n_0 e^{\frac{-mgx}{k_bT}}$	Yes	No	Yes
I.41.16	$L_{rad} = \frac{\hbar\omega^3}{\pi^2c^2(e^{\frac{\hbar\omega}{k_bT}}-1)}$	Yes	No	No
I.43.16	$v = \frac{\mu q V_e}{d}$	Yes	Yes	Yes
I.43.31	$D = \mu_e k_b T$	Yes	Yes	Yes
I.43.43	$\kappa = \frac{1}{\gamma-1}\frac{k_b v}{A}$	Yes	Yes	Yes
I.44.4	$E = nk_bTln(\frac{V_2}{V_1})$	Yes	Yes	No
I.47.23	$c = \sqrt{\frac{\gamma pr}{\rho}}$	Yes	Yes	Yes
I.48.20	$E = \frac{mc^2}{\sqrt{1-v^2/c^2}}$	Yes	No	No
I.50.26	$x = x_1\Big(cos(\omega t) + \alpha cos(\omega t)^2\Big)$	Yes	Yes	Yes
II.2.42	$P = \frac{\kappa(T_2-T_1)A}{d}$	Yes	Yes	Yes
Total:	50	50	32	34

The result from all simulated equations shows that the developed algorithm solves 34 out 50 equations. The commercial GP software Eureqa solves 32 and AI Feynman solves all 50 equations. Of the 100 available equations the first 50 are tested here, excluding any inverse trigonometric functions (two in total) since neither arcsine nor arccosine are implemented in the developed algorithm. Furthermore, the number of data points used for testing the developed program is 100 compared to 300 for Eureqa and between $10, 10^2$ and 10^3 for AI Feynman except equation I.9.18 where 10^6 are used [41].

4.1.2 Comparison of Samplers

The results presented in the right-most column of table 4.2 indicate if one or several runs during testing is successful. Each test consists of 10 independent runs for each of the three sampling techniques roulette, linear rank and Boltzmann. The performance of the different sampling techniques are studied in section 3.2.2. The study suggests that the three different samplers all produce promising result but achieves this using different approaches to exploration and exploitation, and should thus all

be used in order to produce actual results.

For the majority of the equations in table 4.2 all three samplers have the same result; for every *No* none solved the equation; and for most *Yes* they all solved it. There are however 11 cases where the results varies for the three samplers. Cases where only one or two samplers finds the correct equation, or equations where a sampler finds the correct equation but with wrong complexity. These equations are shown in table 4.3.

Table 4.3: Equations which are not solved by all three samplers. Yes indicates that an equation has been found with the correct complexity, while Yes* indicates a solution with RMSE = 0 has been found but with wrong complexity. The number of solved equations for each sampler can be seen on the last row accompanied by the number of solved equations in terms of only RMSE.

Feynman Eq.	**Roulette**	**Linear Rank**	**Boltzmann**
I.6.20a	No	Yes	Yes*
I.12.2	Yes	Yes*	Yes*
I.12.4	Yes*	Yes*	Yes
I.13.4	No	Yes	No
I.13.12	Yes	Yes*	Yes*
I.18.4	No	No	Yes
I.24.6	Yes	No	Yes
I.32.5	Yes	No	Yes*
I.34.27	Yes*	Yes	Yes
I.37.4	No	Yes	No
I.40.1	Yes	No	Yes
Total: 11	5 (7*)	4 (7*)	5 (9*)

Table 4.3 highlights some differences in performances between the three sampling techniques. For each equation at least one sampler has successfully solved the equation but in several cases solutions are found that are more complex than the target (indicated by *). These cases are examples of solved equations which have not been sufficiently simplified.

Looking at all tested equations the number of solved equations sum up to 28 (30*), 27 (30*) and 28 (32*), for roulette, linear rank and Boltzmann selection, respectively. They all solve 27 or 28 equations when including both accuracy and parsimony. However, Boltzmann selection finds the most solutions (32) if the parsimony requirement

is neglected. While the differences in performance between the samplers are small, the results suggest that Boltzmann sampling is better at finding the correct genetic material in order to solve an equation due to it performing best both in terms of parsimony and accuracy. It is also concluded that a combined usage of all three samplers lead to a better overall result compared to only using one.

4.2 Ventilator Data

In order to test the algorithm on a real application, data from a ventilator containing time, pressure, tidal volume and air flow is studied. A ventilator provides respiratory support to a patient by supplying air to the lungs during inhalation. The data is supplied by i3tex AB and originates from a ventilator that uses a fan to force air into the lungs, which i3tex AB has helped develop. A subset of the raw data is presented in figure 4.1. The pressure and flow refers to sensor measurements in close proximity to the fan while the tidal volume (volume related to inhalation and exhalation) is calculated by some internal algorithm. The genetic algorithm is tasked to find an expression describing the relationship between flow and different combinations of time, pressure and tidal volume.

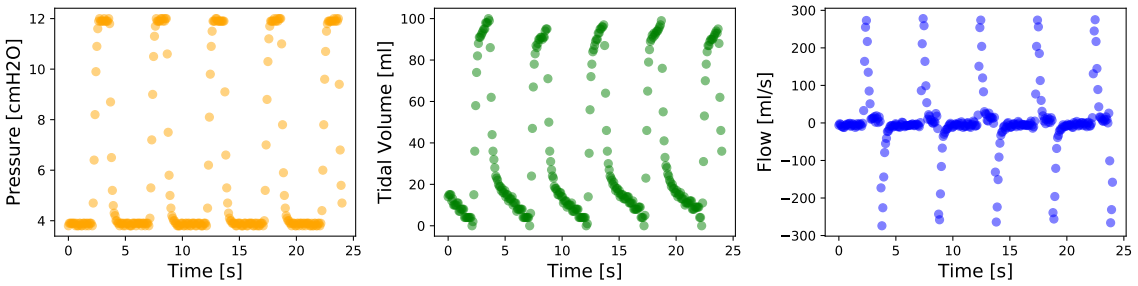


Figure 4.1: A subset of the raw data from ventilator logs. From left to right the graphs shows how pressure, tidal volume and flow changes over time.

The result consists of four simulations with different input data using the same settings as for the previous result, presented in table 4.1. The only difference is that 300 uniformly distributed data points are used instead of 100, in order to ensure sufficient resolution (in accordance with the sampling theorem). The result from each simulation is presented in figures 4.2-4.5. Figure 4.2 shows the resulting error progression and Pareto front from simulations using input data including time, pressure and tidal volume ($Q(t, p, V)$); figure 4.3 shows results from input data including pressure and tidal volume $Q(p, V)$; figure 4.4 shows results from input data including pressure $Q(p)$; and figure 4.5 shows results from input data including time and pressure $Q(t, p)$. For each simulation the TOPSIS method is applied in order to generate a suggestive solution. These solutions are presented in equations (4.1)-(4.4) with their corresponding RMSE, and are plotted in figure 4.6.

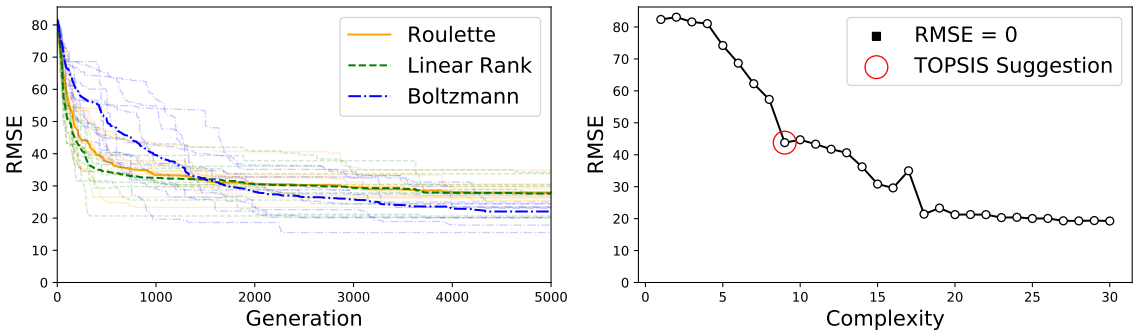


Figure 4.2: Error progression and Pareto front from simulations using 300 data points of time, pressure and tidal volume data, with flow as target. The graphs show the accumulative results from 30 simulations, 10 for each sampling method, using population size of 2000 running for 5000 generations, settings are presented in table 4.1.

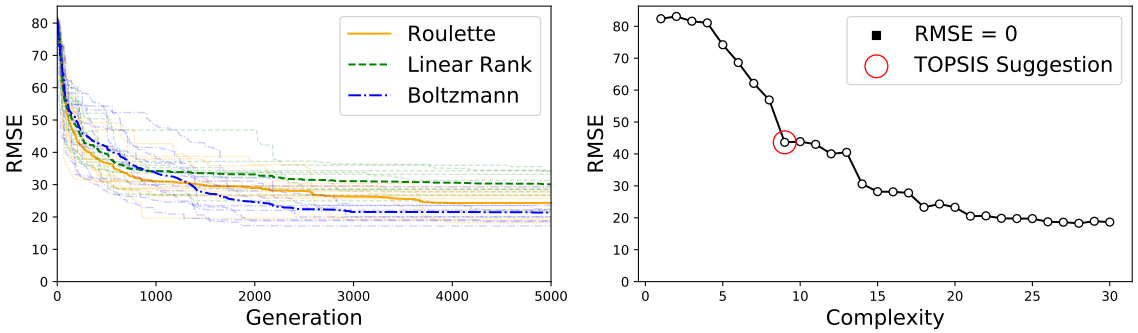


Figure 4.3: Error progression and Pareto front from simulations using 300 data points of pressure and tidal volume data, with flow as target. The graphs show the accumulative results from 30 simulations, 10 for each sampling method, using population size of 2000 running for 5000 generations, settings are presented in table 4.1.

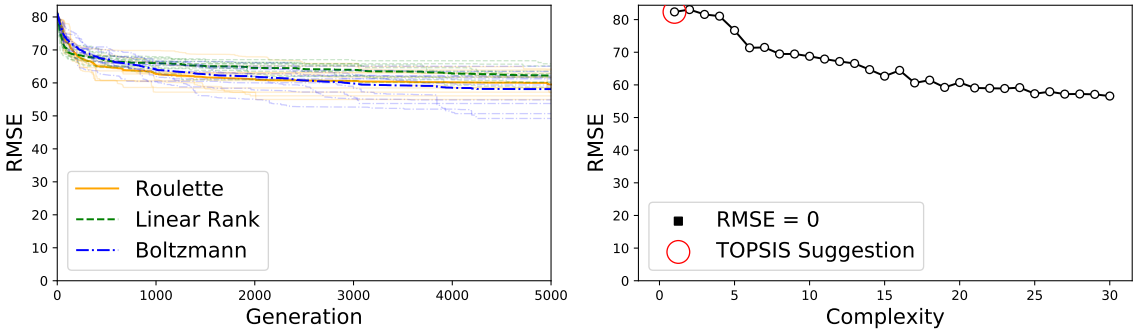


Figure 4.4: Error progression and Pareto front from simulations using 300 data points of pressure data, with flow as target. The graphs show the accumulative results from 30 simulations, 10 for each sampling method, using population size of 2000 running for 5000 generations, settings are presented in table 4.1.

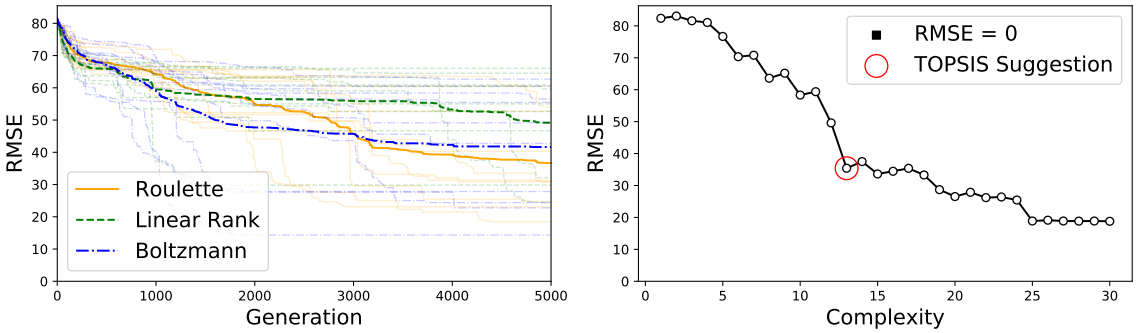


Figure 4.5: Error progression and Pareto front from simulations using 300 data points of time and pressure data, with flow as target. The graphs show the accumulative results from 30 simulations, 10 for each sampling method, using population size of 2000 running for 5000 generations, settings are presented in table 4.1.

The resulting equations generated (suggested by TOPSIS) from the first, $Q(t, p, V)$, and second, $Q(p, V)$, simulations are identical apart from minor variation in numerical values. Their corresponding error progression and Pareto fronts are also similar, indicating that the flow can be described without time as an explicit variable given the pressure and tidal volume. The third, $Q(p)$, simulation provides no real suggestion for a relationship linking flow directly to pressure. The best suggested solution (of the ones suggested by TOPSIS) in terms of RMSE is generated in the fourth, $Q(t, p)$, simulation, where the flow is related to pressure and time. In every simulation apart from the fourth simulation Boltzmann selection provides the best average error progression. This is however not the most important feature since the suggested solution is acquired through TOPSIS, often implying that the best solution in terms of RMSE will not be selected.

$$Q(t, p, V) = p(10.83p - V - 33.67), \quad \text{RMSE} = 43.8 \tag{4.1}$$

$$Q(p, V) = p(10.69p - V - 32.46), \quad \text{RMSE} = 43.6 \tag{4.2}$$

$$Q(p) = p, \quad \text{RMSE} = 82.4 \tag{4.3}$$

$$Q(t, p) = -343.1 \cos(2.54t) \cos(0.40p), \quad \text{RMSE} = 35.4 \tag{4.4}$$

In figure 4.6 all suggested solutions are shown with the target values for flow. The resulting equations from the first and second simulations provides accurate predictions regarding position of inhales and exhales but provides poor estimates for maximum and minimum flow rates. The equation from the fourth simulation provides accurate predictions regarding both position of inhales and exhales as well as peak values for the flow rates.

Out of all simulations the solution with the lowest RMSE is generated in the second simulation according to equation (4.5), which has complexity 28. It is shown in figure 4.7. The solution has similarities to the equation suggested by TOPSIS for the

first and second simulation but has higher accuracy with a RMSE of 18.3 compared to 43.6 and 43.8.

$$Q(p, V) = 1.4p\sin(0.03V)(p(p - \cos(0.46p)) - 1.38(V + \cos(0.14V)) + 1.94) \quad (4.5)$$

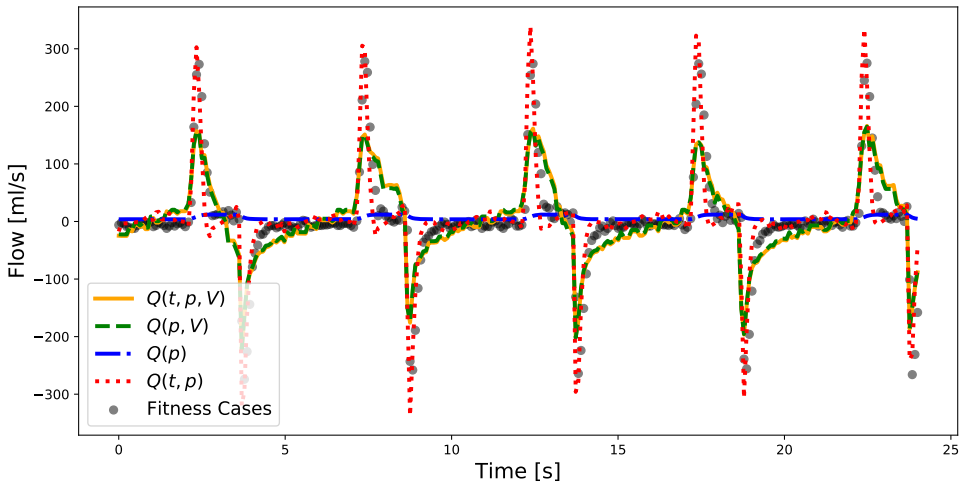


Figure 4.6: Graph showing all four suggested equations from the simulations performed on the ventilator data, linking flow to time, pressure and tidal volume.

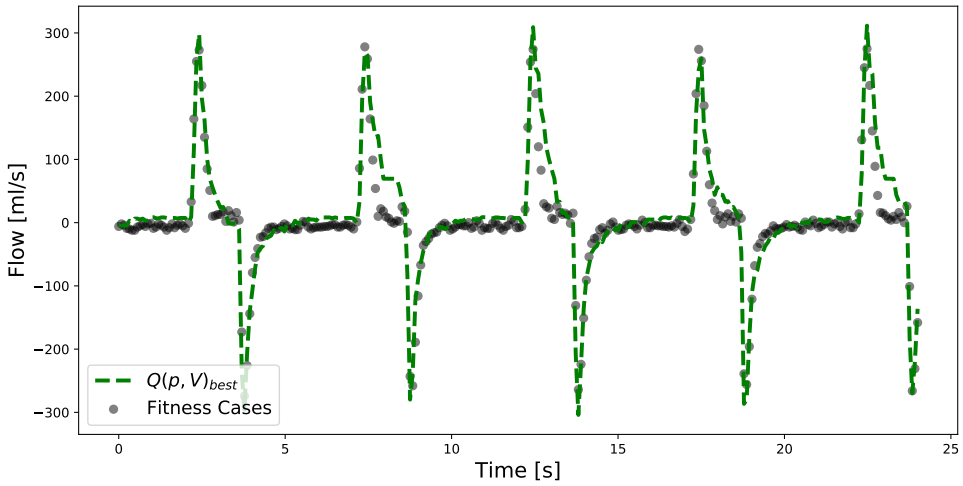


Figure 4.7: Graph showing the best solution out the four performed simulations, linking flow to pressure and tidal volume.

5

Conclusion and Discussion

This chapter is intended to concertise the main findings of the thesis and to discuss the result as well as other concepts utilised and mentioned throughout the report. The main topics of discussion are validity of the input data, algorithm performance and improvements, specifically regarding bloat control and sampling techniques. In the closing section the authors presents their ideas regarding future work and interesting possibilities for genetic programming going forward.

5.1 Key Findings

The objective of this thesis was to create a model performing symbolic regression using genetic programming. A model capable of generating interpretable mathematical expressions describing relationships within data with high accuracy. Without any detailed analysis of the result it is safe to conclude that this objective has been met.

The comparison of the developed algorithm with AI Feynman and Eureqa, demonstrated the model's ability to discover a variety of simple and complex mathematical expressions from data. The result is comparable, arguably better than, Eureqa (there are some uncertainties regarding program parameters used when producing the result for Eureqa). The created model finds 34 out of 50 equations, compared to Eureqa's 32. Also, of the equations that was not found the majority of them was not found by neither the thesis model nor Eureqa – highlighting expression structures which are difficult for genetic algorithms to distinguish. AI Feynman is however outperforming both of the other methods solving all 50 equations, which at least is an indication of their algorithm doing a better job in finding typical relations in physics.

Several bloat controlling techniques have been studied, such as limited crossovers, parsimony pressure and the enhanced version of SAT. The use of only limited crossovers as a bloat control technique was not successful. Neither parsimony pressure nor the enhanced version of SAT were successfully controlling bloat with constant values for α (the penalty constant in parsimony pressure) and k (the proportion of individuals SAT is applied to), respectively. The introduction of adaptive-k proved to be the best technique both in terms of RMSE and in terms of controlling bloat – no matter the input data. It also allows for the most non-disruptive bloat control out of the studied techniques since it does not affect choice of parents or subtrees in crossovers. Additionally it tries to replace the bloat-causing expression

trees with the best possible approximation while still reducing the tree size.

The study of samplers showed how different sampling techniques impacts the exploration and exploitation phases of the genetic search. The size distributions in section 3.2.2 highlighted how selection pressure changes the sizes of the individuals in the population. Boltzmann sampling illustrated this in the perhaps most clear way as the selection pressure changes over the course of the entire simulation. Boltzmann selection also showed the most promising results in both tests (Feynman equations and ventilator data), although the difference was only slight. The selection of hyperparameters for Boltzmann selection also highlighted the complication of introducing additional parameters to an algorithm already including several performance-sensitive parameters.

5.2 Discussion

This thesis covers a substantial amount of concepts and ideas regarding stochastic optimisation, symbolic regression, genetic programming as well as development, tuning and performance of a genetic program. There are thereof a corresponding amount of topics for discussion, in the following section the authors attempts to distil the main areas of discussion and improvement of the entire thesis.

5.2.1 Feynman Equations

Since the Feynman lectures covers varying topics of physics such as mechanics, radiation, heat, electromagnetism and quantum mechanics they contain equations of varying complexities and structures. This also means that it would be possible to choose 50 equations of different nature. A closer look at the equations in table 4.2 reveals that some of the equations have similar structures. For example I.6.20a, I,6.20 and I.6.20b are essentially variants of the same equation. Equations I.10.7, I.15.3x, I.15.3t, I.15.10, I.34.14 and I.48.20 all contain the same denominator $\sqrt{1 - v^2/c^2}$, and neither the developed program nor Eureqa successfully solves any of them. This indicates that both genetic programs struggle with this kind of structure, however it is perhaps not necessary to have six different equations to conclude that. The first 50 of 100 available equations, which AI Feynman and Eureqa have been tested on, were chosen in this thesis. In hindsight a better option would for instance have been to test every second equation from the 100, resulting in a more diversified equation set. This would help highlighting further strengths and weakness of the developed program and genetic programming in general.

To gain more insight on the actual differences in performance between the developed program and the other two methods it would also be informative to study the noise tolerance on all tested equations. This has been done for AI Feynman and would further highlight how well the developed program actually performs – which has great relevance for application on real times series data.

5.2.2 Ventilator Data

The ventilator data provided an interesting challenge for the algorithm, testing its ability to derive expressions from data with noise. The challenge was also unique in the sense that there is no real answer as to what equation is correct, and that there is no certain physical connection between the variables and the target. The results from the simulations highlighted a problem regarding TOPSIS and the method's interaction with the Pareto front. The TOPSIS method selects a single equation from the Pareto front which in turn only provides a single solution for each complexity. Coupling this with the algorithm's inability to ensure that every equation on the Pareto front is in its least complex form (most parsimonious form) means that there might exist other solutions than the one suggested by TOPSIS with better accuracy and similar complexity. It would therefore enhance the value of TOPSIS if the algorithm could simplify expressions to a greater extent. Also, selecting a single solution from 30 simulations might not paint a complete picture of the result – it would be interesting to see to what extent the different input variables were used in different solutions, especially in simulations when the resulting suggested solution did not include all input variables (such as for simulation one, $Q(t, p, V)$, where time was not included in the suggested equation, see equation (4.1)).

The algorithm was tested using four different subsets of the input data ($Q(t, p, V)$, $Q(p, V)$, $Q(p)$ and $Q(t, p)$). The combinations were selected heuristically and was limited due to time constraints. Although all simulations generated expressions describing the flow, the perhaps most interesting result came from the first simulation when all input variables were included. The resulting expression suggested by TOPSIS is presented in equation (4.1) and includes three terms: one positive and two negative. Comparing this equation with a general equation for flow in a ventilator according to,

$$Q = \frac{P_{\text{rs}}}{R} - \frac{V}{CR} - \frac{P_{\text{alv}}}{R}, \tag{5.1}$$

reveals some similarities. In equation (5.1) the flow, Q, is described as a function depending on the pressure applied to the respiratory system, P_{rs}, the resistance of the system, R, the respiratory system's compliance, C, and a constant pressure corresponding to the alveolar and expiratory pressure, P_{alv} [42]. The pressure P_{rs} is the sum of the pressure from the lungs and the ventilator, whereas in the ventilator data the pressure is only the pressure from the ventilator, making the comparison between the two equations somewhat difficult. However, the two equations have similar structures, indicating that the suggested solution has some conformity with an actual expression used in ventilators – highlighting the potential of the algorithm.

The overall best equation in terms of RMSE (shown in figure 4.7) predicts the air flow accurately within the input domain. The logical next step is to test this relationship in other regions of the data, with different respiratory conditions. This was however never done due to time constraints. Although the result shows a practical example on how the algorithm can be used in order to derive unknown relationships in data.

5.2.3 Algorithm

The developed algorithm was an overall success. It enabled simulations to be performed on a reasonable time scale, and allowed easy access for tuning parameters and implementation of additional modules, such as SAT. However, there are still things left to desire. As with all algorithms speed of calculation is an important factor, and in genetic algorithms it is the fitness evaluation that is the most computationally heavy operation. The operation's speed is in turn dependent on the data structure which is used to represent the individuals' expression trees. There are several published ideas on how to create better data structures for individual's expressions. One alternative is to use directed acyclic graphs (DAGs), which avoids the need for recursive evaluation of the expression trees resulting in faster interpretation of individuals [22, 25]. Also, because of the stochastic nature of evolutionary processes genetic algorithms requires a large number of iterations in order to converge. This creates two possibilities for improvement, either, as previously mentioned, reduce the calculation time for evaluating fitness or alter the course of adaptation in order for the algorithm to converge faster. This last approach leaves room for bias and could e.g. imply changing the size of the terminal and function sets.

An advantage of genetic algorithms compared to neural networks is that they require much less data points (fitness cases). In genetic algorithms adding additional data points does not make significant improvements to performance, given sufficient sampling [41]. Hence, using only 100 (or 300 by Eureqa) data points for the Feynman equations is enough, which in specific cases could be an upside of genetic algorithms. For some equations (I.6.20b, I.9.18, I.29.16) AI Feynman requires usage of a convolutional neural network to identify symmetries, separability and perform variable reduction, resulting in the need of at least 1000 data points, which could possibly change for real applications when there is substantially more noise, creating a need for even more data.

An addition to the general algorithm is to introduce a symbolic simplification step. It could either be used to simplify every expression tree every generation, or only used on all solutions in the Pareto frontier after the last generation. This would force every solution to take its most parsimonious form, which in turn would mean that in most cases of solutions with RMSE = 0 the solution would also be optimal in terms of parsimony. It would most likely lead to an improvement for each individual sampler (most *Yes** in table 4.3 would have been solved optimally), and for the program in general. Also, as discussed regarding the ventilator data and the TOPSIS method, by simplifying all expressions it is easier to select the optimal solution when there is no solution with RMSE = 0.

5.2.4 Bloat Control

Three different approaches to controlling bloat was studied in section 3.2.1, targeting bloat in three different ways. It was concluded that using only size-fair/limited crossovers does slow down bloat, but it also hinders the natural evolution within the population greatly. It did however show descent results for best solution over time,

but it did lack one important feature: adaptability (ability to adapt the magnitude of the bloat control depending on changes in the population).

Parsimony pressure also has its drawbacks, one of which is that it alters the choice of parents for breeding. While it does a great job of promoting smaller individuals, it can also lead to loss of well fitted larger individuals, containing key genetic material. However, the main problem preventing this method from being implemented effectively is the same as for limited crossovers: the lack of adaptability, guaranteeing it controls bloat sufficiently independent on population characteristics. The same goes for the standard and enhanced implementation of SAT; choosing a constant k does not provide sufficient bloat control.

Based on the conclusion that none of the three methods could guarantee sufficient bloat control at all instances of simulations, SAT was implemented using an adaptive k-value. This could have been done using parsimony pressure as well, adaptively changing α, which has been done in other articles [32]. However, due to the fact that there is no instance, to the knowledge of the authors, where SAT has been implemented with varying values of k and that it has some superior features compared to parsimony pressure, it was chosen as bloat controlling technique.

Two further topics for discussion regarding SAT is the cap-value of adaptive-k and the implication of the enhancement of the original SAT method. The cap, which is introduced with adaptive-k, regulates the aggressiveness of the bloat control. Whenever the average tree size grows bigger than the cap, k increases. The cap is necessary for the bloat control to work as desired and allows for equal sizes between different runs and input functions. However, it does not remove all trees bigger than the cap, like the original hard-cap bloat control proposed by Koza [4], thus allowing for some exploration of the function space above the cap. The choice of cap is however very important. If it is too small, the program will be less likely to find correct solutions. If if too big, the program will be slower and may struggle to find solutions with optimal parsimony. The cap of 20 was chosen having analysed the complexities of the 50 chosen equations. A larger cap might have led to better/worse results. It should also be mentioned that SAT will sometimes harm big trees by over-reducing them – it always reduces sizes but will sometimes remove good genetic material as well.

The implementation of SAT was done with a change compared to its original form. In the original version of SAT a random terminal is chosen and the best accompanying parameter to multiply with is computed. The new subtree always has three nodes. It was concluded that this original version would be more likely to substitute in worse-fitting subtrees, than if one were to do the same calculations for all available terminals. It was also concluded that there exists cases when a constant (the mean) would be a better fit than a constant multiplied by a terminal. Therefore the enhanced version was developed. It is guaranteed to be at least as good as the old version since all terminals are studied. It will add to the computational cost of the program, but the magnitude of the addition is negligible in comparison to the rest

of the program.

5.2.5 Sampling Techniques

The thesis investigates the performance of roulette, linear rank and Boltzmann selection, and although their impact on the population is different they all perform similarly well. An interesting discovery was done when comparing the resulting size distributions of roulette and linear rank sampling. Because roulette sampling is described as the selection method with higher selective pressure compared to linear rank it would be reasonable to assume that it would favour larger individuals to a greater extent than linear rank, assuming that larger individuals exhibit better fitness to a larger extent than smaller individuals. Although this is not the case, which can be seen when observing figures 3.7 and 3.8 in section 3.2.2. A possible explanation to this is that linear rank's size distribution is a closer depiction of the actual resulting size distribution from the genetic operators. There is always a probability for a genetic operator to create an individual with invalid mathematical expression, such as division with 0. In roulette sampling these individuals will never be selected, but in linear rank there is an small associate probability of being selected, and the overall probability of selecting a defected individual is higher when there are more of them. In other words, because there is a certain amount of large individuals created every iteration with invalid mathematical syntax, linear rank selection will lead to a more focused size distribution with larger individuals. At first glance it might seem to be a negative process, however this enables expressions trees a *second chance* to become well-adapted individuals, and judging by the results of linear rank, it seems to provide sufficient sampling.

Boltzmann selection is the most intricate of the three selection methods. In the process of selecting which parameters to test it became apparent that it is a difficult process. In the end three initial temperatures were selected as candidates, all using the same other parameters. From the result it seems like Boltzmann selection has a slightly better performance compared to roulette and linear rank selection. Further, from the error progression and size distribution for Boltzmann selection, it seems like the evolutionary process is not really completed (there is still a non-zero gradient on the error progression) implying that the simulation would benefit from staying at a high selection pressure for more iterations. This was tested to some extent but resulted in problems with bloat when the selection pressure was held high for a greater number of iterations. A possible fix to this issue would be to discontinue the change of Boltzmann temperature at a late point in the simulation, effectively freezing the selection profile – enabling exploitation while possibly avoiding too much bloat.

There is a plethora of different sampling methods which can be used for breeding, as is the case for a lot of the components in a genetic algorithm. There is a large amount of published material regarding different sampling techniques. For instance there are several interesting approaches to tournament selection that provides both genetic diversity and bloat control [43].

5.3 Opportunities for Further Work

In its entirety this project has highlighted many areas of genetic programming and symbolic regression which are subject for future studies. Some minor, such as tuning of probabilities and ratios of the genetic operators, while others major – components of the thesis which in them self have and will support material for many articles to come. A major component of a genetic algorithm is sampling of individuals. While the Boltzmann sampling used in this report seem to provide the best result the selection of parameters was done using a rather arbitrary framework. Thus, it would be interesting to study the limitations and possibilities of Boltzmann sampling in detail. A possibility could be to incorporate features of linear rank selection into Boltzmann, in order to have better control over the selection pressure.

SAT has been studied and discussed a lot in this thesis. The enhanced version of it is at least as good as the standard version. However, it can be improved even further. As of now it is searching for either a constant or a constant times a terminal. This could be extended to search for more complicated functions such as polynomials of varying degrees, performing linear regression on a greater variety of subtrees, or other mathematical functions. This would require the k-value to be bigger but would lead to better approximations. One could also include some other functions than multiplication in the search, which the authors of the original SAT have begun studying [44].

In section 2.6 the island model was introduced, a method used to avoid premature convergence within stochastic optimisation. The special case of non-interacting islands, equal to running independent runs, has been used in the developed program. An implementation of interaction between different populations would be an interesting phenomena to study further. Allowing different courses of evolutionary progress to interact and mix.

Arguably, the most interesting topic for future work would be to study how to implement modules and features from AI Feynman in order to improve the algorithm. AI Feynman is based on methods from physics and maths used in order to derive and prove relationships, methods which would undoubtedly improve the performance of any algorithm tasked to find relationship in data. Additionally, the authors of AI Feynman expresses some concerns regarding their algorithm when applied on difficult equations, and mention that it could be improved using genetic features [41]. Perhaps the future of symbolic regression is a combination of the two methods.

Bibliography

[1] Isaac Newton. *Philosophiae naturalis principia mathematica*. William Dawson & Sons Ltd., London, 1687.

[2] James Clerk Maxwell. Viii. a dynamical theory of the electromagnetic field. *Philosophical Transactions of the Royal Society of London*, 155:459–512, 1865.

[3] Cynthia Rudin. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature Machine Intelligence*, 1:206–215, May 2019.

[4] John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA, 1992.

[5] Michael Schmidt and Hod Lipson. Distilling free-form natural laws from experimental data. *Science*, 324(5923):81–85, 2009.

[6] I. Icke and J. C. Bongard. Improving genetic programming based symbolic regression using deterministic machine learning. In *2013 IEEE Congress on Evolutionary Computation*, pages 1763–1770, June 2013.

[7] A. M. TURING. I.—COMPUTING MACHINERY AND INTELLIGENCE. *Mind*, LIX(236):433–460, 10 1950.

[8] John H Holland. Outline for a logical theory of adaptive systems. *Journal of the ACM (JACM)*, 9(3):297–314, 1962.

[9] John H. Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. MIT Press, Cambridge, MA, USA, 1992.

[10] A. Garg and K. Tai. Review of genetic programming in modeling of machining processes. In *2012 Proceedings of International Conference on Modelling, Identification and Control*, pages 653–658, 2012.

[11] M. S. Nobile, D. Besozzi, P. Cazzaniga, D. Pescini, and G. Mauri. Reverse engineering of kinetic reaction networks by means of cartesian genetic programming and particle swarm optimization. In *2013 IEEE Congress on Evolutionary Computation*, pages 1594–1601, 2013.

[12] Mark A Kaboudan. Genetic programming prediction of stock prices. *Computational Economics*, 16(3):207–236, 2000.

[13] Matt Gormley. Machine learning model and feature selection, Spring 2020.

[14] Lauren Hannah. Stochastic optimization. *International Encyclopedia of the Social Behavioral Sciences*, 2, 12 2015.

[15] M Wahde. *Biologically Inspired Optimization Methods*. WIT Press, Ashurst Lodge, Ashurst, Southampton, SO40 7AA, UK, 2008.

[16] Mark Girilami Simon Rogers. *A First Course In Machine Learning, Second Edition*. Chapman and Hall, 2017.

[17] P. Barmpalexis, K. Kachrimanis, A. Tsakonas, and E. Georgarakis. Symbolic regression via genetic programming in the optimization of a controlled release pharmaceutical formulation. *Chemometrics and Intelligent Laboratory Systems*, 107(1):75 – 82, 2011.

[18] ME Shahin, Liu Yun, CMM Chin, Liang Gao, Chin-Tsan Wang, Xiaodong Niu, Ankit Goyal, and Akhil Garg. An application of genetic programming for lithium-ion battery pack enclosure design: Modelling of mass, minimum natural frequency and maximum deformation case. In *IOP Conference Series: Earth and Environmental Science*, volume 268, page 012065. IOP Publishing, 2019.

[19] Ehsan Saljoughi. Application of genetic programming as a powerful tool for modeling of cellulose acetate membrane preparation. *Chem. Ind*, 1:4, 2017.

[20] Guido F. Smits and Mark Kotanchek. *Pareto-Front Exploitation in Symbolic Regression*, pages 283–299. Springer US, Boston, MA, 2005.

[21] S Forrest. Genetic algorithms: principles of natural selection applied to computation. *Science*, 261(5123):872–878, 1993.

[22] S. Handley. On the use of a directed acyclic graph to represent a population of computer programs. In *Proceedings of the First IEEE Conference on Evolutionary Computation. IEEE World Congress on Computational Intelligence*, pages 154–159 vol.1, June 1994.

[23] Ibrahim Kucukkoc, Aslan Deniz Karaoglan, and Ramazan Yaman. Using response surface design to determine the optimal parameters of genetic algorithm and a case study. *International Journal of Production Research*, 51(17):5039–5054, 2013.

[24] Peter John Angeline. Genetic programming and emergent intelligence. *Advances in genetic programming*, 1:75–98, 1994.

[25] Herman Ehrenburg. Improved directed acyclic graph evaluation and the combine operator in genetic programming. In *Proceedings of the 1st Annual Conference on Genetic Programming*, page 285–290, Cambridge, MA, USA, 1996. MIT Press.

[26] Geng Li. *Tuning genetic programming performance via bloating control and a dynamic fitness function approach.* PhD thesis, University of Manchester, 2013.

[27] Nisha Saini. Review of selection methods in genetic algorithms. *International Journal of Engineering and Computer Science*, 6(12):22261–22263, Dec. 2017.

[28] Noraini Mohd Razali, John Geraghty, et al. Genetic algorithm performance with different selection strategies in solving tsp. In *Proceedings of the world congress on engineering*, volume 2, pages 1–6. International Association of Engineers Hong Kong, 2011.

[29] Smit Anand, Nishat Afreen, and Shama Yazdani. A novel and efficient selection method in genetic algorithm. *International Journal of Computer Applications*, 129:7–12, 2015.

[30] Gabriel Kronberger, Stephan Winkler, Michael Affenzeller, Andreas Beham, and Stefan Wagner. On the success rate of crossover operators for genetic programming with offspring selection. In *International Conference on Computer Aided Systems Theory*, pages 793–800. Springer, 2009.

[31] William B Langdon. Size fair and homologous tree genetic programming crossovers. *Genetic programming and evolvable machines*, 1(1/2):95–119, 2000.

[32] Riccardo Poli and Nicholas Mcphee. Parsimony pressure made easy: Solving the problem of bloat in gp. 11 2014.

[33] Thi Chu and Quang Uy Nguyen. Reducing code bloat in genetic programming based on subtree substituting technique. pages 25–30, 11 2017.

[34] Leonardo Trujillo, Luis Muñoz, Edgar Galván-López, and Sara Silva. neat genetic programming: Controlling bloat naturally. *Information Sciences*, 333:21 – 43, 2016.

[35] Kevin L. Edwards Ali Jahan, Marjan Bahraminasab. *Multi-criteria Decision Analysis for Supporting the Selection of Engineering Materials in Product Design*. Butterworth-Heinemann, 2016.

[36] Zhiyuan Wang and Gade Pandu Rangaiah. Application and analysis of methods for selecting an optimal solution from the pareto-optimal front obtained by multiobjective optimization. *Industrial & Engineering Chemistry Research*, 56(2):560–574, 2017.

[37] Jenny E. Jeong and Peng Qiu. Quantifying the relative importance of experimental data points in parameter estimation. *BMC Systems Biology*, 12:e164 – 22, 2018.

[38] Hamed Taherdoost. Sampling methods in research methodology; how to choose a sampling technique for research. *International Journal of Academic Research in Management*, 5:18–27, 01 2016.

[39] L. Rade and B. Westergren. *Mathematics Handbook for Science and Engineering*. Springer Berlin Heidelberg, 2004.

[40] Richard Phillips Feynman, Robert Benjamin Leighton, and Matthew Sands. *The Feynman lectures on physics; New millennium ed.* Basic Books, New York, NY, 2010. Originally published 1963-1965.

[41] Silviu-Marian Udrescu and Max Tegmark. Ai feynman: a physics-inspired method for symbolic regression, 2019.

[42] Umberto Lucangelo, Francesca Bernabè, and Lluís Blanch. Lung mechanics at the bedside: Make it simple. *Current opinion in critical care*, 13:64–72, 03 2007.

[43] Huayang Xie and Mengjie Zhang. Parent selection pressure auto-tuning for tournament selection in genetic programming. *IEEE Transactions on Evolutionary Computation*, 17(1):1–19, 2012.

[44] Thi Chu, Quang Uy Nguyen, and Van Loi Cao. Semantics based substituting technique for reducing code bloat in genetic programming. pages 77–83, 12 2018.

A

Selection of Sampling Scheme

A.1 Selecting Candidate Schemes

Three sampling methods were chosen as candidate samplers to be used for producing results: roulette, linear rank and Boltzmann selection. The choice of sampling schemes where based on the diverse nature of how they perform their sampling. Roulette promotes exploitation, selecting more fit individuals proportional to their normalised fitness. Linear ranking promotes exploration, selecting individuals based on their rank, ignoring the actual distance in fitness between two individuals. Boltzmann attempts to incorporate both exploitation and exploration by changing the nature of the sampling for every generation. Both roulette and linear rank selection does not have any hyper parameters unlike Boltzmann selection.

A.1.1 Selecting Boltzmann Parameters

The temperature of Boltzmann selection is inversely proportional to the selection pressure – a low temperature creates a high selection pressure, and vice versa. For every generation the temperature changes according to,

$$T = T_0(1-\alpha)^k, \quad k = \gamma + \beta\frac{g}{G}, \tag{A.1}$$

as in equation 2.7, in section 2.3.4.6, where T_0, α, β and γ are hyper parameters that can be altered in order to change the temperature profile for a simulation. The important factors to consider for the temperature is the gradient and final temperature (temperature in and close to the last generation). The gradient determines how fast the selection goes from randomly to exponentially dependent on the fitness, and the final temperature governs the selection pressure in the end. If the temperature is not close to zero near the end of the simulation, there is a high probability of selecting individuals with low fitness, which undesirable.

In figure A.1 the temperature profiles for different combinations of parameters are shown. From these temperature profiles three candidate parameter setups were selected for further investigation, these temperature profiles are shown in figure A.2. The candidates were selected due to their difference in selection pressure and represents a transition from high selection pressure ($T_0 = 150/N$) and low selection pressure ($T_0 = 350/N$). The corresponding selection pressures are visualised in figure A.3, where the unnormalised probability of selection depending on the fitness for different stages of the simulation is shown. The gradient determines the selection

pressure. A steep gradient corresponds to high differentiation between individuals with small fitness differences; high selection pressure. The line for the zeroth generation illustrates the initial selection pressure. From low to high temperature the three setups have decreased selection pressure, corresponding to three slightly different approaches to Boltzmann selection used as candidate setups.

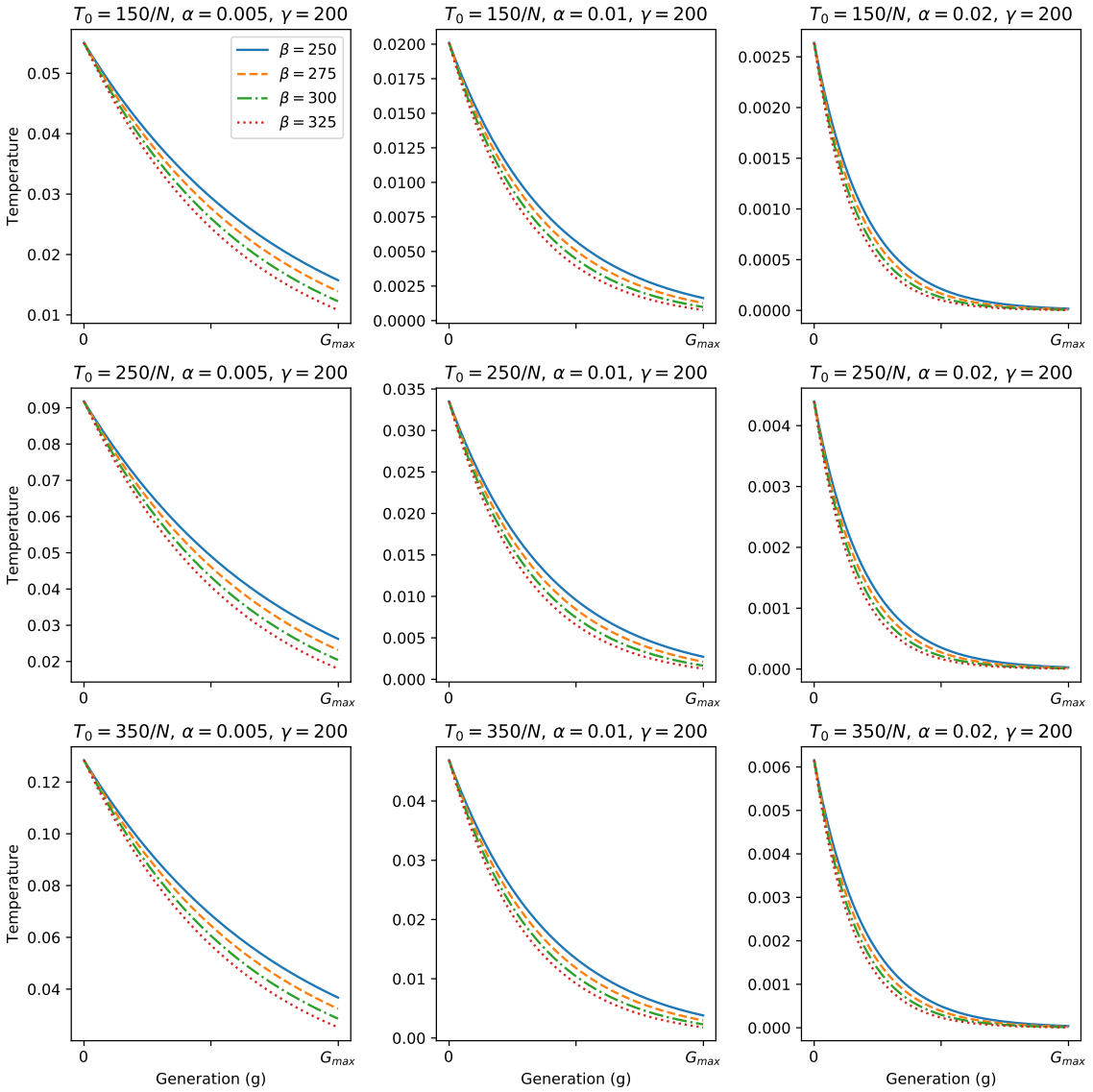


Figure A.1: Temperature profiles for different combinations of parameters for Boltzmann selection. Each graph represents one setup of T_0, α and γ with four lines each using a different β.

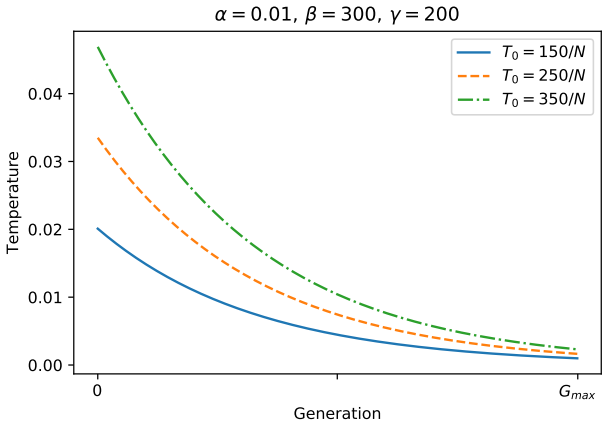


Figure A.2: Temperature profiles corresponding to three setups of Boltzmann selection used for further examination. The three profiles have $\alpha = 0.01$, $\beta = 300$ and $\gamma = 200$ with different initial temperatures, $T_0 = 150/N$, $250/N$, $350/N$.

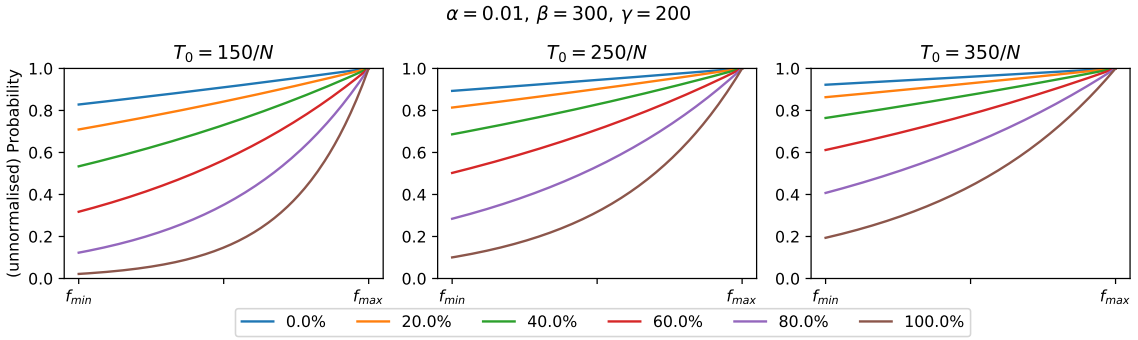


Figure A.3: Selection pressures for three Boltzmann temperature profiles, shown in figure A.2. Each graph presents six stages of the simulation, after 0%, 20%, 40%, 60%, 80% and 100% of generations. The right most graph, corresponds to higher overall selection pressure compared to the middle and left most graph.

B

Time Estimation of Program

B.1 The Dependencies of Execution Time

When looking at algorithms it is very useful to know how the computational time of the algorithm scales with different parameters. Time complexity of an algorithm is often expressed using *the big O notation*. In short, the big O notation describes how the algorithmic time complexity generally scales with respective variable and hence does not provide an exact expression of the execution time. Exact constants, which may also depend on specs of the used computer, are hence neglected. Some examples of this for an input variable n are $O(n)$, $O(log(n))$ and $O(2^n)$ which describe linear time, logarithmic time and exponential time.

The algorithm presented in this thesis contains many different steps and is also stochastic, meaning that when executed there might be different events occurring in different runs. It would still be beneficial to have some knowledge of how the execution time depends on some of the input parameters. The following input parameters were chosen to be studied below: type of sampler, max number of generations, number of individuals in the population, the number of data points in the data set and the number of input variables. For all runs enhanced SAT adaptive was used and capped at 20.

Initially the execution time was studied while varying the number of input variables and the number of data points in the data set. This was done while keeping max number of generations and number of individuals constant at 300. The results of this for three different samplers are presented in figure B.1.

To then study the dependency on the number of individuals and max number of generations, the values for number of data points and number of input variables were kept constant at 100 and 5 respectively. The results for the same three samplers are shown in figure B.2.

B.1.1 Functional Expressions of Execution Time

From the results shown in figure B.1 & B.2 one may analyse the behaviour of the execution time and try to make a rough estimation on how it depends on number of input variables, number of data points, number of generations and number of individuals for each of the samplers. But since this thesis revolves around an algorithm

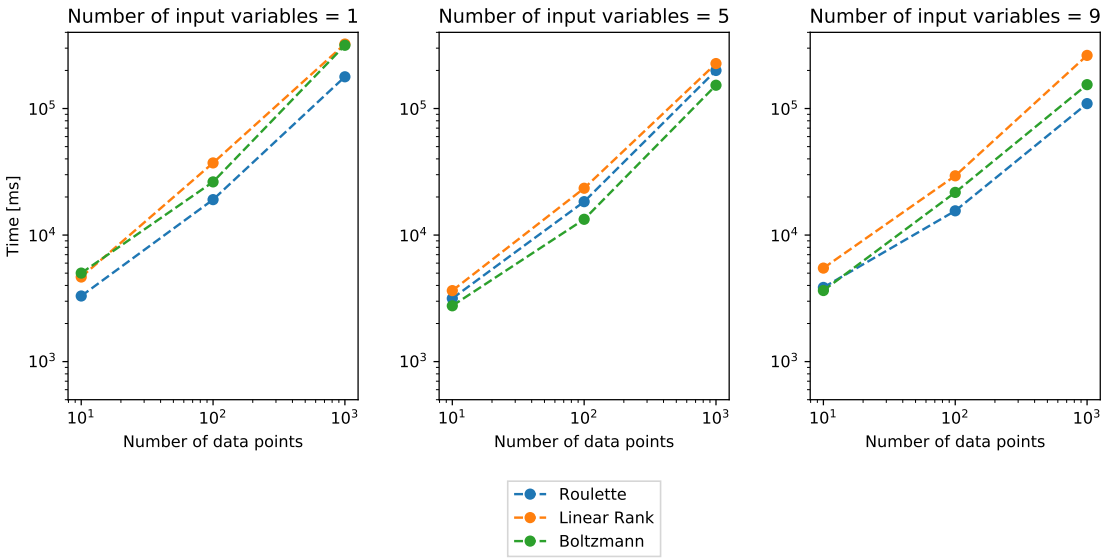


Figure B.1: The executions time of the three different samplers as a function of number of data points for the cases of 1, 5 and 9 input variables.

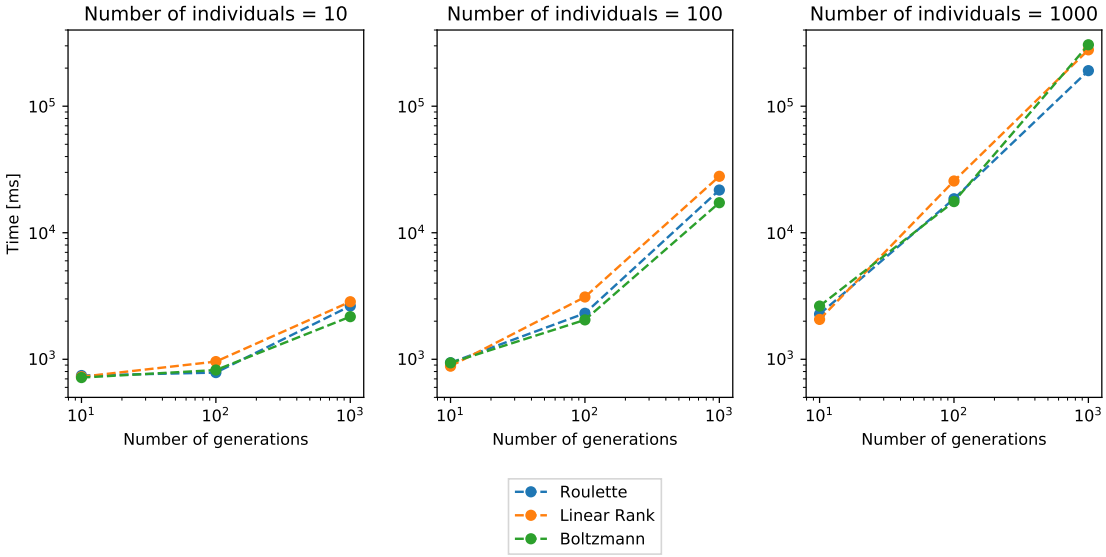


Figure B.2: The executions time of the three different samplers as a function of number of generations for the cases of 10, 100 and 1000 individuals.

for symbolic regression, why not use it to receive symbolic functional expressions for each sampler, and then transform to big O notation? The Pareto frontiers presented in figure B.3 are produced using the algorithm developed in the thesis on the data from the time measurements above.

The Pareto frontiers for each sampler are presented in figure B.3, together with the suggested solution from TOPSIS. Those functions are presented in the following equations, where x_1 is max number of generations, x_2 number of individuals, x_3 number of data points and x_4 number of input variables, according to,

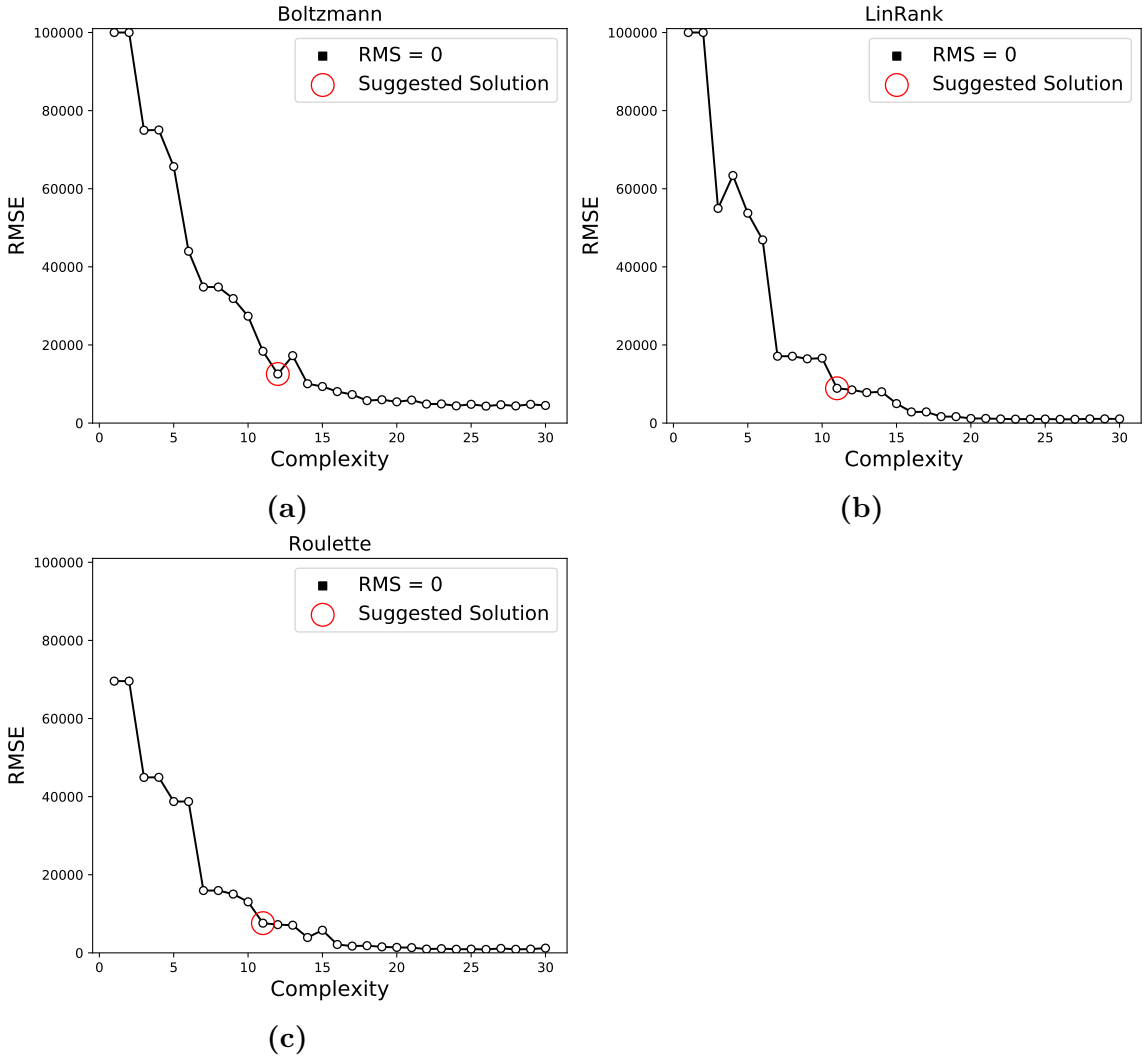


Figure B.3: The Pareto frontiers from simulations run on the measured time data in section B.1, with a suggested solution from the TOPSIS algorithm.

$$t_{Boltzmann} \approx x_1 \left(\frac{x_2}{\pi} + \frac{x_3}{\sqrt{x_4}} - 54.53 \right),$$
$$t_{linrank} \approx 0.026 \cdot x_1 \cdot x_2 \cdot x_3 \cdot e^{-e \cdot x_4},$$
$$t_{roulette} \approx 0.023 \cdot x_1 \cdot x_2 \cdot x_3 \cdot cos(cos(x_4)).$$

These symbolic functions for execution time can the be expressed in the big O notation as,

$$t_{boltzmann} \rightarrow O(x_1) \cdot O(x_2 + \frac{x_3}{x_4}),$$
$$t_{linrank} \rightarrow O(x_1 \cdot x_2 \cdot x_3) \cdot O(exp(-x_4)),$$
$$t_{roulette} \rightarrow O(x_1 \cdot x_2 \cdot x_3).$$

While the three big O expressions differ somewhat from each other, it seems as the execution time scales linearly with the number of generations, number of individuals and number of data points in one way or another for all three samplers. It is interesting that the number of variables seems to be reducing the execution time, which by an analysis does not make sense. It should be noted that while the first three variables take values in the ranges [10,1000] the number of input variables can only be 1, 5 or 9. Hence there might be a big enough variance between execution time of runs to trick the program into thinking that x_4 reduces the overall time.

From the expressions for the three execution times $t_{boltzmann}$, $t_{linrank}$ and $t_{roulette}$ one major conclusion can be drawn: these are not the exact symbolic functions that describe execution time. They do however give the kind of indication which was sought after: execution time seems to be scaling linearly with x_1, x_2 and x_3.

C

Figures from Results

In this chapter the error progression and Pareto frontier for all simulations on the Feynman equations is presented in order of table 4.2.

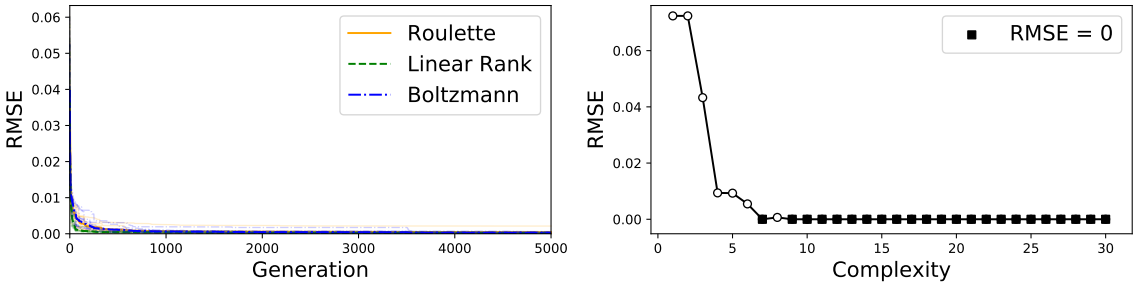


Figure: Equation I.6.20a.

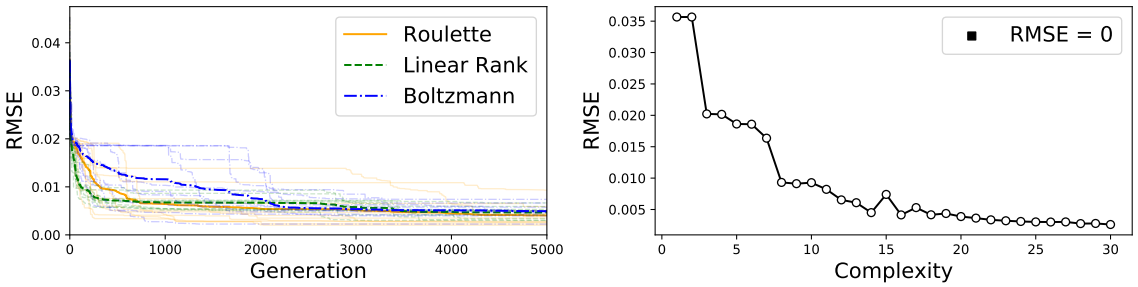


Figure: Equation I.6.20.

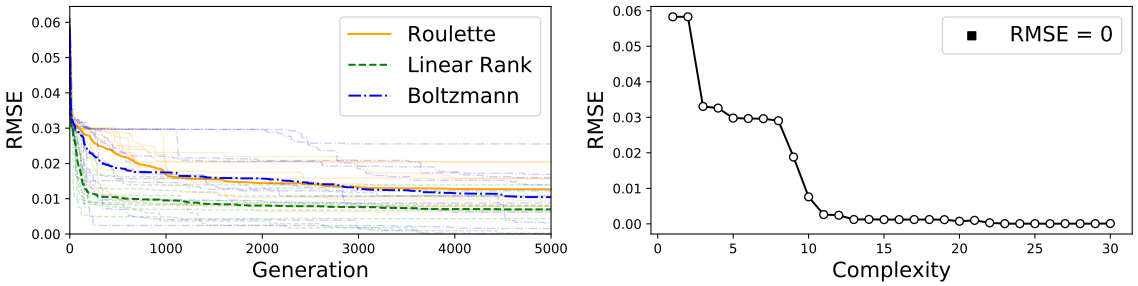


Figure: Equation I.6.20b.

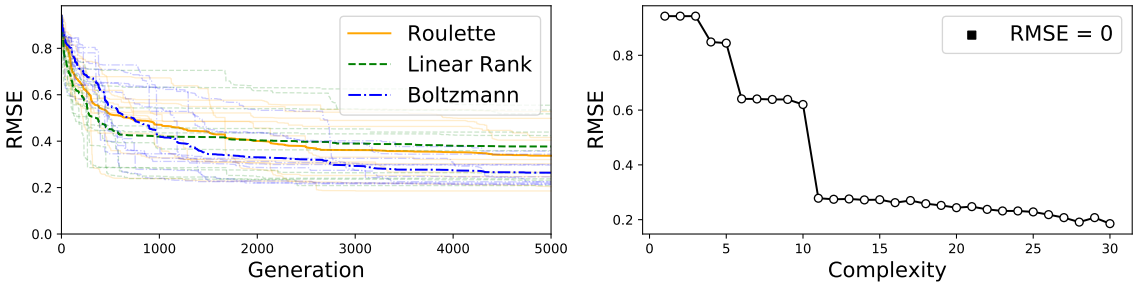


Figure: Equation I.8.14.

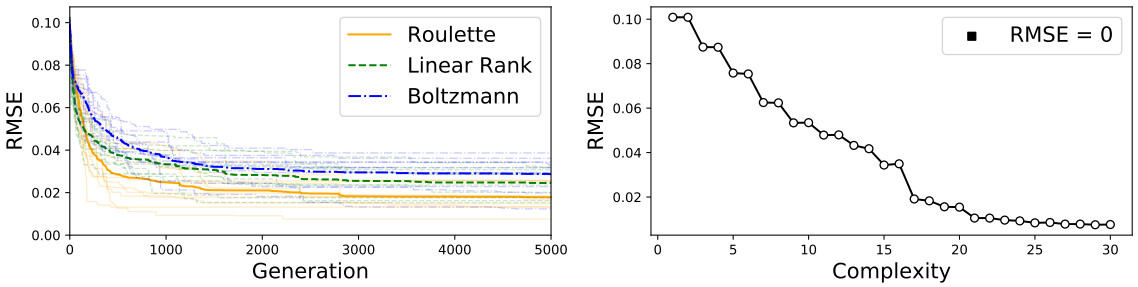


Figure: Equation I.9.18.

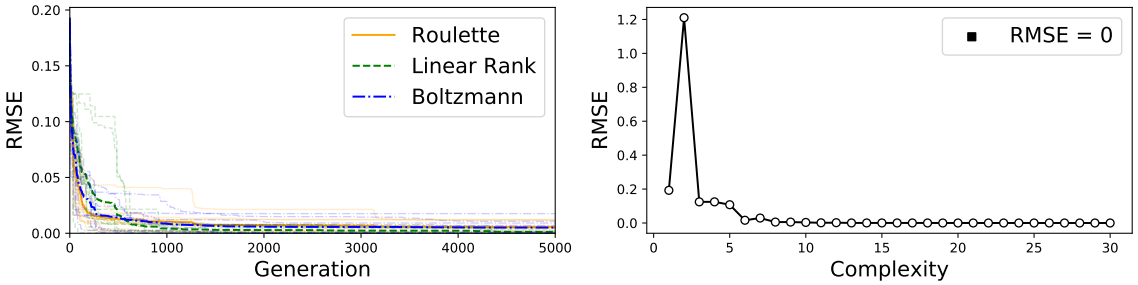


Figure: Equation I.10.7.

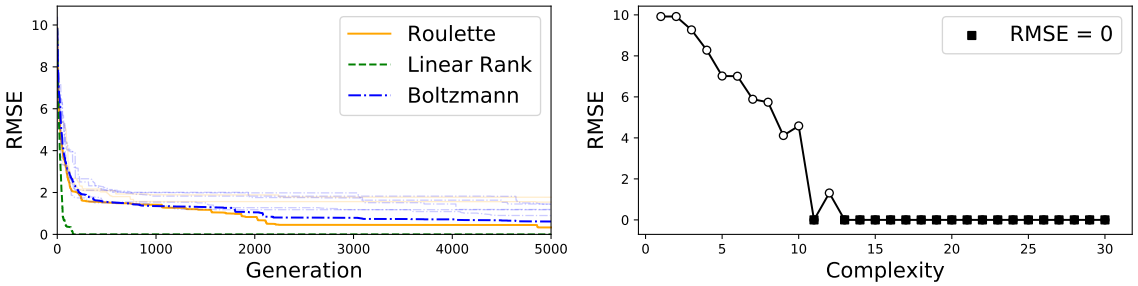


Figure: Equation I.11.19.

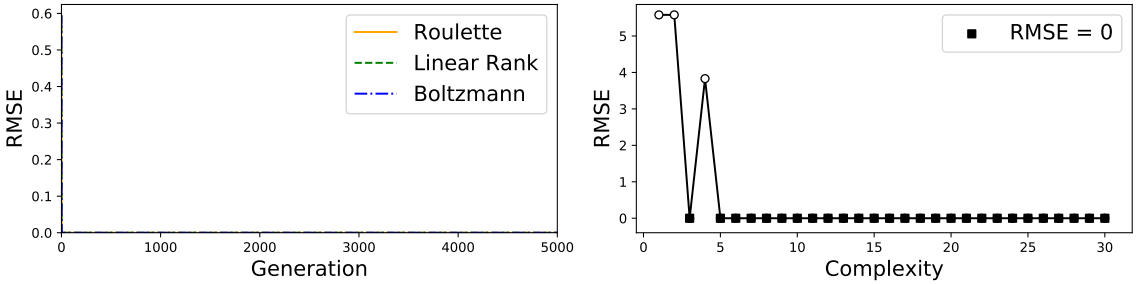


Figure: Equation I.12.1.

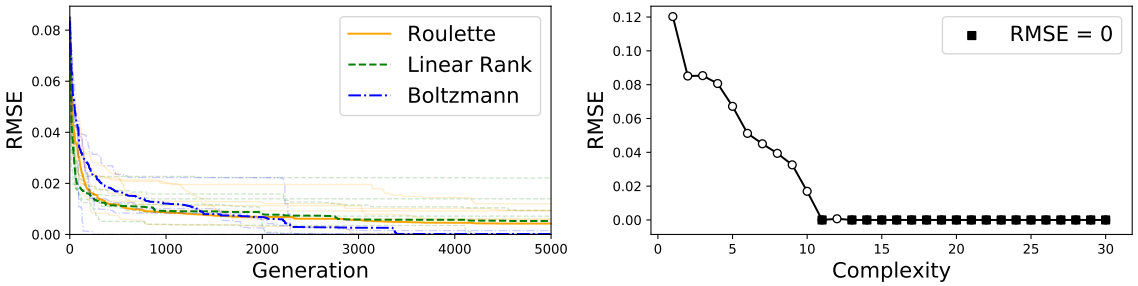


Figure: Equation I.12.2.

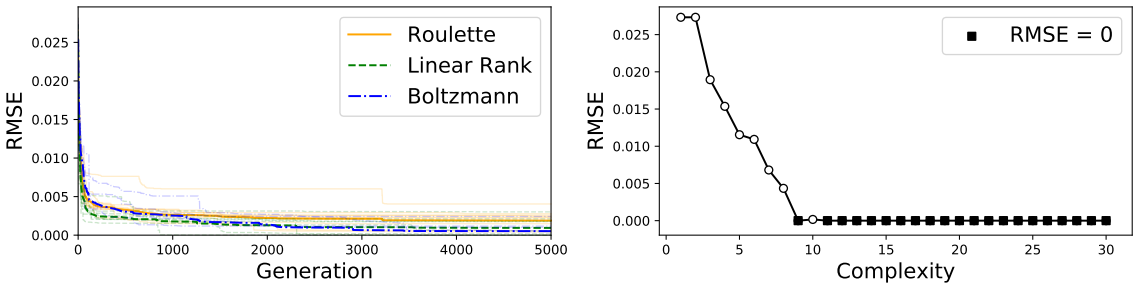


Figure: Equation I.12.4.

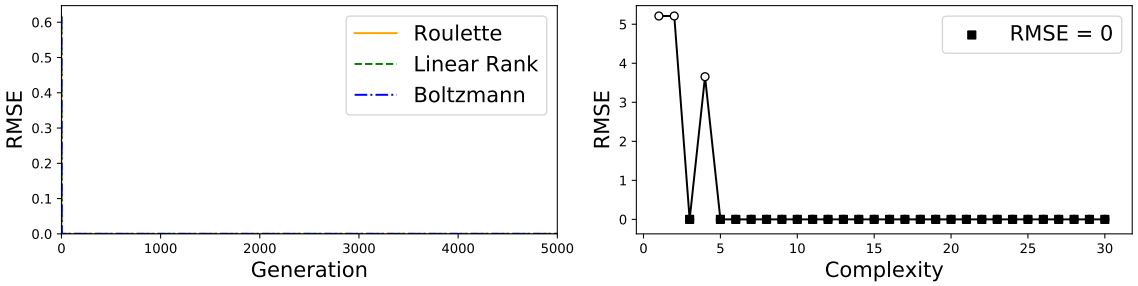


Figure: Equation I.12.5.

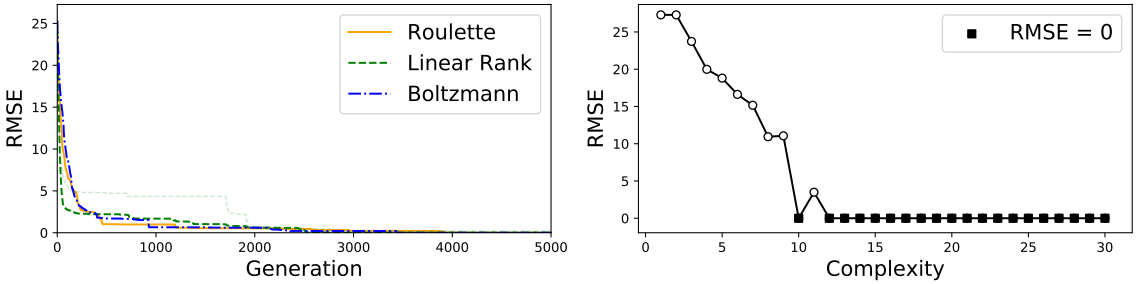


Figure: Equation I.12.11.

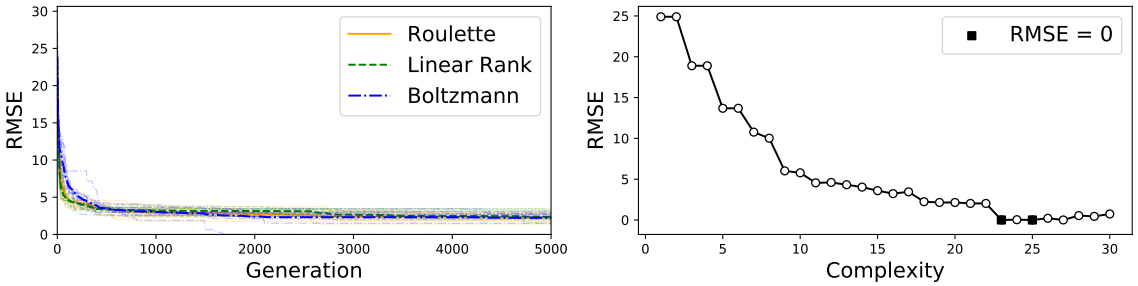


Figure: Equation I.13.4.

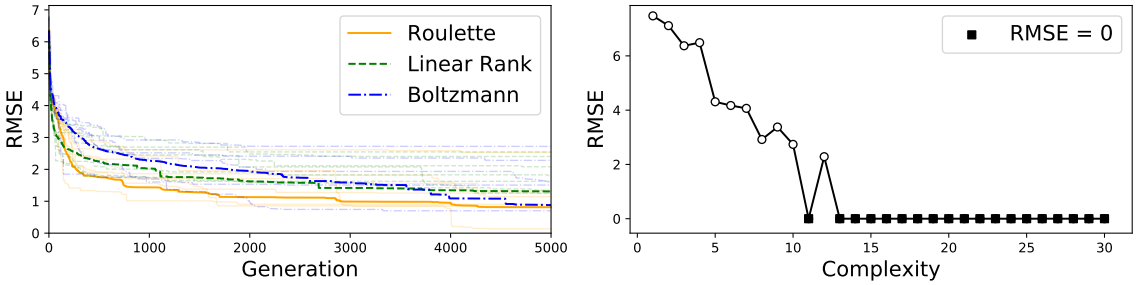


Figure: Equation I.13.12.

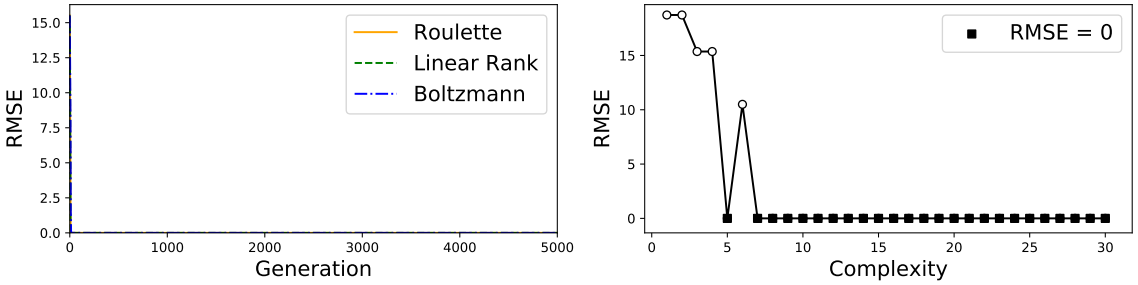


Figure: Equation I.14.3.

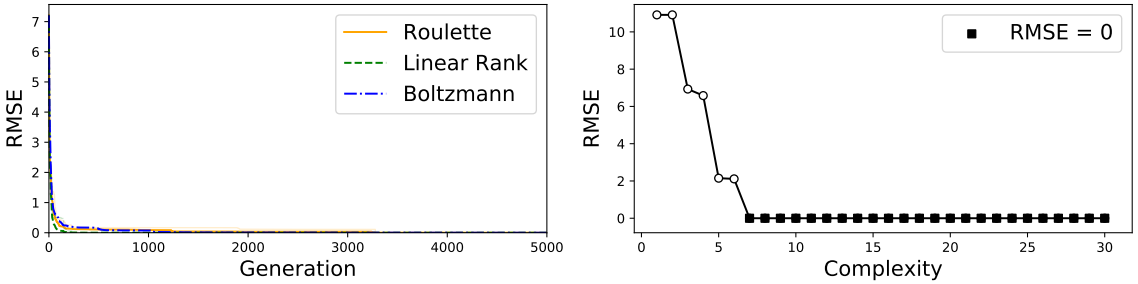


Figure: Equation I.14.4.

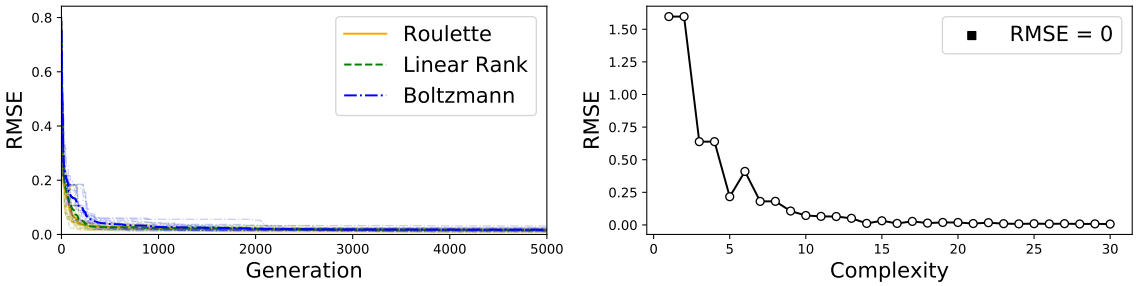


Figure: Equation I.15.3x.

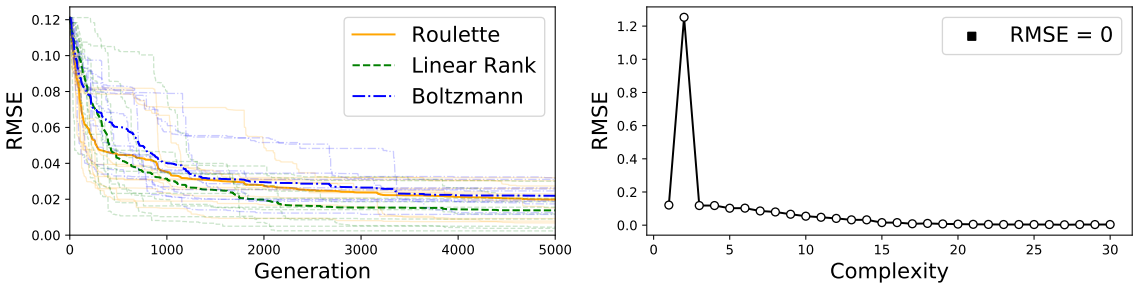


Figure: Equation I.15.3t.

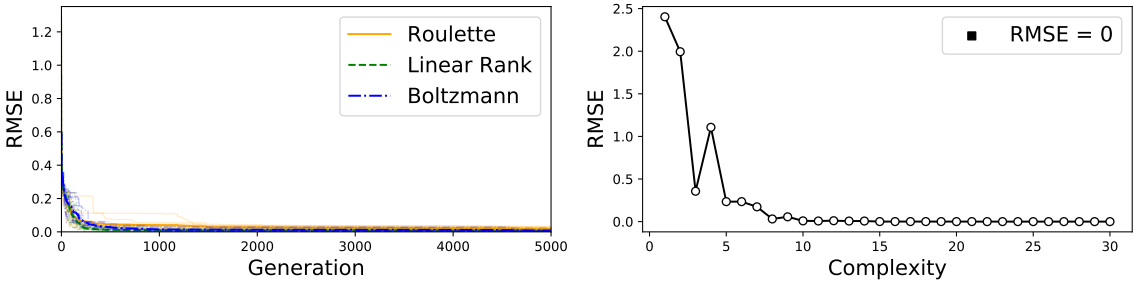


Figure: Equation I.15.10.

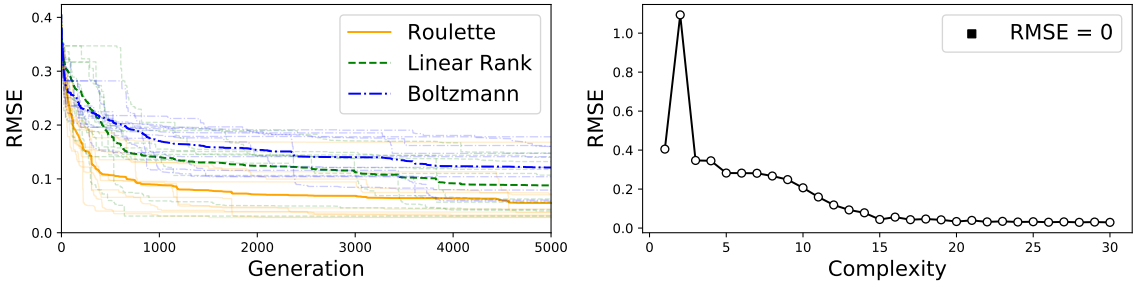


Figure: Equation I.16.6.

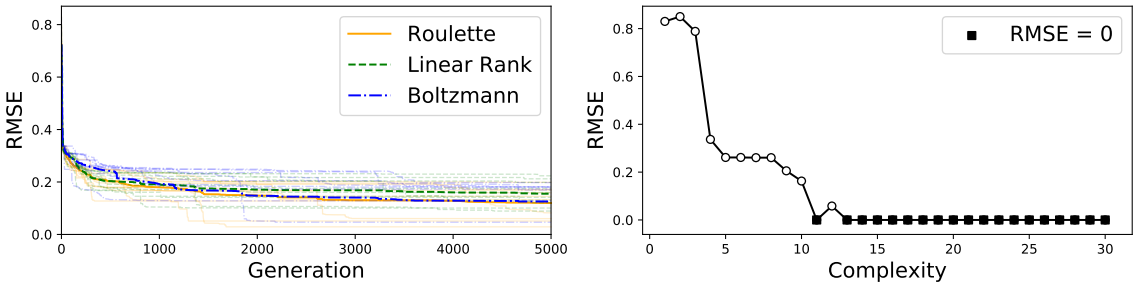


Figure: Equation I.18.4.

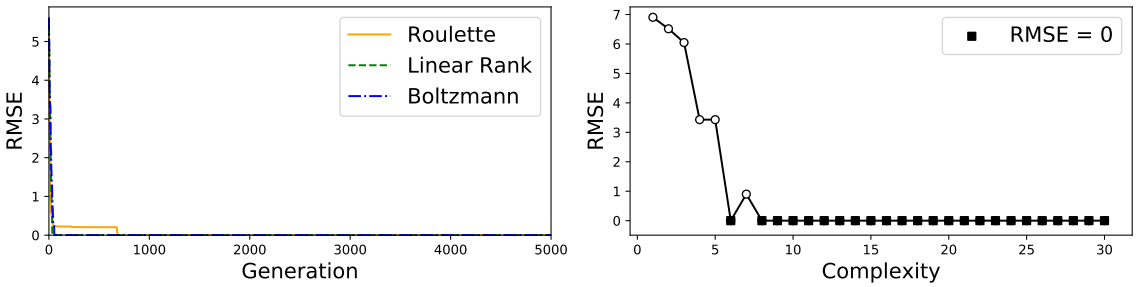


Figure: Equation I.18.12.

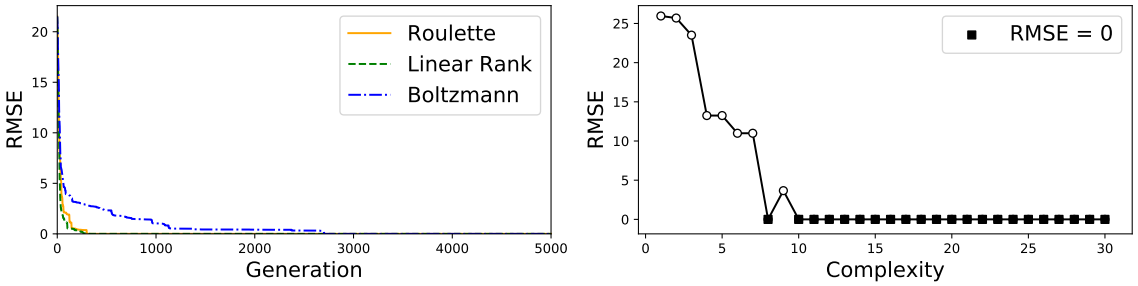


Figure: Equation I.18.16.

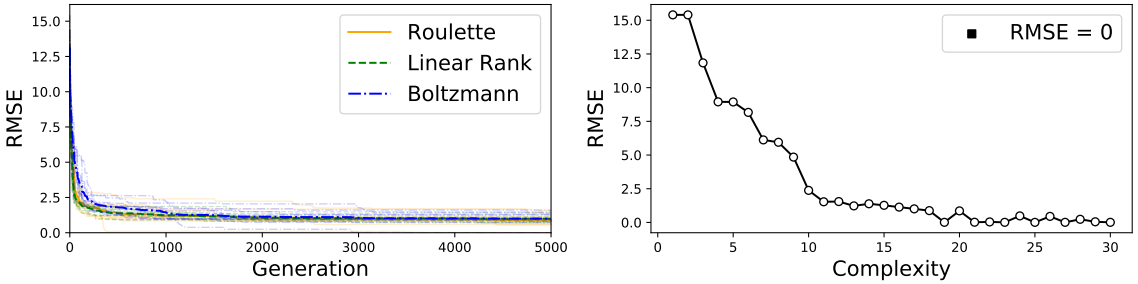


Figure: Equation I.24.6.

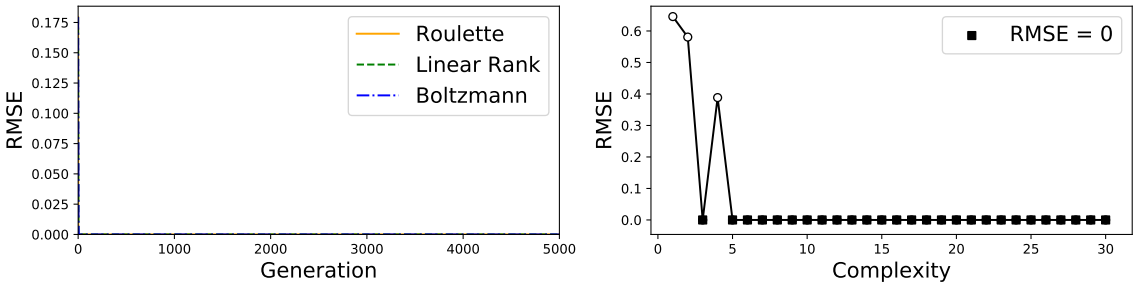


Figure: Equation I.25.13.

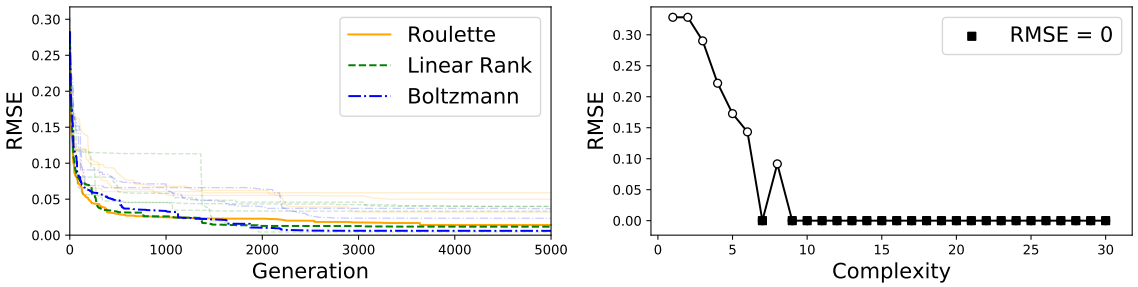


Figure: Equation I.27.6.

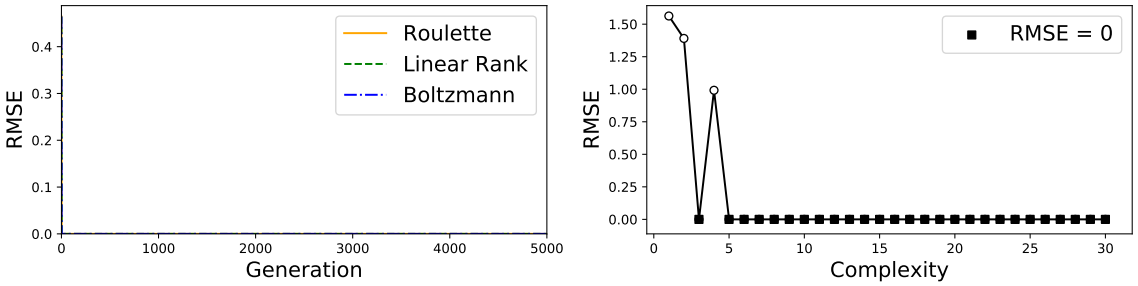


Figure: Equation I.29.4.

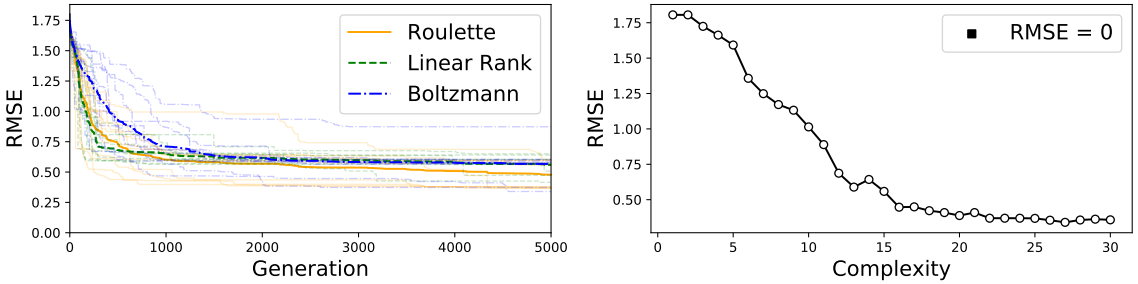


Figure: Equation I.29.16.

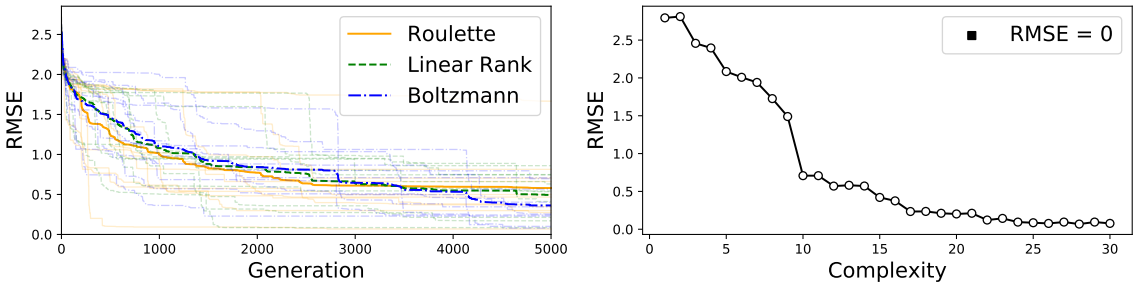


Figure: Equation I.30.3.

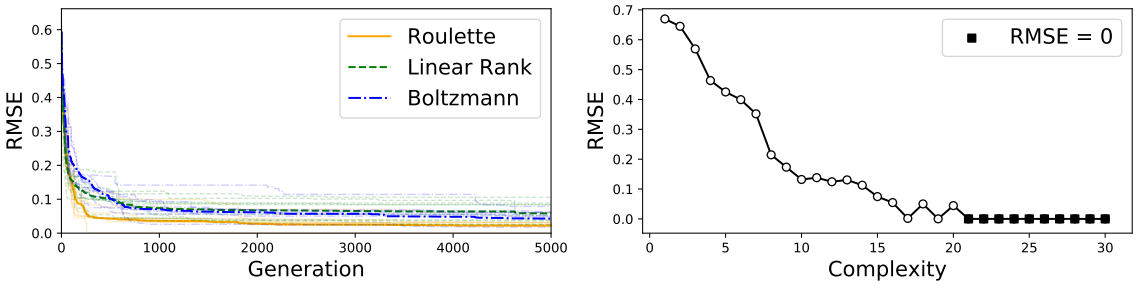


Figure: Equation I.32.5.

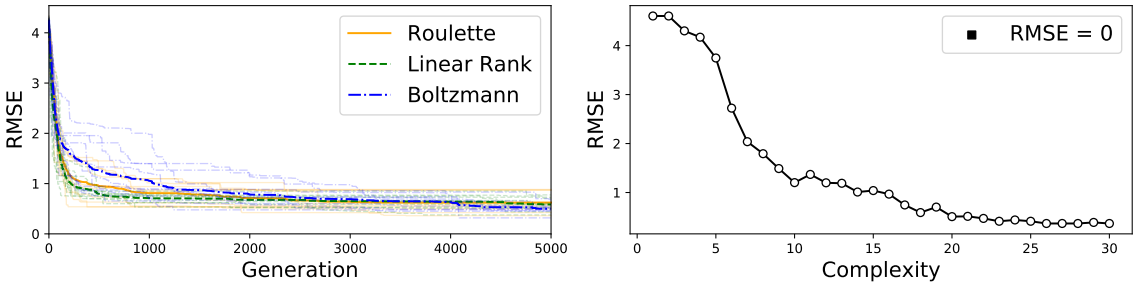


Figure: Equation I.32.17.

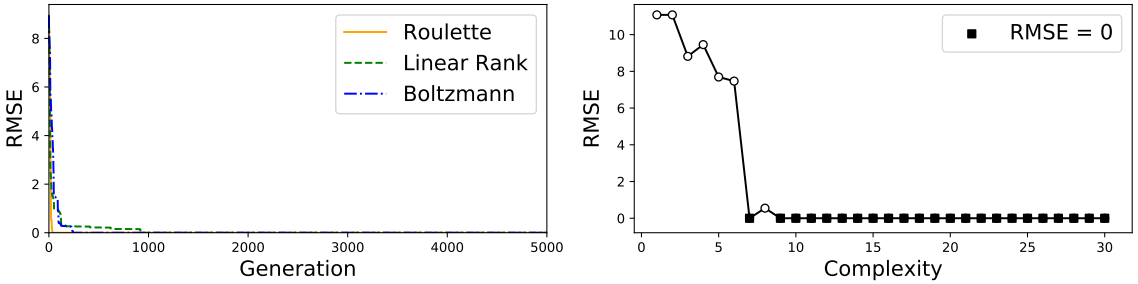


Figure: Equation I.34.8.

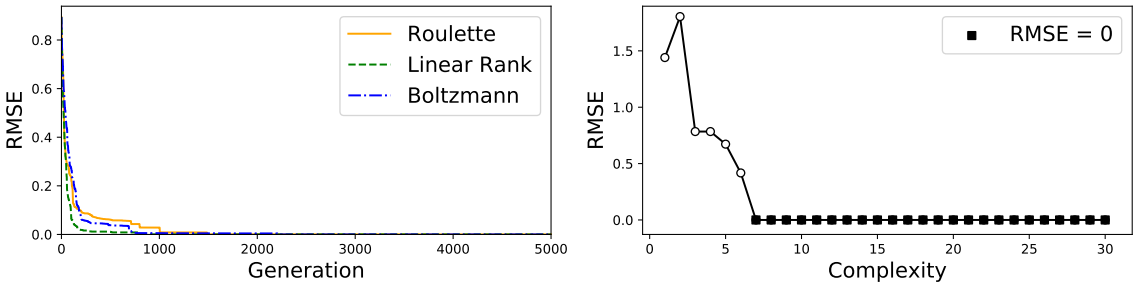


Figure: Equation I.34.10.

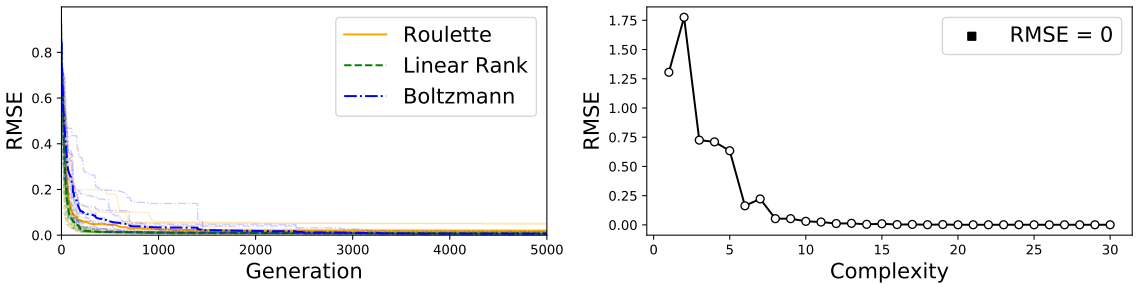


Figure: Equation I.34.14.

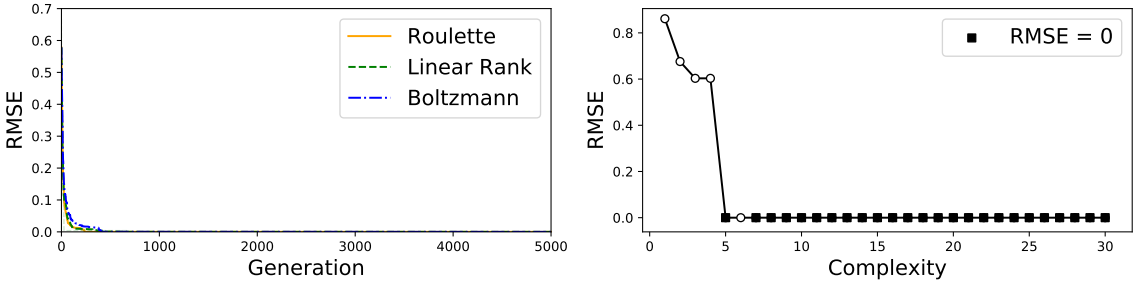


Figure: Equation I.34.27.

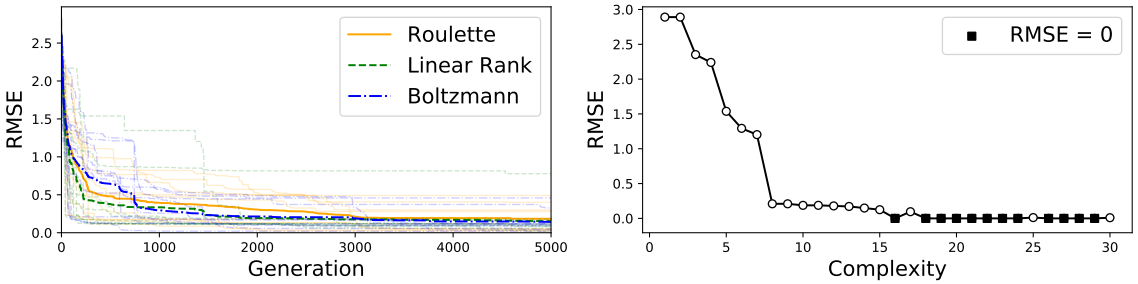


Figure: Equation I.37.4.

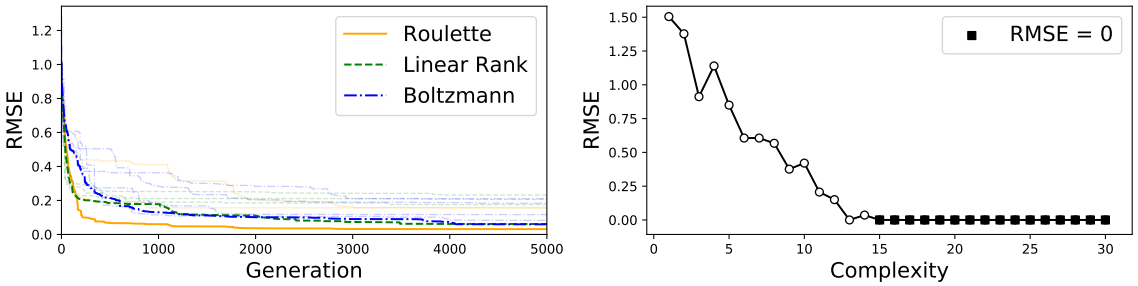


Figure: Equation I.38.12.

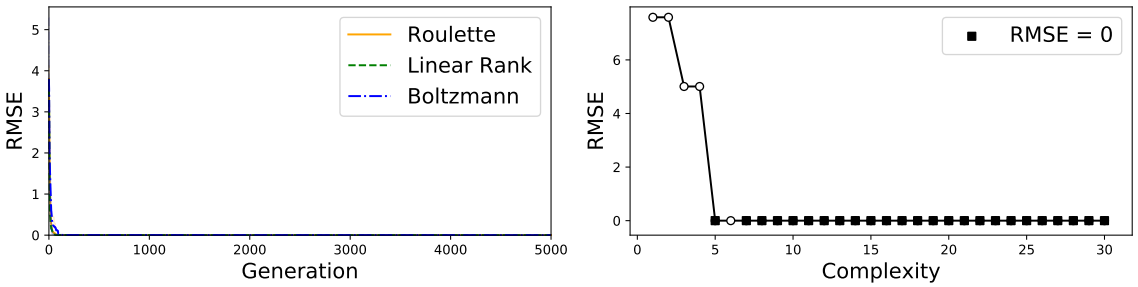


Figure: Equation I.39.10.

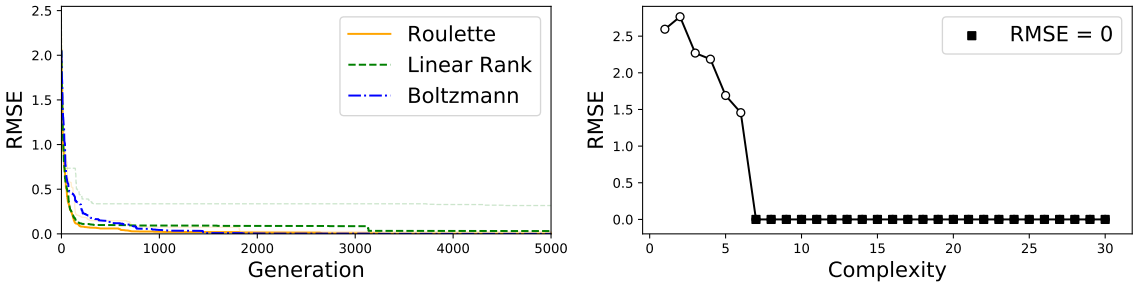


Figure: Equation I.39.11.

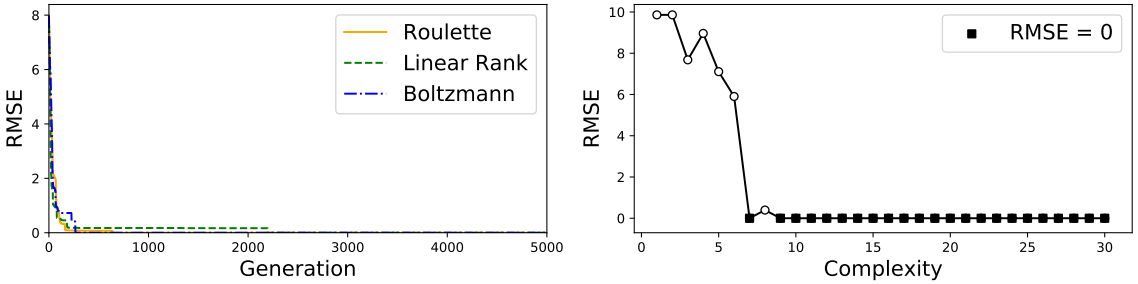


Figure: Equation I.39.22.

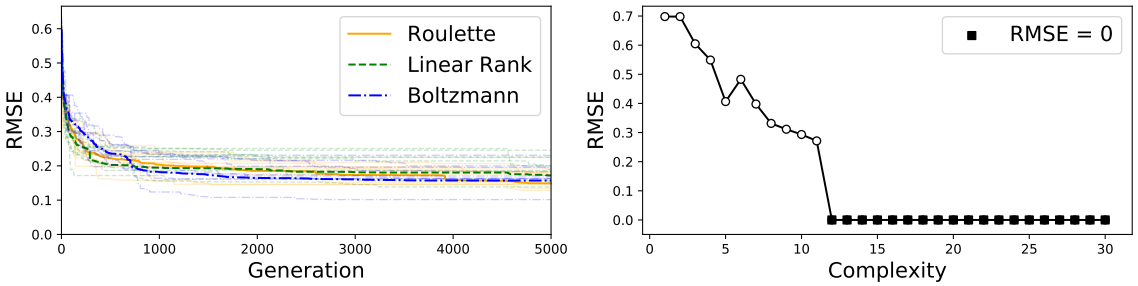


Figure: Equation I.40.1.

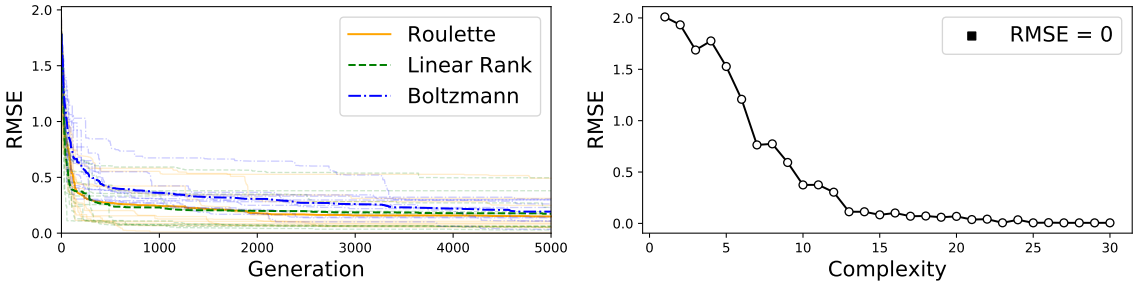


Figure: Equation I.41.16.

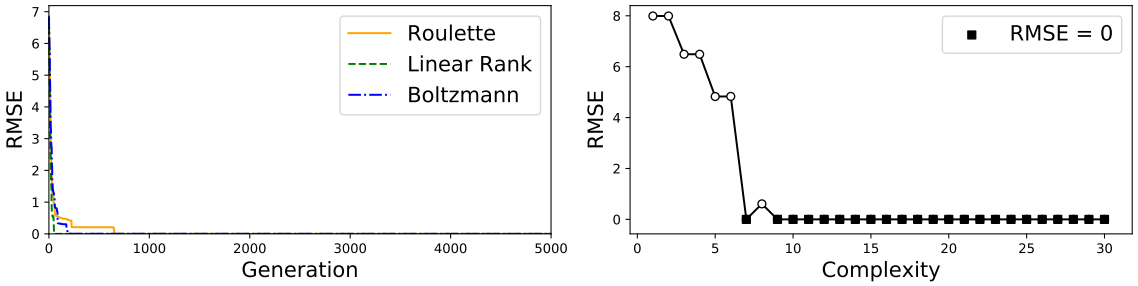


Figure: Equation I.43.16.

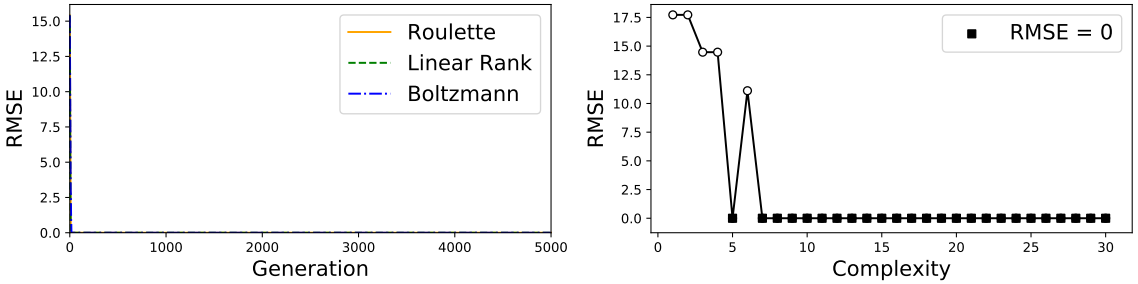


Figure: Equation I.43.31.

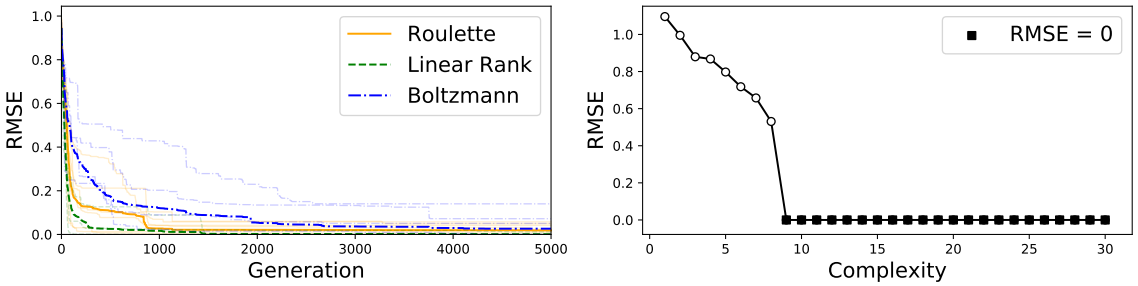


Figure: Equation I.43.43.

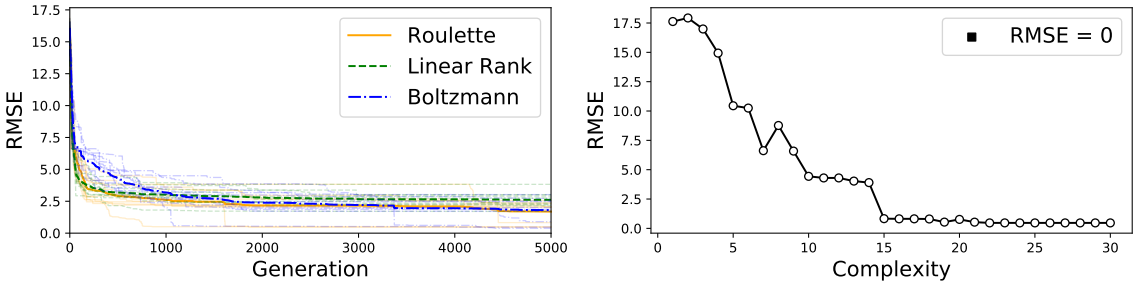


Figure: Equation I.44.4.

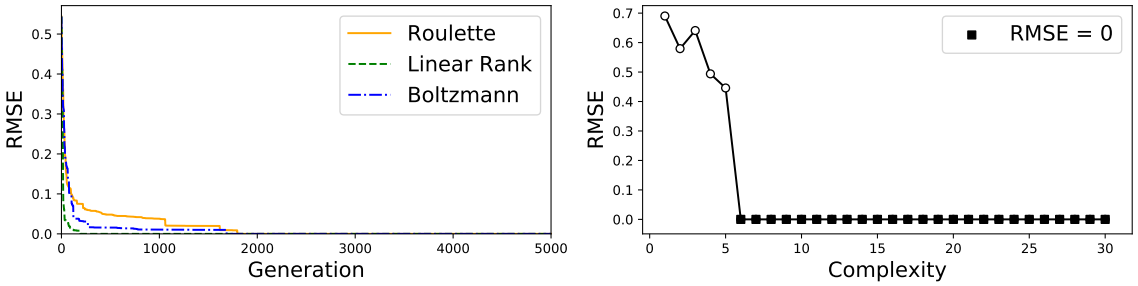


Figure: Equation I.47.23.

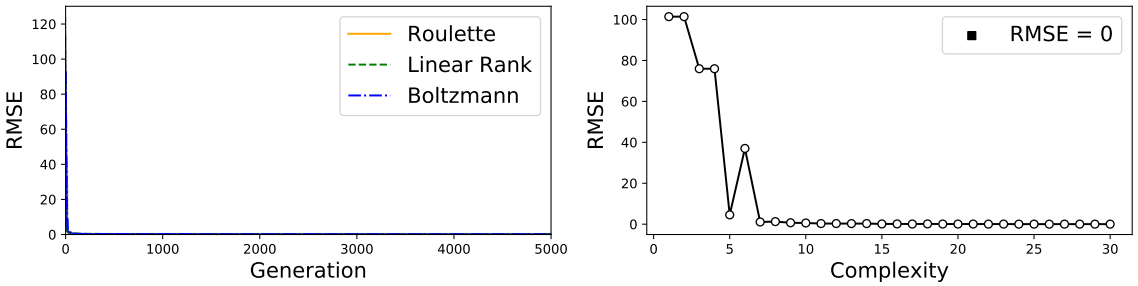


Figure: Equation I.48.20.

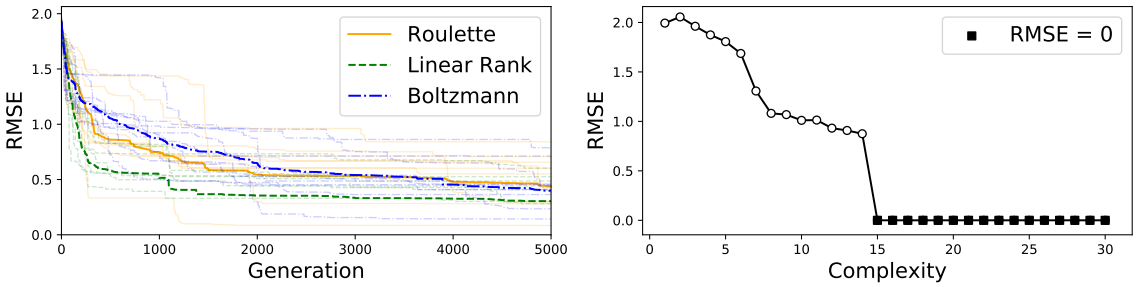


Figure: Equation I.50.26.

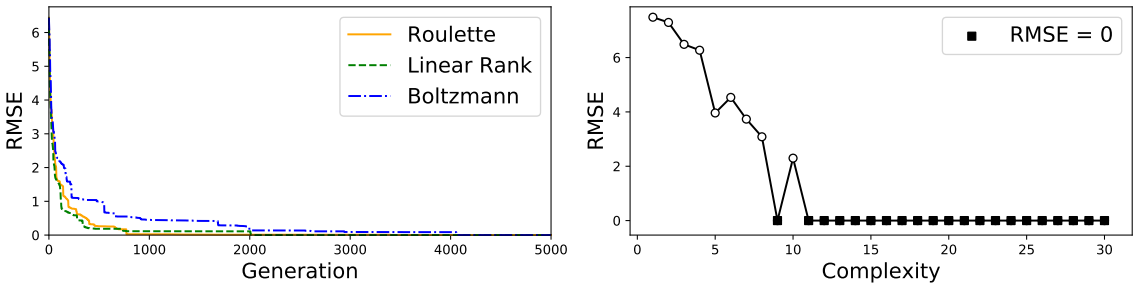


Figure: Equation II.2.42.