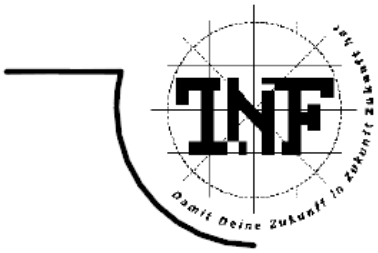
JOHANNES KEPLER
UNIVERSITÄT LINZ
Netzwerk für Forschung, Lehre und Praxis



Towards Solution Parsimony in an Enhanced Genetic Programming Process

MASTER'S THESIS

for obtaining the academic title

Master of Science

in

INTERNATIONALER UNIVERSITÄTSLEHRGANG
INFORMATICS: ENGINEERING & MANAGEMENT

composed at ISI-Hagenberg, Austria

Handed in by:

Alexandru-Ciprian Zăvoianu

Finished on:

15 July, 2010

Scientific Advisors:

Prof.(FH) Priv.-Doz. Dl. Dr. Michael Affenzeller
Prof. Univ. Dr. Daniela Zaharie

Hagenberg, July, 2010

Acknowledgment

Doing the work presented in this thesis would not have been possible without the support and guidance of a number of people.

First, I would like to thank my wonderful parents and my dear Diana for their love, constant support and encouragements, all of which have formed the cornerstone of my most important achievements in life.

Next, I am very grateful to my advisors, Prof. Dr. Michael Affenzeller and Prof. Dr. Daniela Zaharie, for introducing me to the field of nature-inspired computing and for their invaluable help in guiding my research.

I would also like to thank all the members of the Heuristic and Evolutionary Algorithms Laboratory (HEAL), and Dipl.-Ing. Gabriel Kronberger and Dipl.-Ing. Michael Kommenda in particular, for their patience, their valuable hints and a very enjoyable cooperation in our research on genetic programming.

Last, but by no means least, I would like to express my sincere gratitude to Prof. Dr. Bruno Buchberger and Prof. Dr. Dana Petcu for giving me the opportunity to conduct my Master studies within the ISI Hagenberg program and for generally encouraging me to grow, both as a student and as a young researcher.

My one-year stay in Austria has been partially supported by the EU through the SPRERS project (FP7 reference number 246839).

Abstract

In recent years, data based modeling methods have emerged as a significant aid in many fields of modern science and industry. The result of such a modeling method is usually a mathematical description of the analyzed system (e.g., a regression formula).

Genetic programming (GP) offers a means of automatically producing mathematical expressions from data by using concepts inspired by natural evolution. The main advantage of using GP for data based modeling consists in the generally high level of interpretability of the resulting solution models. Nevertheless, the parsimony of GP solution models is extremely important as very large and/or highly complex models are quite hard to interpret. GP solution size is negatively affected by the well studied phenomenon of code bloating: a rapid overall growth in the size of evolved GP models without any corresponding benefits in terms of accuracy.

In our work, we first focus on analyzing the impact of bloat on GP solution size when considering a modified GP process used for solving symbolic regression problems. This GP process contains several enhancements aimed at improving solution accuracy (e.g., an offspring selection strategy [2], a linear scaled error measure [26] and a gender specific parent selection mechanism [56]).

Secondly, based on the above mentioned analysis and on a review of several bloat control methods, we design and integrate in the enhanced GP process a bloat control strategy based on dynamic depth limiting (DDL) and iterated tournament pruning (ITP) that performs extremely well. Our method is able to achieve an average decrease of 40% in GP solution size without having any negative impact on solution quality.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

1.1 Background

Data based modeling methods have emerged as a significant aid in many fields of modern science and industry. When the goal of data based modeling is to determine a mathematical description of the behavior of a given phenomenon by analyzing a set of measurement data, we must solve a regression problem. Although the process of creating equations from data is automated, the human scientist still has the leading role of interpreting, analyzing and embedding or rejecting the proposed models. Whilst the need for model accuracy is self evident, interpretability is also a very important performance criteria: the more interpretable a given model is, the more easily its validity can be verified by a human expert.

Genetic programming (GP) offers a means of producing mathematical expressions from data by using concepts inspired by natural evolution. The main advantage of using GP for data based modeling consists in the generally high level of interpretability of the resulting models.

A real threat to GP model interpretability comes in the form of a well studied phenomenon known as *bloat*: a rapid growth in the size of evolved models without any corresponding benefits in terms of accuracy and interpretability. By contrary, when affected by bloat, models usually become so large that trying to interpret them becomes a tedious task.

At its core, this thesis is motivated by an industrial modeling problem proposed by the steel industry. The overall goal of this problem is to construct highly interpretable, parsimonious (i.e. small sized) models of high quality that can offer new insight into some of the phenomena associated with the blast furnace process.

Further motivation also steams from the desire to generally help improve the interpretabily and also the quality of the GP regression models by studying which bloat control techniques are more successful when using validation based solution designation.

1.2 Goal and Approach

HeuristicLab[55] is an optimization framework developed by the Heuristic and Evolutionary Algorithms Laboratory (HEAL) of the Upper Austrian University of Applied Sciences. The HeuristicLab implementation of GP has proven very successful on several occasions so far, as described in [57] and [32].

The main goal of this thesis is to help enhance the general interpretability of GP based symbolic regression models within the frame of HeuristicLab. Our approach for achieving this objective focused on understanding and combating/controlling the bloating phenomenon that affects the enhanced GP process implemented in HeuristicLab.

Two symbolic regression problems were used for testing: the blast furnace problem we introduced above and a problem originating from the chemical sector that was proposed as a benchmark problem in the 2010 EvoCompetitions challenge [17]. Over the course of the paper, the regression models obtained using genetic programming are compared, in terms of accuracy (and interpretability), with the results obtained using standard regression techniques like linear regression, artificial neural networks (ANNs) and support vector machines (SVM) based regression.

1.3 Original Contribution

Our original contribution consists in *developing a result orientated bloat control system* that further enhances the GP process implemented in HeuristicLab, a process that revolves around an offspring selection strategy [2], such as to generally enable GP to find small-sized solutions for symbolic regression problems.

In order to achieve this goal, we have first studied the nature of the bloat phenomenon in the context of HeuristicLab GP and then we have experimented various solutions to control its unwanted effects on solution size. A certain flavor of originality is also related to the result orientated perspective we have adopted in combating bloat.

According to the above approach, our work is split into two main stages:

- Firstly, we focused on analyzing and understanding the impact of using the strict GP offspring selection method on overall solution performance and bloat.

- Secondly, three methods of bloat control were combined and tested in order to determine which one (or which combination of methods) could lead to an ultimate improvement in solution interpretability without negatively impacting model accuracy. The analyzed bloat control methods include static depth limitation, dynamic depth limitation and pruning strategies.

Finally, *after having designed and implemented a bloat control system*, the general interpretability (parsimony) of the resulting HeurisiticLab GP solution regression models was dramatically improved without any negative impact on general prediction accuracy. Furthermore, in some cases, the bloat control strategies also seemed to drive an increase in overall GP prediction accuracy, making GP based regression a noteworthy candidate when comparing with the other standard linear and non-linear regression methods.

1.4 Outline of the Thesis

The rest of this thesis is organized as follows:

- *Chapter 2* introduces the technique of genetic programming as part of the wider field of evolutionary computation. The focus in this part of the thesis falls on describing all the major parts of the GP process. The last section of the chapter contains a brief overview of the applications of genetic programming in various scientific fields.

- *Chapter 3* starts with a presentation of the importance of white box data based modeling in the context of modern scientific discovery. Afterwards, the problem of symbolic regression is described along with some standard solving methods. Next, the concept of GP based regression is thoroughly explained. In the end, the major obstacles in data based modeling are presented along with general and GP specific methods that can help in overcoming them.

- *Chapter 4* contains a description of the two symbolic regression test problems we are using as well as an overview of the testing environment we used for experimenting.

- *Chapter 5* is dedicated to describing the bloating phenomenon outlining presumed causes and popular methods of bloat control. This chapter also presents an overview of how we define bloat control from a HeuristicLab GP solution size perspective - the so called *result orientated perspective on bloat.*

- *Chapter 6* presents details regarding the methodology we used and the initial tests we performed. The results of these initial tests define our performance baseline in terms of GP solution accuracy and size. The chapter ends with an example of how bloating affects solution size within HeuristicLab.

- *Chapter 7* is the central chapter of this thesis as it contains most of the original contribution previously described in Section 1.3.

- *Chapter 8* contains the conclusions, a summary of the obtained results and also an overview on future perspectives.

The thesis is finally completed by a bibliography.

Chapter 2

Genetic programming

2.1 What is Genetic Programming?

Genetic programming (GP) is a programming technique inspired by biological evolution that can automatically solve problems without requiring the user to specify the form or structure of the solution in advance. Viewed at the highest level of abstraction, genetic programming is *a systematic, domain-independent method for getting computers to solve problems automatically starting from a high-level statement of what needs to be done*[46].

GP belongs to the larger class of nature inspired problem solving methods called evolutionary algorithms. As the name might suggest, the theoretical foundation of evolutionary algorithms is represented by Charles Darwin's theories. Darwin (1809-1882) was a revolutionary English botanist, naturalist and zoologist who laid the foundation for the modern theory of evolution and identified natural selection as its main driving force.

His most famous work is the 1859 book "On the Origin of Species by Means of Natural Selection, or the Preservation of Favoured Races in the Struggle for Life" [12] . Darwin's main theories state that evolution surely occurs, that it is a gradual process requiring thousands to millions of years and that the primary mechanism of evolution is a process called *natural selection* (described in Chapter 4 of his book: "Natural selection; Or the Survival of the Fittest"). The process of natural selection, as identified by Darwin, is based on two principles:

1. In sexually reproducing species no two individuals are completely identical, and this variation is heritable (*heredity principle*).

2. In populations with a relatively stable number of individuals, where each individual must struggle for survival, those members that show the "best" (most desirable) characteristics are more likely to survive whilst individuals that exhibit "undesirable" characteristics are not so likely to survive. (*survival of the fittest principle*)

Taking the above into account, Darwin concluded that desirable characteristics are very likely to survive as they are passed on to offspring and thus preserved in future generations. Also, on the other hand, "undesirable" characteristics are not so likely to survive in future generations. As evolution goes on, generation after generation, desirable characteristics will become dominant among the entire population.

Genetic programming is one method of applying the principles of evolutionary biology to computer science. The idea is to *evolve* over the course of time a population of *computer programs*[1] with the goal of eventually obtaining one or more individuals (i.e. programs) that are capable of solving a given problem. As such, *generation* after *generation*, GP transforms the current population of programs, through a largely stochastic method, into a new, and hopefully better, population of programs. Like in nature, the stochastic character of evolution means that GPs cannot guarantee results. On the other hand, the same built-in randomness allows GPs to escape traps which may capture deterministic methods.

2.2 Historical Background

The first research activities in the field of GP have started in the early 1980s. Forsyth created BEAGLE [20] in 1981, a computer package that can create decision-rules by induction from a database. The principle of natural selection stands at the core of the approach as rules that fit the data badly are killed off and replaced by mutations of better rules or by new rules created by combining two better adapted rules. The rules are Boolean expressions represented by tree structures.

Cramer's paper from 1985 describes an adaptive system for generating short sequential computer functions [10]. It is widely accepted that this article is the first to describe the tree-like representation and the operators that can be used in order to genetically manipulate programs.

Even though there were several important achievements regarding GP in the 1980s, it still took some time until the method was widely received by the computer science

[1] Please note that we here define a computer program as an entity that can receive inputs, perform computations and produce output.

community. This was due to the fact that GP is a very computationally intensive technique and the available hardware at that time imposed severe restraints on the size and type of problems that could be tackled. But this was soon to change thanks to the enormous growth in CPU power that started in the late 1980s.

One of the most important publications in the field appeared in 1992. It was "Genetic Programming: On the Programming of Computers by Means of Natural Selection"[29] by John R. Koza, professor for computer science and medical informatics at Stanford University. Containing a thorough theoretical background as well as test results from various problem domains, his work has proven GP's ability to serve as an automated invention machine producing human competitive results for various kinds of problems. By now, there have been three more books on GP by Koza and his team.

Even though GP began to be successfully applied for solving ever more complicated problems in various fields, the development of a comprehensive GP theory lagged behind even in the 1990. Major advancements were made in the early 2000s and an exact GP schema analysis was first presented in "Foundations of Genetic Programming"[35] by Langdon and Poli.

2.3 The GP Process

The major stages of a GP system are shown in Algorithm 2.1. The first step is to randomly create an initial population of programs. In order to determine how good a given program (i.e. a *solution candidate*) is, GP must run it and compare its behavior to some ideal (line 3). Regardless of the specific problem at hand, this comparison must be quantified in order to give a numeric value called *fitness*. The programs that are promising (i.e. have a good fitness value) are chosen to breed (line 4) and produce new programs for the next generation (line 5). The main genetic operators that are used to create new programs from old ones are:

- *crossover*: creates a child program by combining randomly chosen parts from two selected parent programs

- *mutation*: creates a child program by randomly altering a randomly chosen part of a selected parent program

The above described steps are the means through which GP implements the darwinistic principles of biological evolution driven by natural selection. The selection stage is a means of implementing the *survival of the fittest principle*, as it makes sure that

individuals with desirable characteristics are more likely to reproduce. The crossover genetic operator helps to implement the *heredity principle* as it ensures that an offspring inherits the characteristics of both of its parents.

Another very important aspect one needs to take into consideration when simulating biological evolution is the concept of *genetic diversity*. Genetic diversity refers to the total number of genetic characteristics present in the entire population at a given time. During the run of a GP process, one must be careful not to decrease genetic diversity to such an extent that basically all the individuals in the population become almost genetically similar (before a suitable solution has been found). In this case, the evolutive process cannot take place anymore as all future offspring, created through crossover will in fact be nothing more than copies of their respective parents (*premature convergence*).

Maintaining genetic diversity can be a serious challenge as the principles of heredity and survival of the fittest are working against it by removing from future generations the genetic material of those individuals that are considered unfit at a given time, although this genetic information might prove important at a later stage. An important force that is working in favor of genetic diversity is mutation. Even if mutation is only applied to a small part of the population, it has an important diversifying effect that helps prevent premature convergence.

Algorithm 2.1 Basic Genetic Programming

1: *randomly* create an initial population of programs
2: **repeat**
3: *execute* each program and assess its fitness
4: *select*, with a probability based on respective fitness value, one or two programs from the population
5: *create* new individual programs, from the above selected individuals by applying *genetic operators* with specific probabilities
6: **until** an acceptable solution is found or some other stopping condition is met
7: **return** the best individual found

In the next sections of this chapter a more detailed description of each of the above mentioned GP stages will be provided.

2.3.1 Representation

The representation of solution candidates is one of the key elements in any evolutionary based problem solving technique. This is because on the one hand, the representation scheme should be flexible enough to store suitable solutions for the given problem, but

on the other hand, it should also be relatively simple as to allow for the design of good genetic operators. In the context of GP the solution candidates are in fact *hierarchical computer programs of variable size* [29] and the representation that is most common in literature is the *syntax tree* [29] [35].

If we consider the program $min(x - 6, x + y * 2)$, its syntax tree representation is presented in Figure 2.1. We shall use this example in order to highlight some key aspects that need to be taken into consideration in the context of syntax-tree based genetic programming.

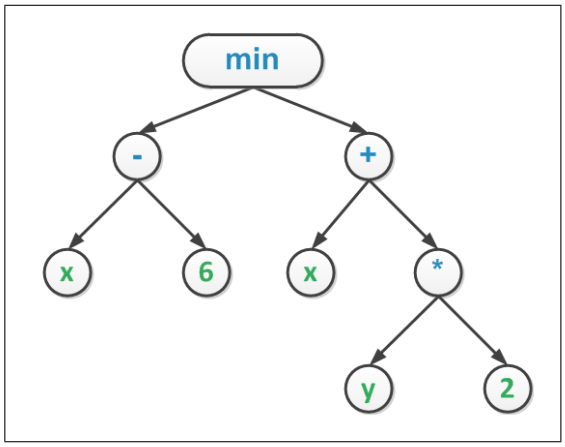


Figure 2.1: GP syntax tree representing the program $min(x - 6, x + y * 2)$

- All leaf nodes are either *variables* (x and y) or *constants* (6 and 2). In GP they are called *terminals* and they are evaluated directly (i.e. their return values can be calculated and returned immediately).

- All *function nodes* (min, $-$, $+$ and $*$) have child nodes that must be evaluated before using their values as input for their respective parents.

- The most convenient string representation for syntax trees is the *prefix* notation, also called the *Polish*[2] notation: $(min\ (-\ x\ 6)\ (+\ x\ (*\ y\ 2)))$

- in the case of fixed function arities (i.e. the number of inputs for each function is fixed and known), brackets can be dropped without causing any ambiguity: $min\ -\ x\ 6\ +\ x\ *\ y\ 2$

The syntax tree is favored as a means of solution representation in GP because it presents the important advantage of allowing for the representation of programs which

[2]Named in honor of its inventor, the Polish mathematician Jan Lukasiewicz (1878-1956).

have sizes and shapes that change dynamically (which is to be expected in an evolutionary process). Also, it is important to take into consideration that structure trees are in fact the very model in which most programming language compilers internally convert programs.

2.3.2 Evaluation

The evaluation / execution of solutions (programs) is a very important stage in GP as it is a key part in assessing the quality of a given solution candidate. This is only natural as one needs information about the results (and possibly side effects) of running a given program in order to have an idea how on well that respective program is at solving the problem at hand.

The evaluation of hierarchical computer programs represented as structure trees is done recursively, using a depth-first traversal starting from the left. We shall illustrate this by simulating the evaluation of the program in Figure 2.1.

Internal state before execution: $x = 4$, $y = -1$
Execution:
$(min\ (-\ x\ 6)\ (+\ x\ (*\ y\ 2)))$
$\Rightarrow\ (min\ (-\ 4\ 6)\ (+\ x\ (*\ y\ 2)))$
$\Rightarrow\ (min\ (-2)\ (+\ x\ (*\ y\ 2)))$
$\Rightarrow\ (min\ (-2)\ (+\ 4\ (*\ y\ 2)))$
$\Rightarrow\ (min\ (-2)\ (+\ 4\ (*\ -1\ 2)))$
$\Rightarrow\ (min\ (-2)\ (+\ 4\ (-2)))$
$\Rightarrow\ (min\ (-2)\ (2))$
$\Rightarrow\ -2$
Return value: 2; internal states after execution: $x = 4$, $y = -1$

The very simple method of evaluation presented above is another major advantage provided by the structure tree representation of solution candidates in GP.

2.3.3 Initialization

The initial stage of every evolutionary algorithm consists in initializing the population. In the case of GP, the initialization is usually done completely randomly but, in special cases, problem specific construction heuristics may be employed. There are a number of different approaches that can be used to randomly generate the initial population in

the case of syntax-tree based GP. Two of the most widely used methods for creating random initial programs are the *full method* and the *grow method*.

In both methods, the individuals are generated so that they do not exceed a user specified maximum depth D_{max}. The depth of a node in a tree is the number of edges that need to be transversed to reach that node when starting from the tree's root node (which is considered to have a depth of 0). The depth of a tree is the depth of its deepest leaf.

In *the full method* all the leaves of the generated trees are at the same depth (i.e. the method generates full trees of size D_{max}). During the generation process, nodes are taken at random from the function set until the depth $D_{max} - 1$ is reached. At depth D_{max} only terminals can be chosen. The range of program sizes (i.e. measured as the total number of nodes in a tree) and shapes this method can produce is rather limited.

In *the grow method*, nodes are selected from the whole primitive set (i.e. functions and terminals) until the depth limit is reached. Just like in *the full method*, once the depth limit is reached only terminals can be chosen. This method allows for the construction of trees with more varied sizes and shapes as the growth of any subtree can be stopped at a depth smaller than $D_{max} - 1$ by simply choosing terminals for all the leaf nodes.

Koza proposed in [29] a method called *the ramped half-half method* that aimed to combine the two GP initialization methods described above. Half the initial population is created using *the full method* and the other half is created using *the grow method*. The process uses a range of depth limits in order to ensure the creation of programs with various sizes and shapes. During the years, this method has become one of the most frequently used GP initialization techniques [15].

2.3.4 Selection

As previously mentioned, the role of selection in GP is to provide the mechanism which ensures that the *survival of the fittest principle* is respected. This means that, typically, the probability of an individual to have offspring which inherit its genetic information (i.e. traits or characteristics) is proportional to its fitness (i.e. its respective level of desirable characteristics): the better an individual's fitness value, the higher the probability its genetic material will survive within the individuals of the next generation.

There are various ways of performing selection in the context of GP. We will present just some of the most widely used selection strategies:

- *Proportional selection.* In this case each individual in the population is selected with a probability that is proportional to its fitness. The probability of selecting individual i is $p_i = f_i/\overline{f}$ where f_i is the fitness of individual i and $\overline{f}$ is the average fitness of the population. One can imagine this selection strategy as having a roulette wheel on which each individual gets assigned a certain sector; the fitter the individual the bigger the sector on the wheel it receives. When the wheel is rotated, that individual, in whose sector the ball stops in the end, is selected.

  The problem with this selection method is that it is prone to perform badly in the presence of extremely good solution candidates (*super-individuals*). This is because proportional selection assures that an individual that is extremely fit with respect to the rest of the population has very high chances of being selected each turn, and, as a result, have a lot of offspring in the future generation. As a result, in the presence of superindividuals, proportional selection leads to a rapid loss of genetic diversity in the population after only a couple of generations.

- *Linear-rank selection.* In the context of this type of selection, each individual receives a rank based on its fitness value. Taking into consideration a population of n individuals, the most fit individual is assigned the rank n while the least fit individual is assigned the rank 1. The probability of selecting individual i is $p_i = rank_i / \sum_{j=1}^{n} j$.

  Rank selection reduces the dominating effects of supper individuals but at the same time it virtually eliminates the difference in selection probability between individuals with close fitness values. Many variations of linear-rank selection are presented in literature but the central idea is that, whenever selection occurs during the evolution process, the probability to select the best individual must be a predetermined multiple of the probability to select the worst individual. In other words, one must limit the highest and lowest selection probabilities while all other probabilities are proportional to individual fitness and, as such, must lay in between the two extremes.

- *Tournament selection.* There are a lot of versions of this selection method. The most common variant is called *k-tournament.* Each time selection must be performed, k individuals are randomly drawn from the population and the fittest one among them is chosen as the selected individual.

- *Random selection.* An individual is selected from the population completely randomly without taking into consideration its fitness.

2.3.5 Genetic Operators: Crossover and Mutation

The main mechanism through which GP explores the search space (i.e. produces new solution candidates) consists of two main operators, namely crossover and mutation.

Crossover, the most important reproduction operator, produces new offspring by taking two parent individuals and swapping parts from them. In the case of syntax-tree based GP, firstly, randomly and independently, the method selects a *crossover point* (node) in each parent tree. Then, it creates an offspring by replacing the subtree rooted at the crossover point in the first parent with the subtree rooted at the crossover point in the second parent. Figure 2.2 illustrates subtree crossover using the programs $min(x - 6, x + y * 2)$ (*parent 1*) and $3 * x + y/2$ (*parent 2*). The resulting *offspring* program is $min(x - 6, 3 * x)$. The other offspring program that can be obtained by using the above parent formulae and the crossover points indicated in Figure 2.2 is $x + y * 2 + y/2$.

Usually the crossover operation is done using *copies* of the two parents in order to avoid disrupting the original individuals. Crossover points are not selected with uniform probability as, depending on the problem at hand, it might be reasonable to choose either small, medium or fairly big subtrees for crossover.

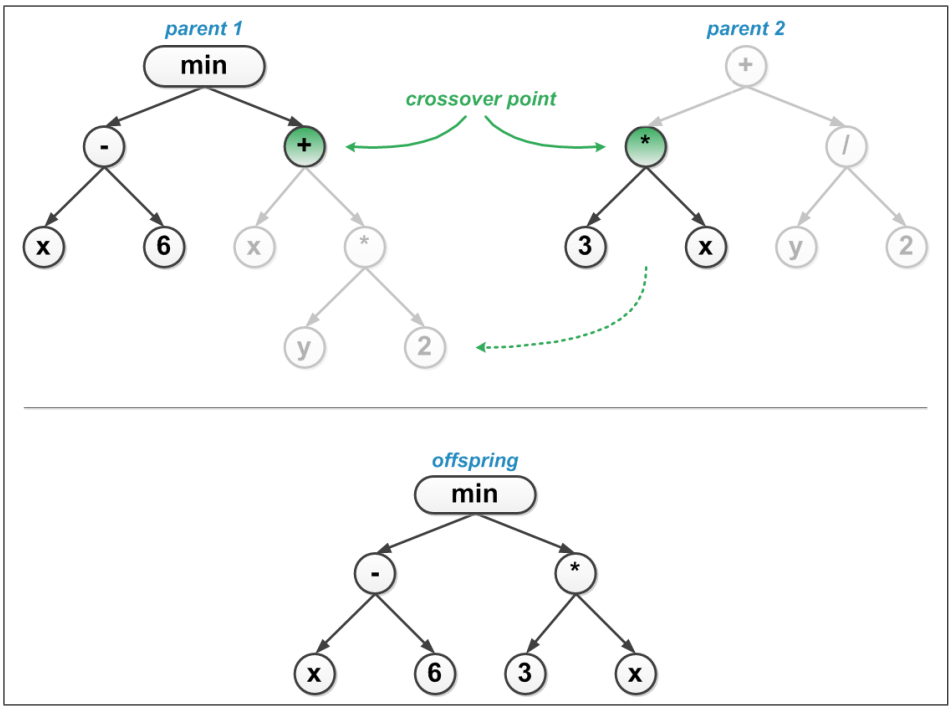


Figure 2.2: Example of subtree crossover

Generally, in the field of evolutionary computation, mutation is regarded as a means of preventing premature convergence by randomly sampling new points in the search space. In the case of genetic programming, there are two main forms of mutation:

- *subtree mutation*: in which a mutation point (node) is randomly selected and the subtree rooted in that node is either replaced by a terminal or by a randomly generated subtree

- *point mutation*: in which the function stored in a randomly selected function node (mutation point) is replaced by a different randomly chosen function with the same arity (if no other function with the same arity exists, the function node can be turned into a terminal node, or the mutation operation on that node can be canceled)

When subtree mutation is applied, exactly one subtree per selected individual must be modified. Point mutation however is applied on a per-node basis, which means that during one application of point mutation on a selected individual, each node in the tree is considered for mutation with a given probability.

In Figure 2.3 the *offspring* program named *mutant 1* illustrates subtree mutation. This program was obtained by replacing the subtree rooted in the mutation point of the program *parent* with a randomly generated subtree. The program *mutant 2* is the result of point mutation as it was generated by replacing the *subtraction function* from the program *parent* with the *multiplication function*.

In the case of the classic GP algorithm described in 2.1 the choice of which genetic operator should be used to create an offspring is probabilistic. Also, in this case, genetic operators are mutually exclusive (unlike in most evolutionary techniques where offspring are obtained via a composition of operators). Their probabilities of application are called *operator rates*. Usually, crossover is applied with a very high rate of 90% or above, whilst the mutation rate is rather small, typically in the region of 5%.

When the rates of crossover and mutation add up to a rate of p which is smaller than 100% another genetic operator called *reproduction* must be used with a rate of $1 - p$. Reproduction consists of simply selecting an individual based on fitness and inserting a copy of it in the next generation.

The usage of crossover and mutation in the context of GP can easily lead to the creation of syntactically incorrect programs. This aspect has to be taken into consideration when designing a GP system. Thus, depending on the specific problem at hand, one may choose to:

- define certain constraints that must be considered when applying crossover and mutation, constraints that prevent the creation of incorrect offspring

- implement some sort of repair strategies aimed at transforming syntactically incorrect offspring into correct ones

- simply ignore incorrect offspring and just try to derive new, and hopefully correct, offspring

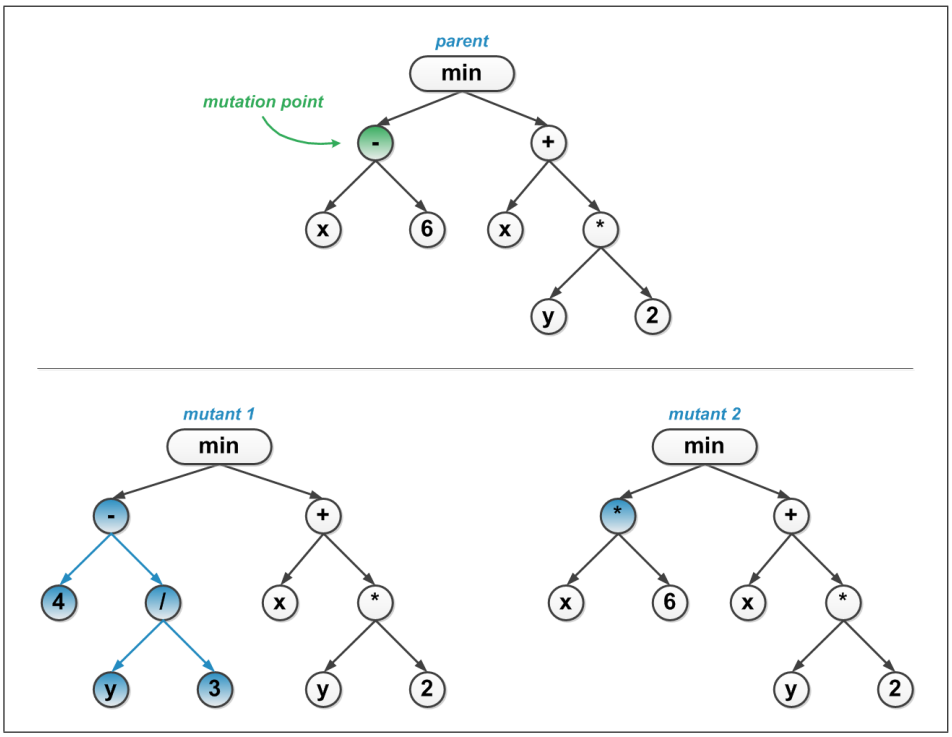


Figure 2.3: Example of subtree (*mutant 1*) and point mutation (*mutant 2*) in the context of GP

2.3.6 Termination and Solution Designation

A general, domain independent, termination criterion for a GP processes is to monitor the number of generations and terminate the algorithm as soon as a given limit is reached. Another widely used termination criterion is to stop the algorithm after the fitness values of several successive best-of-generation individuals appear to have reached a plateau (i.e. evolution has reached a phase where no new progress seems possible) [31].

Domain specific termination criteria are also very frequently used. The algorithm is terminated as soon as a problem specific success predicate is fulfilled. In practice it is

quite common to use a combination of both domain independent and domain specific termination criteria.

After the algorithm terminates, it is time to choose the result the algorithm will return. Typically, the single best-so-far individual is harvested and designate as the result of the run [31]. In some cases though, it might be necessary to return additional individuals or data depending on the problem domain.

There are some applications, like data based structure identification, in which returning the best-so-far individual produced by the GP process during training is not the optimal strategy. A far better approach is to have two disjoint data sets; one that is used for GP training, and another that is used for validation [57]. The programs produced by the GP process are eventually tested on the validation data set and the best performing individual on validation data is returned as the result of the run.

2.4 Applications of Genetic Programming

As GP is a domain-independent method, there is an enormous number of applications for which it has been used either as an automatic programming tool, a machine learning tool or an automatic problem solving engine. Based on the experience of numerous researchers over many years, Poli [46] summarized some of the main properties displayed by areas in which GP has been especially productive in:

- interrelationship between relevant variables is unknown or poorly understood
- size and shape of the solution is unknown
- availability of significant amounts of data in computer readable form
- presence of good simulators to test the performance of tentative solutions to a problem but a lack of methods to directly obtain good solutions
- conventional mathematical analysis does not, or cannot, provide analytic solutions
- an approximate solution is acceptable (or is the only result that is ever likely to be obtained)
- small improvements in performance are routinely measured (or easily measurable) and highly prized

Obtaining machines able to produce human-like results is the very reason behind the existence of the fields of artificial intelligence and machine learning. With respects to this goal, GP has demonstrated an undisputed ability to produce *human competitive results* in the fields of *algorithm synthesis* [37] [19] [53] and *electronics* (synthesis of analogue circuits [30] and of controller topology [31]).

Poli [46] constructed an overview of several fields in which genetic programming has been used to produce high quality results over the years. Here are just some examples of such results from five different areas:

- *image and signal processing*: evolution of algorithms that can detect military equipment in photo recconaissance [23], analysis of stress detection in spoken language [60]

- *financial trading and time series analysis*: modeling of agents in stock market [9], forecasting of real estate prices [24]

- *medicine biology and bioinformatics*: biomedical data mining [6], enhancement of bioinformatics techniques used for detecting gene interactions [47]

- *entertainment and computer industry*: evolution of competitive AI players [36]

- *compression*: programmatic compression [44] and lossless image compression [21]

One of the earliest general applications where genetic programming has been successfully used for is *symbolic regression* or, more generally, *data based modeling* (i.e. learning from data). More details on this topic shall be provided in Chapter 3.

Chapter 3

Symbolic Regression and Data Based Modeling

3.1 Data Based Modeling in Scientific Discovery

Modern science was formed in the period between the late 15th and the late 18th century. Keijzer [25] summarizes that before this period, scientific work primarily consisted of "collecting the observables", or recording the "readings of the book of nature itself". The new foundations of science were based on the usage of physical experiments and the application of a mathematical apparatus that tried to describe those experiments. This approach is best depicted by the works of Kepler, Newton, Leibniz, Euler and Lagrange.

This, somewhat traditional view of modern science, identifies two distinct stages: in the first one a set of *observations* of the physical system are collected and in the second one an inductive assertion about the behavior of the system (i.e. *a hypothesis*) is generated. Observations represent *specific knowledge* about the given system whilst hypothesis are attempts to *generalize* these observations. As hypothesis imply or describe observations, one can argue that they represent a means of "economizing thought" by providing a more compact way of representing data [7].

Although based on simplistic concepts, the process of scientific discovery is not at all trivial. This is due to the fact that the process of formulating scientific law or theory takes place in the context of a mental model of the phenomenon under study. If the presumed model of a given phenomenon is wrong, then any hypothesis based on this model, although mathematically correct, has a very high chance of also being faulty with regards to the studied phenomenon. As such, finding a proper conceptualization (model) of the studied problem (phenomenon) is as much a feat of scientific discovery

as the formulation and proof of the mathematical description or explanation of the phenomenon.

The beginning of the 21st century has brought an important change in the previously described scientific method. This is because of the employment of information technology in the process of hypothesis generation. The huge processing power provided by modern computers has made it possible to efficiently analyze large multi-dimensional data sets and automatically produce a large quantity of hypothesis based on these data. Aiding of scientific discovery by extracting hypothesis from data sets by means of information technology is one of the attributes of the process of *data mining and knowledge discovery*. The discipline is aimed at providing tools for converting data into a number of forms that convey a better understanding of the process that generated or produced these data.

3.1.1 Model Performance Criteria

One particular subfield of data mining is *inductive learning* or *machine learning* or *data based modeling*. Inferring models from data is the activity of deducing a *closed-form explanation* based only on observations [25]. As these observations usually represent only a limited source of information about *parts* of the studied phenomenon, the goal of inferring models that are complete, in the sense that they can explain data that are outside the range of previously encountered observations, is not at all trivial. Furthermore, one must also allow for the fact that available data usually incorporate some type of noise (e.g. they can be affected by measurement errors). When assessing the *performance* of a constructed model, one has to take into consideration two criteria of equal importance:

- The first criterion is related to the capacity of the model to "explain" *current and future data* related to the studied phenomenon. We shall generally name this performance factor as *model quality*, with the understanding that it combines the features of accuracy/fitness (i.e. have small error rates) and generality (i.e. be able to explain a wide range of data related to the studied phenomenon). A good model is a model that exhibits a low *generalization error*. The generalization error is the error of the model when it tries to explain future (unseen) data.

- The second criterion revolves around the *interpretability* of the model. Many scientists argue that confidence in a model can not be based on model quality performance alone, but must also take into account the semantic content of the model and the way this can be interpreted when taking into consideration the domain of the problem to be solved.

Model quality will be covered in more detail in Section 3.3 when we will take a look at some of the obstacles related to producing good quality models, obstacles that are general to all data based modeling techniques. The same section also contains a short overview of the most widely used techniques for ensuring model quality in data based modeling problems.

3.1.2 Model Interpretability in GP

Let us now focus on the importance of model interpretability and how it can be integrated in the process of scientific discovery.

When regarding scientific discovery as a cycle of observation, imagination, formalization and testing, one can easily see the benefits of adding an automated modeling step between observation and imagination. Automated modeling provides scientists with tentative formalizations inducted from the available data, formalizations that can inspire them to create new conceptualizations of the studied phenomenon. Furthermore, as such an automated method is biased only to the available data, it is free to propose approximate solutions to the problem, solutions that can be extremely different from contemporary thought. Analyzing such automatically generated models may lead to a better understanding and an enhanced or even different approach to describing the studied phenomenon.

Naturally, in order to be able to enhance the scientific discovery cycle in the way previously described, we need model induction techniques that are able to produce models that not only fit (have a high quality) data but that are also interpretable by scientists. *This very ability to interpret an automatically generated model should provide the additional justification needed to use the model with more than just statistical confidence.* As each model has its own syntax, the question is how can one evaluate the level to which a given syntax can capture the semantics of the phenomenon one tries to model. There is no standard answer to this question, but the attempt to coarsely classify models as being:

- *black box* - little or no semantic information
- *gray box* - some semantic information that might offer limited insight into the studied phenomenon
- *white box* - large amount of semantic information that provides the scientist with additional information about the studied phenomenon

seems to have gained support within the scientific community.

Two comments regarding the above taxonomy need to be made. Firstly, the category for a given automated model induction technique may vary drastically from problem to problem. Secondly, due to the rather elastic classification criterion, completely white box or black box models seldom exist whilst the gray box model area is well populated but quite polarized.

As we will see in the next sections, in the context of symbolic regression, genetic programming (GP) can be used as an automated white box induction technique that can produce models of comparable quality with standard best performing gray box / black box machine learning techniques like artificial neural networks (ANNs) or support vector machines (SVMs). As expected, the advantage of using GP derives from the high degree of semantic information available in the provided models. The ease of interpretation and the flexibility of the method makes GP based models very well suited for industrial modeling contexts related to structure identification problems.

However, the interpretability of a GP model is highly influenced (especially in the context of data based modeling) by the overall size of the model. A model that is much larger or far more complex than required for the modeling task at hand is of little practical use as trying to interpet it and extract meaningful information is a very intensive task that is seldomly matched by the actual information gain. Furthermore, we shall later see that oversized GP models usually also contain a high amount of intricate but largely meaningless information (see Chapter 5).

In order to take full advantage of the white-box modeling characteristic of GP we must strive to evolve high quality GP models that are as parsimonious as possible.

3.2 Regression and Symbolic Regression

As stated in Section 2.3.2, one of the key elements in solving a problem using genetic programming is finding an appropriate fitness function. In some problems, only the side effects (lateral effects) of executing the programs present interest and as such, the role of the fitness function is to compare the lateral effects of the execution of different programs in a given environment.

In many other problems though, the side effects of running programs present no interest. The goal is just to find a program whose output has some desired property (i.e. the output must match some target values). In this case the fitness function is usually

a measure of the difference between the output of the program and the ideal (i.e. the desired output values). When taking into account the fact that when using an appropriate function base [3], the programs produced by GP are in fact mathematical formulae, the main task becomes that of inducing accurate mathematical expressions from given data. These resulting mathematical expressions are nothing more than models that try to explain the given data. This is generally known as a *symbolic regression* problem.

Regression is a familiar notion for a lot of people and it can be considered as one of the simplest forms of inductive learning. Regression means finding the coefficients of a predefined function (i.e. finding a model based on that function) such that the resulting expression best fits some given data. Poli [46] presents some of the issues related with the classical approach to regression analysis. Firstly, if the fit is not good, the experimenter has to keep trying different types of functions by hand until a good model can be found. This is very laborious and the results depend very much on the skill and inventiveness of the experimenter. Another problem Poli describes is related to the fact that even expert users tend to have a strong mental bias when choosing the type of functions used for regression (e.g. linear, quadratic, exponential, etc).

Symbolic regression is an attempt to go beyond the limitations imposed by classical regression. The idea is to find both the function and its suitable parameters such that the resulting model fits the given data points without making any a priori assumptions about the structure of the function. In other words, the goal of the regression process is to discover both the symbolic description of the model and a set of suitable coefficients.

3.2.1 Formalization and Standard Solving Methods

Regression tries to determine the relationship between a dependent (target) variable y and a set of specified independent (input) variables x. Formally, the goal is to find a model consisting from a function f of x and a set of suitable coefficients w such that

$$y = f(x, w) + \epsilon \tag{3.1}$$

where ϵ represents the error (noise) term.

[3] For example one may restrict the function base to the basic arithmetic operators $+$, $-$, $*$ and /

The form of f from (3.1) is usually pre-defined in standard regression techniques like linear regression f_{LinR}, artificial neural networks (f_{ANN} - a network with one hidden layer) and support vector machines (f_{SVM}):

$$f_{LinR}(x, w) = w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_n x_n \tag{3.2}$$

$$f_{ANN}(x, w) = w_o \cdot g(w_h x) \tag{3.3}$$

$$f_{SVM}(x, w) = w_0 + w_1 \Phi_1(x_1) + w_2 \Phi_2(x_2) + \cdots + w_n \Phi_n(x_n) \tag{3.4}$$

In linear regression w is the set of coefficients $w_0, w_1, w_2, \ldots, w_n$ and they are usually determined using least square regression.

In the case of ANNs we have to use an auxiliary transfer function g which is usually a sigmoid function (e.g. the logistic function $P(t) = \frac{1}{1+e^{-t}}$). Here the coefficients w represent the weights from one layer of the neural network to the other. For example in (3.3) w_h are the weights from the input nodes to the hidden nodes and w_o are the wights from the hidden nodes to the output nodes [57].

SVMs-based regression is a very powerful inductive learning technique as it is generally able to produce very good quality solutions for nonlinear regression problems. In SVMs-based regression, the first step is to map the input onto an n-dimensional feature space using some fixed, nonlinear, transformations like $\Phi_1, \Phi_2, \ldots, \Phi_n$. Hopefully, in the new feature space the problem becomes linear and, as such, a linear model can be constructed in the feature space (see (3.4)) by using the principle of structural risk minimization [54]. The last step is to remap the obtained linear model back onto the original input space. The transformation to and from feature space (i.e. the "kernel trick") allows SVMs to use a linear regression technique to solve non-linear regression problems. A downside of the method is the fact that the intermediate linear model is not explicitly defined (and accessible) and as such, the interpretability of the regression models produced by SVMs lies majorly only in the interpretability of the used kernel and of the used parameters.

3.2.2 Symbolic Regression Using Genetic Programming

In contrast to the techniques presented in the previous section, when applying genetic programming to symbolic regression, the function f which is searched for is not presumed to be of any pre-specified form. In GP the idea is to use low-level functions that are combined to more complex formulae during the run of the evolutionary process. Considering an initial set of primitive functions $f_1, f_2, \ldots, f_n$, the overall functional form induced by GP can take a variety of forms. Usually this set of primitives contains standard arithmetic functions like addition, subtraction, multiplication and division.

X	-4	-3	-2	-1	0	1	2	3	4
Y	15.8	6.7	0.2	-3.7	-5	-3.7	0.2	6.7	15.8

Table 3.1: Input data for example experiment

Need be the case, trigonometric, logical, and more complex functions can also be included. *The flexibility of GP based regression comes from the fact that no assumptions are made regarding the form of the searched function and that evolution is used to guide the search towards the functional form that best explains the given data.* This also means that the search process is free whether or not to consider certain input variables. As such, GP based regression is also able to perform variables selection (leading to dimensionality reduction) [25].

For instance, a functional form induced by GP could be:

$$f(x, w) = f_6(f_1(f_3(x_1, w_2), f_4(f_2(x_2), x_3), f_5(x_3, w_1)), w_3) \tag{3.5}$$

When using concrete primitives for the set low-level functions abstractly marked by $f_1, f_2, \ldots, f_n$ as well as concrete values for the coefficients $w_1, w_2, \ldots, w_n$ we could get:

$$f_1(x, w) = -(*(1.6, x), 4)) \equiv 1.6 * x - 4 \tag{3.6}$$

$$f_2(x, w) = +(-8, *(x, x)) \equiv x^2 - 8 \tag{3.7}$$

For a better understanding of how GP works in the context of symbolic regression, we shall now provide a small example.

Let us assume that we have made 9 observations of a certain phenomenon. For simplicity we shall also assume that the gathered data are error free. Each observation focused on recording only two variables pertinent to the experiment: X and Y (see table 3.1). May Y be the dependent variable and X the independent one. As such the goal is to find a model that attempts to explain the data of Y based on the data of X. This example is based on synthetic data meaning that the values of the set Y have been obtained from the values of the set X by simply using the formula:

$$y = f_{target}(x) = 1.3 * x^2 - 5 \tag{3.8}$$

which thus is also the target of our symbolic regression problem.

In GP based symbolic regression solution candidates are evaluated by applying them to X, thus obtaining the estimated values E. Finally, in order to asses the fitness of the candidate, the estimated values are compared to the known original (*target*) values Y. We shall consider formulae (3.6) and (3.7) as solution candidates in a GP process

E_1	-10.4	-8.8	-7.2	-5.6	-4.0	-2.4	-0.8	0.8	2.4
E_2	8	1	-4	-7	-8	-7	-4	1	8
E_3	21.6	10.4	2.4	-2.4	-4	-2.4	2.4	10.4	21.6

Table 3.2: Estimations produced by GP models for example experiment

that is aimed at solving our example regression problem. Furthermore, by considering these two formulae as parents and performing crossover (as shown in Figure 3.1), we will obtain the offspring formulae:

$$f_3(x) = 1.6 * x^2 - 4 \tag{3.9}$$

When applying f_1, f_2 and f_3 on X each will generate a set of estimations: E_1, E_2 and respectively E_3. These estimated values are shown in Table 3.2. The graphical representations of the functions f_1 (*parent 1*), f_2 (*parent 2*), and f_3 (*offspring*) as well as that of the goal function f_{target} (*target formula*) are shown in Figure 3.2.

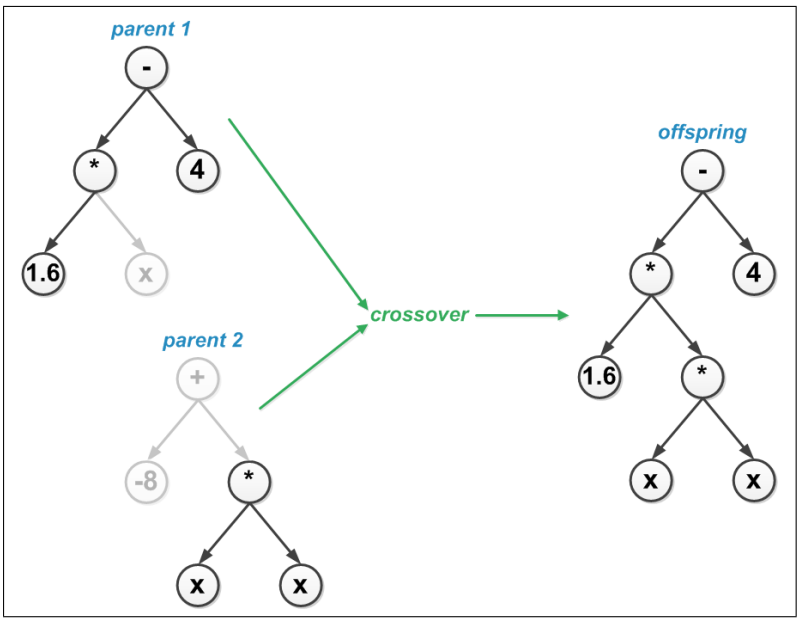


Figure 3.1: GP crossover of example regression formulae

The task of genetic programming in symbolic regression is to find the combination of primitives, input variables and coefficients that can provide estimation values that are as close as possible to the desired target values.

There are several ways in which one can measure the error between estimated and target values in a regression problem. One of the simplest and the most frequently

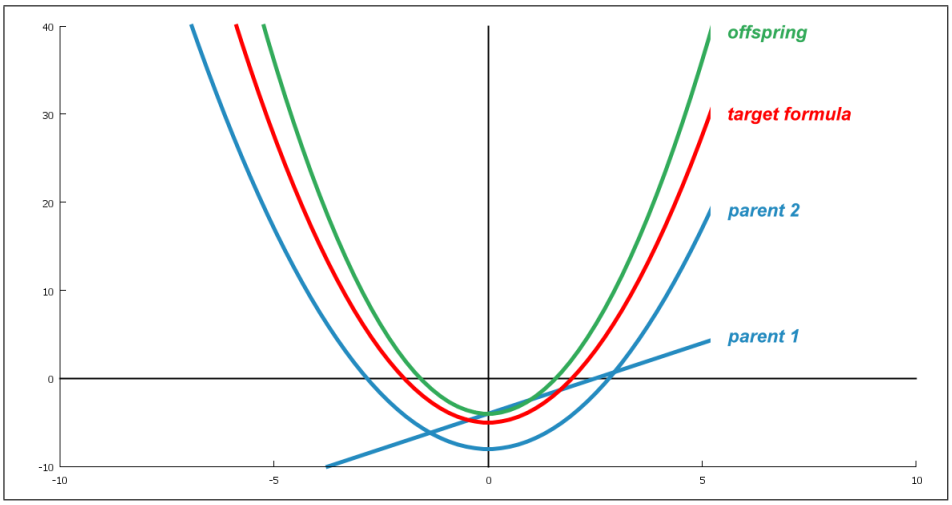


Figure 3.2: Plot of regression formulae used as an example

used methods is the mean squared error (MSE) function. The mean squared error of two vectors Y and E, each containing n values is:

$$MSE(Y, E) = \frac{1}{n}\sum_{i=1}^{n}(Y_i - E_i)^2 \tag{3.10}$$

Obviously, when using MSE as a fitness estimator in GP symbolic regression, the goal is to find a solution candidate with a fitness value as small as possible. When calculating the fitness of the previously considered solution candidates f_1, f_2 and f_3 as $MSE(Y, E_1)$, $MSE(Y, E_2)$ and respectively $MSE(Y, E_3)$ we obtain:

$$MSE(Y, E_1) = 19.95 \tag{3.11}$$

$$MSE(Y, E_2) = 6.76 \tag{3.12}$$

$$MSE(Y, E_3) = 3.73 \tag{3.13}$$

As one might have expected by analyzing the plot in Figure 3.2, from the three possible candidates, function f_3 (the offspring formulae) is the most precise estimator of the target regression function.

The search for formulas that minimize a given error measure is the major goal of GP based regression, but the shape and the size of the solution can also be integrated into the fitness estimation function. For example in some cases, the focus on model simplicity might be just as important as the one on model estimation quality, as simpler models can be interpreted easier, thus offering more meaningful information about the studied problem.

3.3 Obstacles in Data Based Modeling

There are two major general obstacles when considering data based modeling: *noise* and *overfitting*. In this section we will provide a description of these problems as well as some solutions that can aid in overcoming them.

The general regression formula (3.1) offers a very clear insight on what the impact of noise on data is. In the context of data based modeling we observe that, because of various reasons, additional unwanted values are added to the original data. This disturbing additional data is called noise. For example, when considering a physical system that we monitor using a sensor array, noise might be introduced by one or more malfunctioning sensors, or from some sort of interference on the communication channel(s).

If the level of noise is too high, it can be detected and dealt with in a data pre-processing stage, but if the disturbance is of relatively low intensity, it is very hard to detect and combat.

In the field of machine learning, overfitting is the phenomenon of excessively fitting models to data. As previously mentioned, inductive learning is done using training data: sets of records that have values for both input (independent) and the target (dependent) variables. The problem is that it may happen that too complex models might be trained, models that become adjusted to particular features or samples of the training data. This becomes a huge inconvenience when taking into consideration the previously presented problem of having some noise in the data, as an overfitting model becomes more prone to modeling the noise (errors) in the training set rather then trying to explain the whole studied phenomenon using the training data. The effect of overfitting are models with low training errors and high generalization errors.

As mentioned in the first section of this Chapter, one of the two main goals of data based modeling is to produce high quality, robust models that are both accurate and able to generalize well on future data. In order to achieve this goal, we must be able to prevent overfitting but as the complexity of the system to be modeled is usually unknown, there is no rule how to generally avoid this unwanted phenomenon. However, several techniques that can help avoid overfitting have been proposed.

One of the simplest ideas is the *hold-out method*. It is based on performing a split of the initially available data into two *disjoint* data sets: *a training data set* and a *test data set*. Model induction will be carried out using only the training data set. The fact that the resulting model is constructed by only taking into consideration the training set means that it is unbiased towards the test test and as such the test set can be used

in order to estimate the generalization capacity of the induced model. A clear sign of overfitting, when using the hold-out method, is the fact that a model exhibits very good fitness on the training data and poor fitness on the test data (i.e. high generalization error).

When choosing the size of the training and testing data sets one must take into consideration the following:

- The more training samples (and the less test samples), the better the training accuracy, but the worse the ability to generalize well and fit test data (model is more prone towards overfitting).

- The more test samples (and the less training samples), the lower the training accuracy, but the better the ability to generalize although predictions on test data will also suffer from a lack of accuracy (as the obtained model is too simple to explain the studied phenomenon).

When confronted with rather small data sets, the requirement that the training and testing sets must not overlap (must be disjoint) is extremely constricting.

A more advanced concept that tends to extend the hold-out principle is *n-fold cross validation*. The basic idea is to partion the initial data set into n disjoint folds (intervals or fractions). Model training will be performed n times, each time using $n-1$ folds as training data and one fold as test data. At the end, the n test accuracy results (the results obtained by measuring each turn model accuracy on the left-out fold) are averaged. This resulting average test accuracy is considered a good indicator of both the overall fitness of the model as well as its ability to generalize well. One of the downsides of the technique is the fact that it is rather time consuming (training must be performed n times).

Another downside of cross-validation (far more important in the context of GP based data modeling) is the fact that the method is usually suited only for modeling techniques that presume an implicit functional form of the model (see Section 3.2.1) and only try to find suitable parameters such that the resulting model best fits the data. By systematically trying parameter settings and using n-fold cross validation with implicit functional form techniques, an experimenter has a very high chance of finding the parameters that define the model(s) with the best performance on the given data (both in term of accuracy and generality).

When applying GP to data based modeling, n-fold cross validation is not a very appropriate performance measure due to the fact that in each of the n training runs, the

method can induce n very different models based on very different functional forms. Although by averaging test accuracies an image of the overall performance of using GP based data modeling on the given problem can be presented, in this case, n-fold cross validation does not present a means of detecting overfitting behavior in individual models.

An approach that has proven to be very suitable for controlling overfitting in the case of GP splits the data set into three partitions [57]:

- *Training data* that are the real basis of the algorithm and that are actively used by the modeling process.

- *Validation data* that are available to the algorithm but that are usually not used in order to assess the fitness of solution candidates during the evolutionary optimization process. As such, trained models are unbiased towards it and validation data can be used for detecting the overfitting phenomenon or for finally selecting the model that is returned by the GP process (i.e. the process returns the model that exhibits the best accuracy with regards to the validation data). When having the policy to return the model with best validation accuracy one also needs to take into consideration that the solution produced by the GP process has also become biased towards the validation data set. The same holds, if during the run of the process, model validation results are used to somehow steer the evolutionary process in one direction or another.

- *Test data* that can not be considered by any part of the training algorithm. As described in previous paragraphs they are treated as "new data", unavailable for training, that can only be used in order to determine whether or not the GP process was able to generate robust, high performance models.

In short, in the case of GP data partitioning, the training set is used to create or train models, the validation set is used for model selection and the test set is used to estimate the actual generalization error on future data.

Chapter 4

Modeling Scenarios and Testing Environment

4.1 Modeling Scenarios

As mentioned in the introductory chapter of this thesis, for testing purposes we have considered two symbolic regression problems. The first one comes from the steel industry and it concerns the production of molten iron in the blast furnace process, whilst the second one is a benchmark problem for genetic programming (GP) based on data related to a chemical experiment.

The blast furnace problem proposes two different modeling scenarios for the same target variable and the benchmark problem consists of a single modeling scenario.

4.1.1 The Blast Furnace Problem

In [32] Kronberger presents an overview of the blast furnace process, as well as the results of early modeling attempts for some tasks related to the process. The author argues that although physical and chemical reactions in the blast furnace are well understood on a high level of abstraction, many more subtle inter-relationships between injected reducing agents, burden composition, scaffolding, top gas composition and their effect on the the produced molten iron or slag are not totally understood. Knowledge about these inter-relationships can be used to improve various aspects of the blast furnace process like iron quality, amount of consumed resources or overall process stability.

Input material enters the blast furnace at two locations. Raw materials (ferrous oxides in the form of sinter or lump ore) and coke are charged in alternating layers at the top of the blast furnace. In the lower area of the blast furnace, hot blast (air heated at around 1200 ° C) and reducing agents are inserted through a number of tuyeres. Reducing agents include heavy oil, pulverized coal, natural gas, tar and plastic pallets.

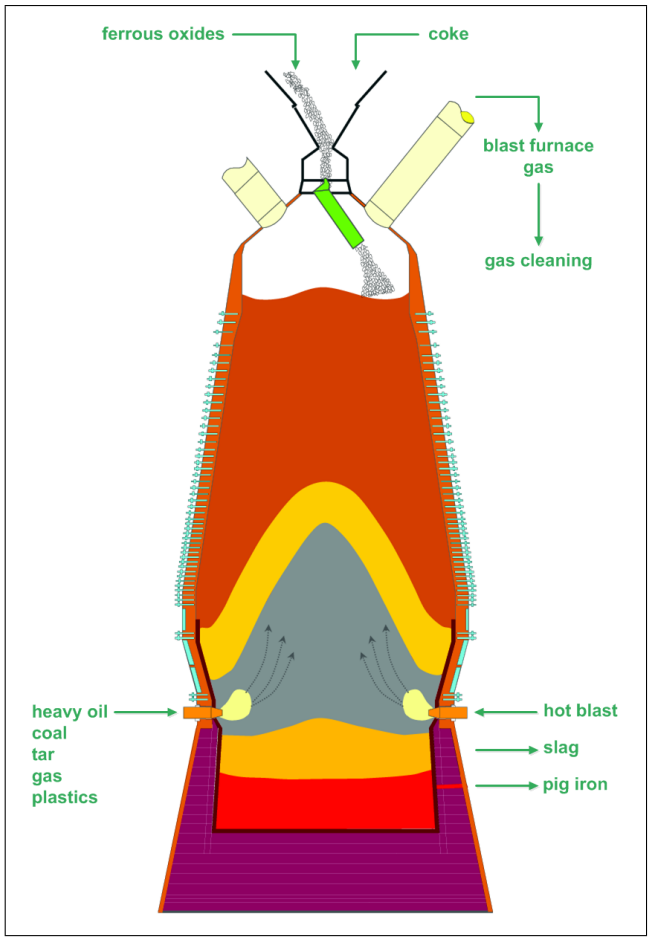


Figure 4.1: Schematic diagram of the blast furnace

Inside the furnace, the inserted hot blast (containing oxygen) reacts with the carbon (in the form of coke) to produce carbon monoxide and heat. Carbon monoxide then reacts with the ferrous oxides to produce molten iron and carbon dioxide. Carbon monoxide is also created in the Boudouard reaction from carbon dioxide and carbon. Hot carbon dioxide, unreacted carbon monoxide and nitrogen (residue from the hot blast) pass up through the furnace as fresh raw materials and coke travel down into the reaction zone. The resulting metallic iron is melted and carburized. The carbon content of the resulting iron is around $4 - 5\%$.

The products of the process are molten iron and the liquid byproduct called slag (both at the bottom of the furnace) and blast furnace gases at the top of the blast furnace.

Process component	**Nature of variables**
Sinter	physical and chemical analysis
Coke	physical analysis
Charging plan	amount and weight distribution of input materials and coke
Hot blast	amount, pressure, temperature, fraction of oxygen
Reducing agents	amount of heavy oil, coal, gas, plastics
Hot metal	weight and chemical analysis
Slag	weight and chemical analysis
Blast furnace top gas	pressure, temperature and composition
General blast furnace data	temperature, melting rate, pressure
Temperature measuring rod	core, middle and border temperatures
Cooling losses	power consumption of wall and bottom cooling elements

Table 4.1: Measured and calculated variables for the blast furnace process

The resulting molten iron (pig iron), which is used for the production of steel, and the liquid slag, that is used in the construction industry, are removed by tapping. Figure 4.1 presents a schematic diagram of the blast furnace process.

In our case, the raw data set contains measured and computed values for a number of 45 variables that describe various physical and chemical attributes of the input material, reducing agents and produced output as well as some control parameters and parameters that describe the internal state of the blast furnace. An overview of the used variables in presented in Table 4.1

Based on the raw data, an hourly data set was prepared that contains around 5200 rows. This data set covers roughly one year of blast furnace activity. The initial hourly data set also contained a number of missing or incorrect values that were the result of periods of time when the furnace was in a faulty state or underwent maintenance. The records containing erroneous data were removed during an initial preprocessing stage.

The concrete problem revolves around the prediction of the amount of carbon monoxide used during the blast furnace process. As expected from the description of the process, the amount of used carbon monoxide is strongly related to the amount of raw material and properties of the hot blast. As carbon monoxide is also involved in the Boudouard reaction, which largely controls the gas atmosphere in the furnace, knowing more details

regrading other factors that may influence carbon monoxide utilization is useful for better controlling the blast furnace process.

On trying to solve this problem we shall employ two modeling scenarios created using input from experts in the steel industry. The first one, named *A5*, contains a selection of 14 variables, whilst the second one, named *A6*, uses 44 variables.

4.1.2 The EvoCompetitions 2010 Symbolic Regression Problem

EvoStar [17] is the name of Europe's annual premier co-located events in the field of Evolutionary computing. It is composed of a number of independent conferences like EuroGP (specialized in genetic programming), EvoCOP (specialized in combinatorial optimization), EvoBIO (specialized in data mining for bioinformatics) and several workshops collectively entitled EvoWorkshops.

Since 2010, EvoStar contains a new section called EvoCompetitions. This section proposes a series of benchmark problems from different fields with the purpose of challenging participants to develop evolutionary inspired algorithms that can provide high quality solutions. The second test problem that we use in this thesis is the symbolic regression problem proposed in the 2010 edition of EvoCompetitions. As we shall later see in Section 6.2.3, this is in fact a fairly linear regression problem that has proven to be rather challenging for GP based regression methods (this is the main reason it was chosen as a benchmark problem in EvoStar).

The modeling task for this problem consists in the prediction of expensive but noisy laboratory data regarding a chemical composition (the output) to 57 cheap process measurements, such as temperatures, pressures, and flows (inputs). The selected equation has to include the most sensitive inputs relative to the output. The case is based on data from a real industrial application from an American company operating in the chemical sector.

The initial dataset has around 1000 records. After a preprocessing stage, the first few rows which contained suspicious values were dropped from dataset. The mean value of the dependent variable across the entire data set is 2.980 and the standard deviation is 0.338.

4.2 Testing Environment

In this section of the thesis we shall describe the two software environments that we have used for the associated testing activity. Emphasis is put on the HeuristicLab implementation of the GP process and on the major enhancements it proposes with regards to the standard GP process. We shall also give a brief presentation of the WEKA platform which contains implementations for various regression methods (three of which have been selected for comparison). The specific settings used for the HeuristicLab tests as well as for the WEKA tests are presented in detail in Section 6.2.

All the HeuristicLab and WEKA tests have been carried out using the HEAL blade system. The system is constructed on a Dell PowerEdge M1000e chassis with each blade having 2 Intel Xeon E5420 QuadCore CPUs clocked at 2.50 Ghz, 32GB of RAM and a 72 SAS GB. The operating system is a 64bit version of Microsoft Windows Server 2008 Enterprise Edition.

4.2.1 The GP Process in HeuristicLab

HeuristicLab[55] is an optimization framework developed by the Heuristic and Evolutionary Algorithms Laboratory (HEAL) of the Upper Austria University of Applied Sciences with the given goal of aiding the design, testing, implementation, application and monitoring of various (and mostly evolutionary inspired) heuristic optimization algorithms.

The HeuristicLab implementation of GP has proven very successful for solving data based modeling tasks on several occasions so far [57] [32]. The GP process implemented in HeuristicLab includes several enhancements/features aimed at improving the overall quality of the generated symbolic regression models and the convergence time of the algorithm.

Gender Specific Parent Selection

First off is the concept of gender specific parent selection [56]. It is based on the idea of male vigor and female choice, as it is considered in the model of sexual selection discussed in the field of population genetics. In GP, this idea was adapted and easily implemented by considering two different selection schemata for the selection of the two parents required for each crossover: the usage of random selection for one parent and

of a selection strategy with far greater selection pressure (e.g. proportional or linear rank selection) for the other parent.

The gender specific selection concept not only brings the GP process a little bit closer to its biological archetype, but also presents relevant advantages in flexibility. By using two different selection strategies, one can influence the selection pressure of a GP run more precisely. It is thus possible to better control the interplay between genetic forces supporting or reducing diversity in a more directed way, allowing for a better tuning of the GP behavior.

The Offspring Selection Strategy

The second major enhancement of the standard GP process is the offspring selection strategy [2]. The idea is to not only consider the fitness of the parents in order to produce an offspring during the evolutionary process. As such, a given offspring is accepted as a member of the population of the next generation if and only if it outperforms the fitness of its own parents. Figure 4.2 schematically displays the main

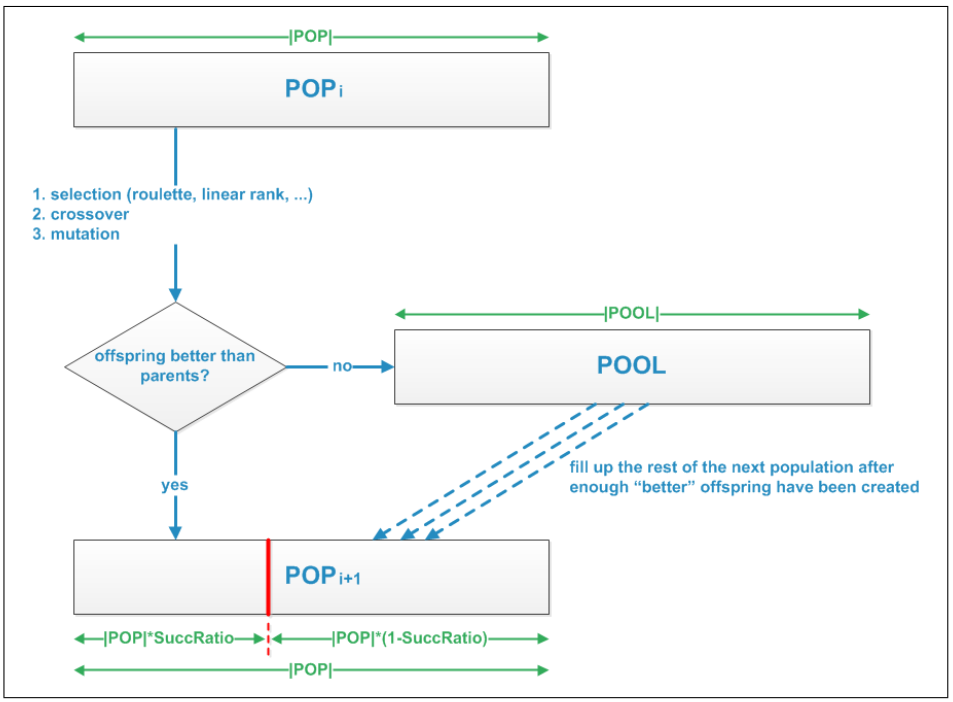


Figure 4.2: Offspring selection model

aspects of the offspring selection model. When employing offspring selection, the same as for conventional GP, offspring are generated by parent selection, crossover and mutation. The difference consists in the introduction of a second (offspring) selection stage.

A variable called success ratio (*SuccRatio*) indicates the ratio of the next generation (POP_{i+1}) members that must outperform their respective parents. As long as this ratio is not fulfilled, offspring are created and the successful ones are inserted into the next generation, while those that do not outperform their parents are stored in a rejection pool (*POOL*). When the success ratio for the next generation is reached, the rest of the members of the generation are randomly chosen from the rejection pool ("lucky losers").

Within the new selection model, selection pressure (*SelPres*) is a measure for the effort that is necessary to produce a sufficient number of successful offspring. It is defined as the ratio of generated offspring to the population size:

$$SelPres = \frac{|POOL| + |POP| * SuccRatio}{|POP|} \tag{4.1}$$

Setting an upper limit for selection pressure, gives a quite intuitive termination criterion: the algorithm terminates when it is no longer possible to find a sufficient number of offspring that outperform their parents.

Linear Scaled Error Measure

Another GP enhancement in the HeuristicLab implementation that is worth mentioning is the use of a linear scaled error measure (as described by Keijzer in [26] and [27]):

$$MSE_{scaled}(Y, E) = \frac{1}{n}\sum_{i=1}^{n}(Y_i - (a + bE_i))^2 \tag{4.2}$$

where the linear scaling coefficients a and b are defined by:

$$a = \bar{Y} - b\bar{E} \tag{4.3}$$

$$b = \frac{cov(Y, E)}{var(E)} \tag{4.4}$$

The advantage of using a scaled error measure, as opposed to the traditional approach (see Section 3.2.2), lies in "the change of the view" that the selection operator has on the worth of an individual expression. As MSE_{scaled} rescales the expression on the optimal slope and intercept, selection will favor expressions that are close in shape with the target, instead of demanding first that the scale is correct. Further discussions and a comparison of solution qualities achieved using this principle can be found in [27].

Highly Flexible Initialization Method

The initialization method used in HeuristicLab is based on the PTC2 algorithm proposed by Luke [38]. This is because the PTC2 algorithm offers more flexibility than the *grow* and *full* methods as it allows for the creation of populations that respect a given size and depth distribution, as well as for the usage of likelihood weights for functions and terminals.

Point Mutation Parametrization

The GP process in HeuristicLab relies on *point mutation parameterization* for the terminal set. In our GP implementation, all variables in a given model are weighted by a certain factor. These weights, as well as the used constants, can be either integral or real valued and they are initialized and modified according to a uniform distribution (based on a minimum and a maximum value) or to a Gaussian one (defined by average μ and standard deviation σ). The point mutation operation (see Section 2.3.5) is responsible for modifying the associated values of constants and variable weights during the run of the GP algorithm.

4.2.2 The WEKA Platform

WEKA [22] (Waikato Environment for Knowledge Analysis) is a freeware machine learning platform written in Java and developed at The University of Waikato, New Zealand.

WEKA contains a comprehensive collection of algorithms for data analysis and predictive modeling (i.e. classification and regression algorithms that in WEKA are indiscriminately referred to as *classifiers*). Together with the preprocessing and visualization techniques available in WEKA, the wide range of modeling techniques makes the platform a very powerful candidate when it comes to analyzing and comparing the performance of different data based modeling techniques for a given problem.

In section 3.2.1 we have listed three standard methods that are commonly used for solving linear and non-linear regression problems: linear regression, artificial neural networks and support vector regression. All the tests involving these methods, presented in this thesis, have been carried out using their respective WEKA implementation:

- *LinearRegression* - WEKA implementation of linear regression

- *MultilayerPerceptron* - WEKA implementation of artificial neural networks regression (based on the back propagation algorithm)

- *SMOreg* - WEKA implementation of support vector regression (C-SVM regression)

Other implementations of the above mentioned regression methods exist in WEKA (for example in the case of support vector regression we also have the option of using a wrapping of the well known LibSVM implementation [8]). Our choice for the specific WEKA classifiers listed above was influenced by the speed of the implementation, the ease of configuration and the overall method performance after a round of preliminary tests.

Chapter 5

Bloat - The major problem in GP

5.1 Bloat - Background

Poli [46] notes that starting in the early 1990s, researchers began to notice that in addition to progressively increasing the average and best fitness values, genetic programming (GP) populations also presented other, less favorable, dynamics. In particular, they observed that the average size (in number of nodes) of the programs in the population started growing at a rapid pace after a certain number of generations. Usually, this increase in program size was not matched by any corresponding increase in fitness. This phenomenon has come to be known as *bloat*.

Bloat has an immense negative impact on genetic programming as large programs are computationally expensive to further evolve, are hard to interpret and usually generalize very poorly. Because of this, in many cases, one can say that bloat effectively leads to the stagnation of the evolutionary process.

It is important to make the distinction that code growth is a healthy result of the evolutionary process when searching for better solutions. For example, GP runs usually start with populations of small, randomly generated programs, and it is necessary for these programs to grow in complexity (i.e. size) in order to be able to comply with the given fitness criterium. So, program growth in size does not equal bloat. We should define the bloating phenomenon as *program growth without (significant) return in terms of fitness* [46].

5.2 Theoretical Aspects

Code bloat in GP has been intensely studied and, over the years, several theories have been proposed to explain various aspects related to it. However, to date, there is no universally-accepted theory that can explain the wide range of empirical observations regarding bloat. In the next sections we shall briefly present some of the most important theoretical aspects regarding bloat.

5.2.1 Bloat and Introns

In their 2009 article [48] Silva and Costa present a well documented overview on the history of the bloating phenomenon in GP. They remark that when Koza published his first book [29], most of the evolved programs he presented contained pieces of code that did not contribute to the solution and that if removed would not alter the produced results. Koza's solution was to impose fixed depth limits to the trees created by crossover and to manually edit the solutions at the end of each GP run in order to simplify expressions and remove redundant code.

In 1994, Angeline studied this phenomenon in more depth [4]. He noticed the ubiquity of redundant code segments within GP individuals and, using a slight biological similarity, named them *introns*. The most interesting part of the analysis presented the positive aspects that introns might have on the GP process. Angeline remarked that introns provided crossover with syntactically redundant constructions where splitting could be performed without altering the semantics of the swapped subtrees. He argued that introns emerge naturally from the dynamics of GP and that "it is important then to not impede this emergent property as it may be crucial to the successful development of genetic programs".

According to Luke [39], introns are found under two distinct forms: *inviable code* and *unoptimized code* (see Figure 5.1). *Inviable code* cannot influence the fitness of an individual no matter how many changes it suffers, either because it is never executed or because its return value is ignored. *Unoptimized code* is viable code that contains redundant elements that can be removed from an individual without affecting its overall fitness. Whilst *inviable code* also offers defense from crossover, *unoptimized code* is highly susceptible to variations of its structure and, if altered by crossover, its return value can greatly influence the fitness of the individual.

For example, if we consider the two individuals:

$$M_1 = +(y, \underline{*(0, -(4, 2))}) \equiv y + \underline{0 * (4 - 2)} \tag{5.1}$$

$$M_2 = +(x, \underline{+(-2(-(4, 2)))}) \equiv x + \underline{((-2) + (4 - 2))} \tag{5.2}$$

where x and y are some arbitrary chosen dependent variables, then the underlined sections (subtrees) are superfluous and can be removed without affecting the fitness of the individuals. We can modify any of the subtrees rooted at the nodes within the fragment $-(4, 2)$ without affecting the fitness of M_1 (i.e. *inviable code*). However an attempt to modify any of the subtrees rooted at the nodes within the fragment $+(-2(-(4, 2)))$ is very likely to also modify the fitness of M_2 (i.e. *unoptimized code*).

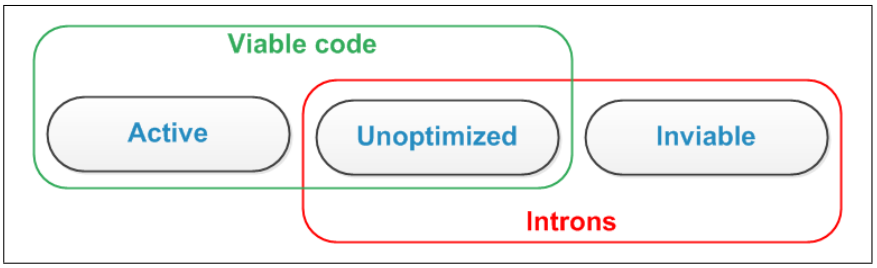


Figure 5.1: A Venn diagram of Luke's perspective on the various kinds of code in GP

Although the concepts of introns and bloat appear to be very similar, we shall now see that there are some very important shades of difference into this matter.

5.2.2 Five Theories of Bloat

In spite of several studies ([3] [43] [49]) that largely confirmed Angeline's initial remark that introns might generally offer some sort of protection from crossover and subtree mutation, the negative effects of intron proliferation are extremely serious as computational power is wasted on manipulating code that has no contribution to the quality of the solution, instead of being used for the effective search of better solutions. Because of this, intron and bloat control has become a very active research area in GP and several theories concerning why bloat occurs have been advanced. One should note that these different explanations regarding bloat are not necessarily contradictory and, whilst some appear to be generalizations or refinements of others, several theories seem to complement each other very well. We shall now provide a short description of the most important five theories regarding bloat:

1. The *replication accuracy theory* introduced by McPhee and Miller [42] states that, especially in the later stages of evolution, the success of a GP individual depends

on its ability to have offspring that are functionally similar to it. As such, individuals that contain introns (and thus are usually larger than the mean size of the population) have a selective advantage as the introns provide destructive crossover with genetic material where swapping can be performed without affecting the effective code. Destructive crossover is nothing else but the classic subtree crossover operator defined in Section 2.3.5. In the context of bloat theories, it is called destructive because, in the more advanced stages of the run, it seldom creates offspring that are better than their parents.

Soule studied the replication accuracy theory in the context of tree based GP and confirmed it by observing that using a non-destructive crossover can reduce the amount of introns in the GP population and, implicitly, the average population size [50]. The non-destructive crossover Soule proposes retains an offspring only if is better than its parents in terms of fitness and, if not, a parent is chosen to replace it in the population of the next generation. Luke proved Soule's findings wrong [39], arguing based on the difference in behavior between *inviable code* and *unoptimized code*. Luke showed that while non-destructive crossover helps in reducing the *inviable code*, it also helps in the proliferation of other types of bloated code to such an extent that, actually, the overall mean size of the population is increased. He explained Soule's initial results by the fact that code growth was actually delayed by the amount of parent replication along the generations.

2. The *removal bias theory* was proposed by Soule and Foster in [51] and is based on the observation that when using a tree representation for GP, *inviable code* usually resides low in the syntax tree, in smaller then average subtrees (called inactive subtrees). Whenever crossover is applied inside such an inactive subtree, it produces an offspring that has the same fitness as its parent, as code added inside an inactive subtree also becomes inactive [51]. In order to obtain this effect, there is an upper limit on the size of the excised code: it must be smaller than the original size of the inactive subtree. As there is no limit on the size of the inserted subtree, the obtained offspring retains the fitness of its parent, but is also usually larger than it. This eventually leads to growth in the average program size.

3. In [39], Luke proposes the *modification point depth theory*. This can be viewed as a generalization of the removal bias theory [52] as it is based on the fact that there is a strong correlation between the depth of the node a genetic operator modifies in a parent and the effect on the fitness of the resulting offspring: the further the node is from the root of the tree (the deeper), the smaller the change in fitness (regardless of the fact that the modification takes place in an inactive subtree or not). Because of the destructive nature of regular crossover and subtree mutation,

smaller changes (created by deeper modification points) will, over the course of evolution, benefit from a selective advantage which will lead to larger and larger individuals (the deeper the modification point, the smaller the excised subtree - there is again a removal bias).

More importantly, Luke argues within this theory, that introns do not cause bloat and that the growth in solution size is determined by the search for better fitness. He states that the propagation of *inviable code* is a consequence and not a cause of size growth.

4. Langdon and Poli [34] proposed the *nature of search space theory* as a possible explanation of bloat in GP. This theory is based on the observation that when using a variable length representation (like a syntax tree) there are a lot of ways to represent the same program: some are shorter and some are longer. A regular fitness evaluation function will attribute the same fitness to all, if their behavior is the same. Given the destructiveness of regular crossover, when better solutions become hard to find, there is a selection bias towards programs that have the same fitness as their parents. Because there are many more longer ways to represent a program than shorter ways, there is natural drift towards longer solutions and thus bloating occurs.

5. The most recent theory regarding bloat is the *crossover bias theory* by Poli et al. [45]. It explains code growth in tree-based GP by presenting the effect that standard crossover has on the distribution of tree sizes within the GP population. When applying subtree crossover, the genetic material removed from the first parent is inserted into the second parent, and vice versa; consequently the mean tree size remains unchanged. After repeated crossover operations, the population approaches a particular distribution of tree sizes where small individuals are much more frequent than the larger ones (for example crossover generates a very high amount of single node individuals). Because very small individuals are usually unfit, the larger programs have a selective advantage and, thus, the mean individual size within the GP polulation increases.

   An overview of the strong theoretical and empirical evidence that supports the crossover bias theory is presented in [48].

5.2.3 Discussion

In [48], Silva also argues that, when taking into consideration all these theories, there is one thing that if removed would cause bloat to disappear: *the search for fitness*

(e.g. experiments have shown the absence of bloat when selection is random [5] [34]). Ironically, this is also the only thing that if removed would render the whole process useless. As removing the main cause of bloat does not seem to be a real option, research has also been focused on trying to control the magnitude of the phenomenon.

Another observation to be made when comparing the above presented theories, is that *they are primarily structured around the destructive behavior of standard genetic operators*. When switching to a protected crossover method, like the one proposed by Soule [50], the most interesting theoretical aspects are the ones suggested by Luke [39]: protected crossover virtually eliminates *inviable code* but also seems to increases the amount of *unoptimized code*.

5.3 Classic Bloat Control Methods

Several practical approaches aimed at counteracting/controlling the bloat phenomenon have been proposed over the years. We shall now present a short description of the most important of them (based on an overview of the subject from [46]).

5.3.1 Size and Depth Limits

One of the simplest methods used to control code growth is to impose size or depth limits on the generated offspring programs [29]. Usually, in implementing this concept, after applying a genetic operator, one checks if the resulting offspring respects the size and the depth limit. If the offspring exceeds one of these limits, it is disregarded and one may choose either to return one of the parents or retry the genetic operation using the same or different parents.

Tests have shown that when using depth thresholds, the population fills up with "bushy programs" where most branches reach the depth limit (i.e. full trees) [41]. By contrast, size limits produce populations of "stringy programs" that tend to all approach the size limit [41]. A somewhat surprising recent finding is that the usage of size limits actually speeds code growth in the early stages of the run [13]. The reason for this is that imposing size limits biases the population towards a very uneven size distribution like the one suggested in the crossover bias theory.

When using size or depth limits, programs must not be so tightly constrained that they are not able to express accurate solutions. Poli [46] argues that as a rule of thumb, one should try to estimate the size of the minimum possible solution (that can be achieved

using the given terminal and function set) and add a percentage (e.g., 50-200%) as a safety margin. Depending on the complexity of the problem, some trial and error may also be required.

In [48] Silva and Costa propose a method for dynamically adjusting the depth and size limits during runs. The tests they performed showed that dynamic depth limits behaved very well, being able to produce results that were as accurate as the ones obtained using fixed depth limits, while using significantly smaller trees. On the other hand, the size limits were not so successful as they always produced results that were significantly less accurate than the ones obtained using fixed or dynamic depth limiting.

5.3.2 Anti-bloat Genetic Operators

There have been several attempts to control bloat by using specially designed genetic operators. Crawford et al. [11] proposed a *size fair crossover* method. The main idea is to constraint the choices that can be made during the execution of the crossover operation such as to prevent growth. Like in the classic crossover operation, a crossover point is selected randomly in the first parent. The size of the subtree to be excised is then calculated. This value is used to constrain the crossover point in the second parent, such that the size of the second excised subtree isn't "unfairly" large.

Several *size fair subtree mutation* operators have also been proposed. As far back as 1993 Kinnear [28] proposed a mutation operator that prevented an offspring of having a size that was 15% larger than its parent. Langdon [33] also proposed two mutation operators that ensured that a randomly created subtree has on average the same size as the code it replaces.

5.3.3 Anti-bloat Selection

Parsimony pressure is another well known bloat control technique originally suggested by Koza in [29]. The idea is to change the selection probability of a given individual by extracting a value based on the size of the individual from its fitness. Bigger programs are penalized more and thus tend to have fewer children. Formally, each individual x is assigned a new selection fitness $f_{sel}(x) = f(x) - c * s(x)$ where $f(x)$ is the original fitness, $s(x)$ is the size of x and c is the *parsimony coefficient*. The original fitness value is only used to recognize solutions or serve as a process stop criterion. Although evidence that presents the benefits of dynamically adjusting the parsimony coefficient during the run are presented in [61], most implementations of the method use a constant value of c.

Controlling bloat (program size) and maximizing accuracy at the same time effectively turns the evolutionary process into a multi-objective optimization problem (or a constrained optimization problem). Parsimony pressure combines the two objectives in the f_{sel} function and treats the minimization of size as a soft constraint. The intensity with which bloat is controlled depends on the value of the parsimony coefficient. If the value is too small then there is no push to minimize bloat. By contrast, if the value is too large, then solution minimization will become the primary objective and the runs will converge towards extremely small but inaccurate programs [49]. Good values for the parsimony coefficient are highly dependent on the particular problem to be solved and on the configuration parameters of the GP process.

There are also methods that try to keep the two optimization criteria (size and accuracy) separate. They typically rely on the principle of *Pareto dominance*: "Given a set of objectives, a solution is said to Pareto dominate another if the first is not inferior to the second in all objectives and additionally there is at least one objective where it is better" [46]. For example in [16], Ekart and Nemeth propose a modified tournament selection operator based on Pareto dominance. An individual is selected if it is not dominated by a set of randomly chosen individuals. In the case of failure, another individual is picked from the population until one that is non-dominated is found.

5.4 Bloat and GP Solution Size - The Result Orientated Perspective

A very important remark that needs to be made right from the start is that in this thesis we are only interested in the *result orientated perspective on bloat*. This means that *throughout this work we only focus on producing parsimonious GP solutions rather than trying to control the code bloating phenomenon in general*. This flavor is quite important as we shall immediately see.

The impact of the bloat phenomenon on solution size is studied by considering the size and accuracy dynamics of the GP population over several generations and by collaborating these with the convergence speed of the GP process (i.e. the number of generations needed to reach the solution of the GP run). We shall now see that the result orientated perspective on bloat is either based on the *training perspective* on bloat or on the *validation perspective* on bloat depending on what type of data set split strategy GP uses.

5.4.1 Bloat - The Training Set Perspective

In a GP environment that uses only a training and a test data set, the final effect of bloat on the size of the GP result or solution is very direct and easy to understand. As solution designation is based solely on model training performance, the training perspective on bloat coincides with the GP result perspective on bloat.

In this case, after a number of generations, one can observe that while the average solution size over the entire population continues to grow, there is no significant improvement in training accuracy. But even if not significant, any marginal improvement (regardless of new model size) will immediately force the GP process to accept the new model as the current solution candidate. As such, the direct impact of bloat on GP solution model size is evident: if no better solution in terms of training accuracy is found after code bloating starts, then there are very high chances to obtain a quite parsimonious solution model, else, it is very likely to end up with a large (bloated) best solution.

Trying to modify the standard behavior of the solution designation process such as to not take into account marginal accuracy improvements that are large-sized is extremely troublesome. This approach would require a general method for defining what is a "marginal improvement" and what is "large-sized" with regards to the multitude of particular cases that might arise as an evolutionary process guides a GP run.

The remaining alternative is to actually try to limit bloating until the solution of the run has been found. The solution of the run is the best performing model with respect to training accuracy. The catch is that, because of the stochastic nature of the GP process, a (marginally) more accurate training model (then the best one so far) can be found, theoretically, any time.

With respects to the above, the best bloat control technique (when using a training set for both model evolution and solution designation) is the one that can ensure overall model parsimony in the entire population for the longest time without negatively impacting the fabric of the evolutionary process guiding the search.

This idea of delaying (or preventing) bloating in the population for as long as possible (ideally forever), whilst at the same time not negatively impacting the evolutionary process, is the central (and natural) paradigm behind virtually all existing bloat control techniques.

5.4.2 Bloat - The Validation Set Perspective

In Section 3.3 we defined the overfitting phenomenon as the general major obstacle in data based modeling. We also stated that a very suitable approach for controlling overfitting in GP is the usage of a three-fold data set split into a training set, a validation set and a test set.

The impact of code bloating in GP based modeling that uses a three-fold data set split is identical to the validation set perspective on bloat as, in this case, the validation set is the one used for final solution designation. This is important as the training and validation perspectives on GP bloat can differ significantly. In order to understand this difference let us first consider the following simple scenario that can arise when using GP with a three fold data set split strategy:

- In the n^{th} generation of the GP run we come across a fairly small model, M_{small}, that turns out to be the best performing model of the entire run both on the training set and on the validation set.

- In the $(n+1)^{th}$ generation of the run we come across a large model, M_{large}, that has a much higher accuracy than M_{small} on the training data set and that is an offspring of M_{small} obtained through crossover. Let us assume that this model is "not affected by bloat": does not contain introns (either as *inviable code* or *unoptimized code*) and does not contain "*messy code* structures" [39] (e.g., $+(+(+(1,2),3),4)$). These semantical or syntactical *messy code* structures are still the subject of intense ongoing research in the field of GP in general and bloat control in particular.

- The GP run terminates immediately after the $(n+1)^{th}$ generation.

With regards to the validation accuracy of M_{large} compared to that of M_{small} we consider three distinct possibilities:

- The validation accuracy of M_{large} is significantly better than that of M_{small}.

- The validation accuracy of M_{large} is equal or marginally better than that of M_{small}.

- The validation accuracy of M_{large} is worse than that of M_{small}

From a training set perspective in all the three cases M_{large} is an improvement with regards to its parent as it has a much higher training accuracy.

From a validation point of view things are a bit more complicated. Whilst in the first case there is no doubt that M_{large} can also be viewed as an evolved improvement of its parent, in the second case, we might argue that M_{large} is affected by bloat (program growth with no significant return in fitness) and, in the third case, we have good reasons to suspect that M_{large} suffers from overfitting (low training error and high generalization error).

As one might expect after reviewing this scenario, validation based solution designation is helpful in controlling the cases where the larger individual has a validation accuracy that is worse or equal to the best found solution so far. However validation can not offer protection from large individuals that contain *marginally overfitting code* - code that improves training accuracy significantly but has insignificant benefits from a validation orientated perspective.

As a generalization, we can say that, when using three-fold data set split, a continuous generation-wide growth of program size (and possibly training accuracy) that is not matched by a *significant increase* in validation accuracy can be regarded from a GP validation perspective as bloating. In this specific case the bloating comes under the form of code that we have labeled as *marginally overfitting code*.

When also taking into consideration the very high probability of having *messy code* structures (that are effectively bloat both from a training and a validation point of view) within a GP model, the concept of code bloat from a GP validation perspective is illustrated in Figure 5.2.

From all of the above presented, we argue that the usage of validation based solution designation can help in ensuring both a more parsimonious and a more accurate GP solution by providing protection against overfitted large individuals. However, the usage of a validation set can not help with detecting bloat in the form of *marginally overfitting code* or *messy code*.

The use of a validation test has a profound implication in the way we interpret bloat control, in the sense that it reliefs us of the task of managing population growth after the point in which the GP process starts overfitting. From our point of view, trying to control bloat after this point does not make much sense as the (size of the) GP solution will not be changed anymore.

The paradigm behind bloat control, when using a three-fold data set split, is also slightly different from the one mentioned in the previous section, as an ideal bloat control strategy would only need to control the phenomenon till the best validation accuracy

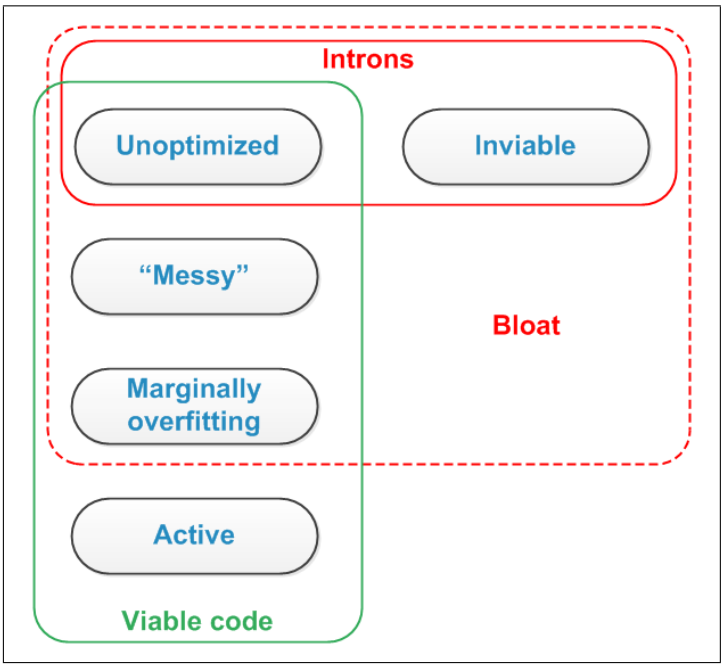


Figure 5.2: The GP validation perspective on bloat structure

is reached. Usually, in most data based modeling techniques (GP being included), the best validation solution is obtained far ahead of the best training solution.

Every HeuristicLab GP configuration used in this thesis relies on a three-fold data set split and, as such, the result orientated perspective on bloat that we take into account in our work is defined by the validation set perspective on bloat.

Chapter 6

Methodology and Initial Test Results - The Performance Baseline

6.1 Methodology

We shall now present details regarding the methodology we have adopted for performing tests and making comparisons throughout this thesis. The focus falls on describing how we compare different genetic programming (GP) configurations based on the general performance of the results they produce.

6.1.1 Input Data Preparation

For the two blast furnace modeling scenarios, A5 (14 variables) and A6 (44 variables), we have split the data set into a training set of 3200 records and a test set of 2000 records. When using GP based regression, the 3200 records data set is further split into two partitions: a training data set of 1200 records and a validation data set of 2000 records.

The EvoCompetitions 2010 symbolic regression problem is already pre-partitioned by the competition organizers into a training set (747 records) and a data set (320 records). When using GP based regression, the training dataset is also further divided into two separate sets (training 66% and validation 33%).

An important note regarding the EvoCompetitions problem is that Gabriel Kronberger from the HEAL team entered the contest proposing a model found using the GP standard regression process implemented in HeuristicLab. His solution was declared the winner. The knowledge acquired by analyzing the GP modeling process, allowed the HEAL team to successfully perform variable selection. As such, 9 variables from the original set of 57 variable were removed from the modeling scenario. All the tests we have carried out for this problem also consider the reduced 48 variable modeling scenario.

When using GP, it may also happen that not all the data available for training is used. Depending on specific parameter settings, only a predefined amount of samples (from all the available training data) are randomly chosen as the actual GP training set.

6.1.2 Comparing GP with other regression methods

Interpretability

In Chapter 3 we have presented relevant information that GP produces white-box symbolic regression models that are more interpretable then the gray-box regression models produced by ANNs and SVMs. The only prerequisite for this is the fact that the size of the GP based models should be as small as possible.

Linear regression is also a white-box modeling technique, it's only (major) disadvantage is its implicit assumption that the dependent variable linearly depends on the other variables in the data set.

Accuracy

As the three classic regression methods we are using for comparison are deterministic, the best model for each of them with regards to the modeling scenarios has been chosen using a systematic testing strategy that will be presented in detail in Section 6.2.1.

The GP process is stochastic in nature and as such multiple runs (more than 25) with the same configuration are necessary in order to determine the general behavior of that GP configuration. When we wish to evaluate the performance of a given GP configuration with regards to the other regression methods, we choose for comparison the best performing solution (according to MSE on test data) obtained among all the GP runs based on that given configuration.

Overall Performance

Based on the above, we consider that GP has an overall better performance than ANNs and SVMs if it is able to produce parsimonious solutions that are at least as accurate as the ones found using these gray-box modeling methods.

In the case of linear regression, we consider that GP performs better only if it is able to produce parsimonious models that are more accurate then those created using linear regression.

In conclusion, when assessing the overall performance of GP based regression models, interpretability (parsimony) is only the second performance criteria, whilst model accuracy is the first one. By model accuracy we always mean model accuracy with respect to future, "unseen", data and, as such, the most accurate model is the one that exhibits the smallest MSE value with regards to the test data set.

6.1.3 Comparing two different GP configurations

We previously mentioned that we need to perform multiple runs with the same configuration in order to determine the general behavior of that configuration. As such, we performed 100 GP runs for each specific GP modeling attempt, where a modeling attempt is defined by the pair (GP configuration, modeling scenario). The result of this is that each individual GP modeling attempt is defined by a data set of solutions containing 100 records. We shall refer to this data set that contains all the 100 solutions as the *the full data set*.

The fact that model accuracy is the first GP performance criteria, determines our interest in especially analyzing and improving the parsimony of the best solutions that can be generated with a given GP configuration. As such, for each modeling attempt, we also construct a *top accuracy subset* that only contains the 25 most accurate solutions in the full data set. When constructing this subset, ties broken in favor of the more parsimonious model. The top accuracy subsets are used for accuracy and size comparisons alike.

Our inter-GP comparisons are largely based on basic statistical measures related to both the full solution data sets and the top accuracy subsets. This basic statistical measures consist of two central tendency indicators (the average (μ) and the median ($\mu_{1/2}$))as well as the standard deviation σ.

When comparing among GP configuration results based on the *full data sets*, we also make use of *significance testing* to confirm our empirical based hypotheses. The significance test we use is the *Mann-Whitney-Wilcoxon* (also know as the Mann-Whitney U) *test*. The significance level we use is $\alpha = 0.05$ in the case of one-tailed tests. The choice for this particular non-parametric test was made because we do not wish to presume that our solution data is normally distributed, either according to size or accuracy.

We do not use significance testing with regards to the *top accuracy subsets* as, due to the selection process, the samples in these subsets violate the assumption of independence that all standard parametric and non-parametric significance tests assume. As the samples in the *top accuracy subsets* exhibit associative dependencies, for the purpose of this thesis we consider that it is sufficient to only present some empirical observations related to these subsets.

Whilst we shall use all the three modeling scenarios previously introduced in order to make the final full comparison between the performance of the initial GP configuration and the performance of our proposed bloat control system, during the comparison of some "intermediate" GP configurations, we shall only refer to the A6 blast furnace scenario as it is both the most complicated and the most interesting from a practical perspective. However, in each case, we have performed around 30 tests in order to confirm that the general behavior of a certain GP configuration is largely the same for all the three modeling scenarios.

In the case of the A6 scenario, apart from comparing GP configurations based on statistical facts related to the *full data sets* and *top accuracy subsets*, we shall also use two empirical solution performance quantifiers: *small* and *high accuracy*. The concrete values that define these quantifiers are presented in Section 6.2.4.

6.2 Initial Tests

We shall now provide a description of the tests we have conducted in order to produce the regression models that we consider as being the performance baseline (accuracy and size) for the modeling scenarios described in Section 4.1.

6.2.1 Weka Test Configurations

In order to find the best parameter settings for the three standard regression methods, we have adopted a strategy of systematic testing. Initially, for each regression method,

and each considered modeling scenario, our approach was to select the best performing model using *10-fold cross validation* on the training set. While this worked very well on the EvoCompetitions regression problem, in the case of the two blast furnace scenarios, we ended up with rather *overfit models*. We then decided to also try the *training set - validation set - test set* approach for the standard regression methods and this proved to be a lot more successful.

For each of the two data set partitioning methods used we conducted the following series of tests:

- In the case of the *LinearRegression classifier* we can opt for three different attribute selection methods and various values for the ridge parameter R. Multiple models were created using all the possible pair combinations between selection methods and R where $R \in [10^{-12}, 10^{-11}, \ldots, 10^{-3}, 0.01, 0.25, 0.5, 1.0, 2.0, 4.0, 8.0, \ldots 128.0]$. We ran a total of 66 tests for each modeling scenario.

- In the case of the *MultilayerPerceptron classifier* there are more parameters that can be configured: the number of hidden layers H, the learning rate L, the momentum M, the training time N, the type of learning rate (constant or decaying) D. Again, a high number of possible combinations were tested using: $H \in [1.0, 2.0, auto]$, $L \in [0.05, 0.1, 0, 15, \ldots, 0.45, 0.5]$, $M \in [0.1, 0.2, 0.3, 0.4, 0.5]$, $N \in [500, 1000, 1500]$ and $D \in [constant, decaying]$. We ran a total of 450 tests for each modeling scenario.

- The best configuration for the *SMOreg classifier* was found by varying the following parameters: the general complexity parameter C with $C \in [0.1, 0.25, 0.5, 1, 2, 4, 8, \ldots 1024]$, the RBF kernel parameter γ with $\gamma \in [0.005, 0.01, 0.015, 0.02, 0.025]$ and the ϵ parameter of the epsilon intensive loss function with $\epsilon \in [0.0005, 0.001, 0.005, 0.01, 0.05, 0.1, 0.15, 0.20, 0.25]$. We ran a total of 700 tests for each modeling scenario.

6.2.2 HeuristicLab Test Configuration

In this section we shall present what we shall refer to as the *initial HeuristicLab GP configuration* - InitialHL-GP in short. It is the GP process configuration that is able to produce the best models for all the considered modeling scenario.

The considered GP process uses all the enhancements presented in Section 4.2.1. The chosen male-selection operator was proportional selection, whilst the chosen female-

selection operator was random selection. The offspring selection success ratio was 1 meaning that all members of the next generation had to outperform their respective parents. Constants were initialized and modified uniformly (between -20 and 20), whilst variables were initialized and modified according to a normal distribution $N(0, 1)$. The other, more common, GP algorithm parameters that we have used are listed in Table 6.1.

Process parameter	**Value**
Function library	$+$, $-$, $*$, %, *power*, *sqrt*
Population size	1000
Mutation rate	15%
Max tree height	15
Max evaluated solutions	5000000
Max generations	1000

Table 6.1: Common GP parameter settings used in all HeuristicLab GP configurations presented in this thesis

The stopping criterion that was reached first in all the runs stated that the GP process should terminate if there had been no improvement with regards to the validation accuracy for more than 20 generations (*BestValidationSolutionAge*=20).

The only bloat control method the initial version of HeuristicLab GP used was a static depth limit that prevented trees from having a depth larger than 15. The actual limit value was chosen empirically after performing numerous tests on several symbolic regression scenarios (which also included our three current modeling scenarios) and observing that, in all of them, most of the good and very good solutions had a depth smaller than 10.

6.2.3 Performance Baseline

Accuracy

In order to compare the generalization error of different regression, models we have chosen to use an additional indicator, other than the mean squared error (MSE). As such, for each obtained model we also calculate *Spearman's rank correlation coefficient* (ρ). This coefficient is calculated on the test set in order to provide a more accurate view of the prediction capability of a given model.

If, when considering regression problems, the smaller the MSE the better the model, in the case of Spearman's rank correlation coefficient, the better the model, the closer the value of ρ is to 1.00.

The best results obtained using the three standard regression methods discussed earlier and the InitialHL-GP process are summarized in Table 6.2 (the best result is chosen according to MSE and the corresponding ρ is displayed). While GP based regression performs quite well on the blast furnace scenarios, its accuracy on the EvoCompetitions problem is considerably lower.

	LinReg		**ANN**		**SVM**		**Initial GP**	
Scenario	MSE	ρ	MSE	ρ	MSE	ρ	MSE	ρ
Blast furnace A5	1.159	0.81	1.498	0.75	**0.867**	**0.87**	0.869	**0.87**
Blast furnace A6	1.175	0.81	1.160	0.80	0.912	0.84	**0.865**	**0.87**
EvoComp problem	0.022	0.90	0.018	0.92	**0.012**	**0.94**	0.037	0.79

Table 6.2: Accuracy information regarding four regression methods (i.e. MSE on test data set)

An important observation is the fact that GP achieves the best results on the A6 scenario, and that these results are pretty similar to those obtained on the reduced A5 scenario, both in terms of accuracy and semantics (i.e. the variable impact analysis revealed that they both largely contained the same set of dependent variables). However, as we will later see, these initial best performing GP models also contain a rather large level of redundancy.

	Solution MSE on test set					
	Full data set			**Acc. subset**		
	μ	σ	$\mu_{1/2}$	μ	σ	$\mu_{1/2}$
Blast furnace A5	1.103	0.153	1.065	0.960	0.036	0.970
Blast furnace A6	1.448	0.584	1.280	0.929	0.037	0.930
EvoComp problem	0.162	0.071	0.050	0.043	0.002	0.045

Table 6.3: Statistical information regarding the solution accuracy of InitialHL-GP

We continue with the overview of our accuracy baseline by presenting in Table 6.3 information regarding accuracy for the *full data sets* as well as the *top accuracy subsets* of the initial GP configuration.

All of the above presented observations support the conclusion that the InitialHL-GP process is quite powerful and it is able to produce results of comparable, if not even better, accuracy than the considered standard regression methods.

Solution size

In order to have an overview of the general behavior of the initial HeuristicLab GP process with regards to solution size, in Table 6.4 we present solution size related statistical information regarding the full *full data sets* and the *top accuracy subsets*

	Solution size					
	Full data set			**Acc. subset**		
	μ	σ	$\mu_{1/2}$	μ	σ	$\mu_{1/2}$
Blast furnace A5	46.92	20.58	43.00	44.84	19.64	44.00
Blast furnace A6	52.60	28.11	47.00	53.36	29.94	52.00
EvoComp problem	29.92	17.57	26.00	35.16	21.94	30.00

Table 6.4: Statistical information regarding the solution size of InitialHL-GP

Overall

The main goal of this thesis is to reduce the size of the solution regression models produced with the enhanced GP process implemented in HeuristicLab while at the same time ensuring no deterioration of solution accuracy. As such, the previously solution size statistics together with all the results regarding initial GP accuracy, are considered as the combined *accuracy-size baseline* for any GP enhancement aimed at improving GP solution parsimony within the HeuristicLab framework.

6.2.4 Bloat - A Concrete Example

After having presented the general behavior of the InitialHL-GP process in terms of size and accuracy, we can now present why we consider that, from a result orientated perspective (see Section 5.4), a considerable part of the solutions are affected by bloat, and that, the GP process can be further enhanced in order to produce more parsimonious solutions.

For this example, we shall only consider the results obtained for the A6 blast furnace scenario, the scenario where GP performed better than any of the other considered regression methods. For the A6 modeling scenario we consider the following empirical solution performance quantifiers: a *small* solution has a size ≤ 35 and a *high accuracy* solution has a test MSE ≤ 0.90. The accuracy threshold was chosen by taking into account the general performance of linear regression, ANNs and SVMs based regression for this scenario. The size threshold has been chosen based on the observation that 25% of the solutions generated by the InitialHL-GP process have a size that is smaller or equal to 35.

Having set these thresholds, let as look at the *full data set* of solutions generated with the InitialHL-GP process. In the entire set we have:

- 19 solutions that that have a *small* size;
- 6 *high accuracy* solutions that have a mean size of 61.83;
- one small-sized (size = 20) highly-accurate solution (test MSE = 0.88);
- the best accuracy (test MSE 0.86) is achieved by a solution of size 54 that contains 12 dependent variables.

Interestingly, among the 6 top performing solutions we have two more models that achieved a test MSE of 0.88, one with a size of 77 and the other with a size of 79. So, while these two solutions exhibit the same generalization error as the single high accuracy small-sized solution, they are almost 4 times as large.

Let us take a closer look at the dynamics of the two GP runs that produced the small solution (Figure 6.1) and the large-sized solution (Figure 6.2). We can see that in the GP run that produced the large sized solution, the average model size (*AvgModSize*) over the entire population as well as the minimum model size (*MinModSize*) tend to increase dramatically after only a few generations. By contrast, in the run that produced the parsimonious solution, these indicators increase steadily and in a much more dampened manner. The convergence speed (*SolGen*) towards the solution (i.e. the model with the best validation accuracy) is largely similar in the two GP runs (and quite fast when compared with a standard GP approach): the small solution is found after 15 generations and the large one after 17 generations.

An obvious explanation for the difference in sizes is the fact that the size of the solution is strongly correlated with the average model size and the smallest model size of the

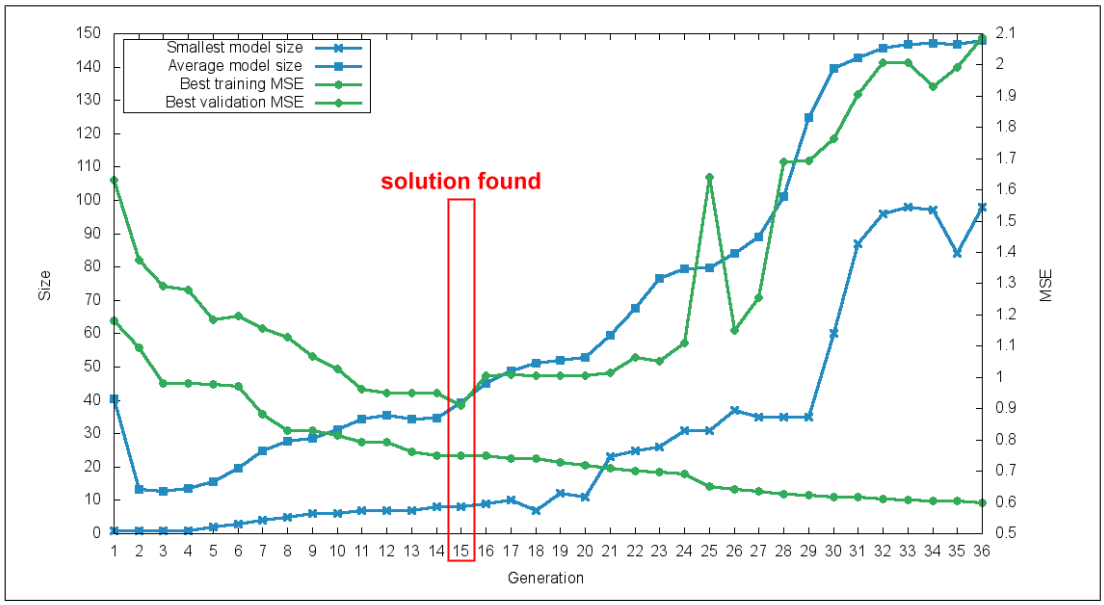


Figure 6.1: InitialHL-GP run that produced a parsimonious model with a test MSE of 0.88

generation when the solution was found. For the two considered GP runs, this data is summarized in Table 6.5.

	SolGen	*MinModSize*	*AvgModSize*
Small solution	15	8	39.317
Large solution	17	24	102.73

Table 6.5: Population dynamics indicators for the two considered GP runs

Averaging related statistics for the *SolGen*, *MinModSize*, *AvgModSize* indicators based on the *top accuracy subset* of InitialHL-GP for the A6 modeling scenario are presented in Table 6.6. The table also contains an indicator for the average solution size (*AvgSolSize*). On average, the best validation solution was found at the 16^{th} generation. The main reason for this fast convergence rate is the offspring selection strategy used within HeuristicLab GP. Whenever comparing with a standard GP implementation that does not use this enhancement, one should instead look at the number of evaluated individuals till the solution was found. According to this last metric, on average, the convergence speeds of the two types of GP processes appear to be rather similar.

As one can see, the mean average model size (*AvgModSize*) in the generation when the solution was reached is quite high: 68.52. Allthough the mean minimum model size is considerably smaller, with a value of 21, the average size of the solution (53.36) tends to be closer to the mean average model size than to the mean minimum model size.

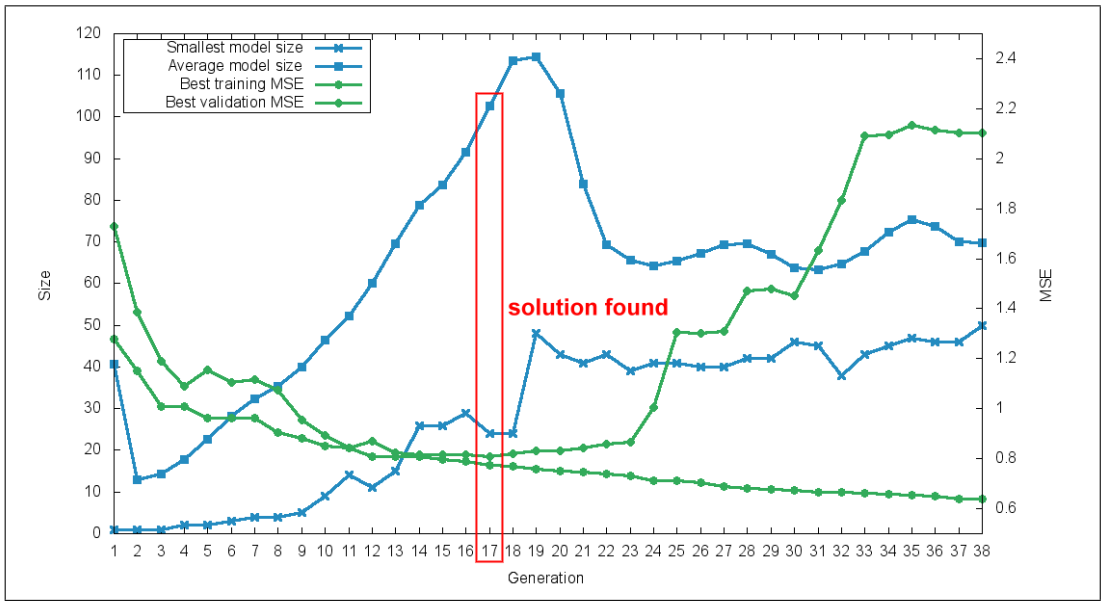


Figure 6.2: InitialHL-GP run that produced a large-sized model with a test MSE of 0.88

Indicator	μ	σ
SolGen	16.24	3.91
MinModSize	21.00	13.25
AvgModSize	68.50	24.20
AvgSolSize	53.36	29.24

Table 6.6: InitialHL-GP population dynamics indicators for the *top accuracy subset* of the A6 modeling scenario

The HeuristicLab GP implementation we tested with for this thesis only stores size information related to the best performing validation model. If we had size information regarding the best performing training model as well, it would have been extremely interesting to analyze the two example GP runs from a purely training set perspective as well. This is because, for our two considered runs, in the generation where the model with the best training accuracy was found (i.e. the last generation of each run) the values of the *MinModSize* and *AvgModSize* indicators are much smaller for the run we have labeled from a result orientated perspective as being affected by overfitting.

Chapter 7

Reducing Solution Size in HeuristicLab GP

7.1 The Impact of the Offspring Selection Enhancement on Solution Size

Let us first present some assumptions on how the offspring selection strategy (see Section 4.2.1) in its strictest form (*SuccRatio* parameter set at 1.00) might influence the final solution size of the HeuristicLab GP process (i.e. its influence on bloat from the "validation perspective" - Figure 5.2).

On the one side, the offspring selection enhancement used by HeuristicLab GP can be regarded as an improvement on the non-destructive crossover methods proposed by Soule [50] [51] and as such, based on Luke's experiments in [39], we expect that using this enhancement will lead to a dramatical decrease of the amount of *inviable code* in the population. Furthermore, the combination of offspring selection and validation based solution designation also helps in combating large individuals that exhibit an overfitting behavior.

On the other side, taking into account the behavior of unoptimized code (Section 5.2.1) and the structure of the offspring selection scheme (Section 4.2.1) we also have good reasons to expect that when using this enhancement, the probability that *unoptimized code* will be propagated through the generations actually increases. A hint towards this possible behavior is also presented in [39].

With regards to the "*marginally overfitting code*" and "*messy code* structures" we can make no assumptions on what the effect(s) of offspring selection on this type of code bloat are.

Whilst based on the above assumptions (some of which not fully tested) we cannot draw a decisive conclusion on what the overall effect of strict offspring selection on final GP solution size might be, there is strong evidence ([57] [58] [59] [32]) that this enhancement is extremely beneficial in terms of accuracy. Therefore, in this thesis, our focus will fall on analyzing the impact of offspring selection on GP solution size.

7.1.1 Offspring Selection Experiments

In order to get a better overview on the effect of offspring selection on GP solution size and accuracy in the case of our modeling scenarios, we shall first compare the performance of the InitialHL-GP process with that of a *standard GP configuration* - StandardHL-GP. that does not use this enhancement.

Afterwards, we shall also make use of the gender specific selection enhancement (also described in Section 4.2.1) in order to test if varying the internal selection pressure while using strict offspring selection has any impact on bloat and resulting model accuracy.

7.1.2 Comparison with Standard GP

The standard GP configuration we have used for testing was a HeuristicLab implementation of Algorithm 2.1 which also benefited from three GP enhancements presented in Section : *linear scaled error measure*, *PTC2 based initialization* and *point mutation parametrization*.

Again, constants were initialized and modified uniformly (between -20 and 20) whilst variables were initialized and modified according to a normal distribution $N(0, 1)$. The parents were selected using proportional selection. The *BestValidationSolutionAge* parameter was set at 100 and it was the first reached stopping criterion in every run. The more basic GP parameters settings we have used are the ones described in Table 6.1.

In Table 7.1 we present statistics regarding solution accuracy and solution size when using the standard GP configuration.

When comparing the results in terms of accuracy, based only on the central tendencies and standard deviation, we have strong reasons to presume that StandardHL-GP performs far worse than InitialHL-GP (Table 6.3) on the *full solution data sets* as it has higher mean and median values and a larger standard deviation. These observations are also pertinent to the *top accuracy subsets*.

	Full data set			Acc. subset		
Perf. criterium	μ	σ	$\mu_{1/2}$	μ	σ	$\mu_{1/2}$
Solution test MSE	2.178	1.046	1.821	1.177	0.177	1.232
Solution size	45.44	33.50	39.00	44.84	24.55	43.00

Table 7.1: StandardHL-GP configuration performance - A6 scenario

In order to provide a more detailed overview of the accuracy of all the results in the solution data sets, we now introduce a type of plot (Figure 7.1) that will be used extensively throughout this chapter for result dissemination: *the comparative performance-parameter kernel density estimation plot.* In this case, the performance parameter is solution accuracy (i.e. test MSE value) but later on we shall also present solution size information in the same manner.

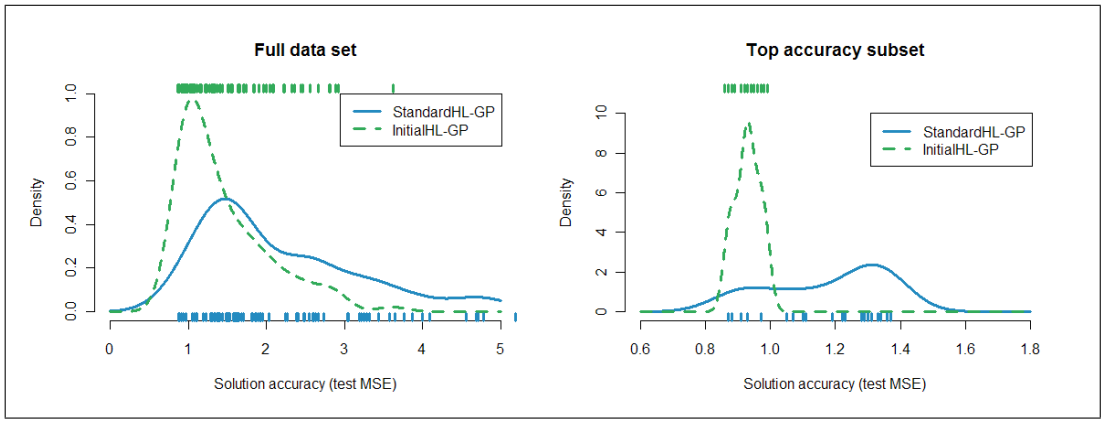


Figure 7.1: Comparative kernel density estimation of GP solution accuracy for StandardHL-GP and InitialHL-GP - A6 scenario

The observed difference in accuracy between the two GP configurations is also confirmed by the kernel density estimation plot. Unsurprisingly, this difference is also statistically significant as the *Mann-Whitney-Wilcoxon test* yielded a one-tailed *p-value* smaller than 0.001.

Finally, we must mention that whilst the general behavior of StandardHL-GP is worse in terms of accuracy than that of InitialHL-GP, the former was still able to produce two 2 high accuracy models of which one was also small-sized (according to the empirical solution performance quantifiers defined in Section 6.2.4);

When comparing the results in terms of solution size, the standard GP configuration has smaller mean and median values with regards to both the *full solution data set* and the *top accuracy subset.*

The comparative kernel density estimation with regards to solution size is presented in Figure 7.2. An interesting observation is the fact that this plot appears to confirm our observations regarding the difference in central tendencies for the *full data set* whilst in the case of the *top accuracy subset* there seems to be no real difference.

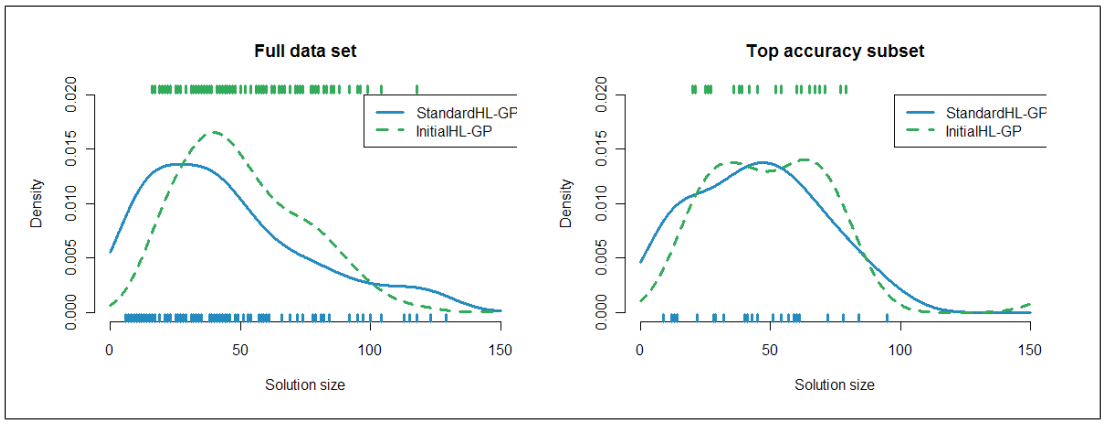


Figure 7.2: Comparative kernel density estimation of GP solution size for StandardHL-GP and InitialHL-GP - A6 scenario

The hypothesis that the observed difference in central tendencies is statistically significant with regards to the *full solution data set* is confirmed by the one-tailed *Mann-Whitney-Wilcoxon test* (one-tailed *p-value*=0.005).

7.1.3 The effect of parent selection pressure

Within the offspring selection strategy, the parent selection pressure can be:

- *high* - when both parents are chosen using proportional selection
- *medium* - when one parent is chosen using proportional selection and the other using random selection
- *low* - when both parents are chosen using random selection

We have conducted a series of tests to see if this factor has any influence on final solution performance when using offspring selection. As InitialHL-GP is actually an example of using medium parent selection pressure, starting from it we have easily obtained configurations implementing high and low parent selection pressures by simply modifying one of the selection operators. All the other configuration parameters were left unchanged.

The basic statistics regarding solution accuracy are listed in Table 7.2 and the statistics regarding solution size are listed in Table 7.3. The comparative accuracy kernel density estimation plot is presented in Figure 7.3 whilst the comparative size kernel density estimation plot is presented in Figure 7.4.

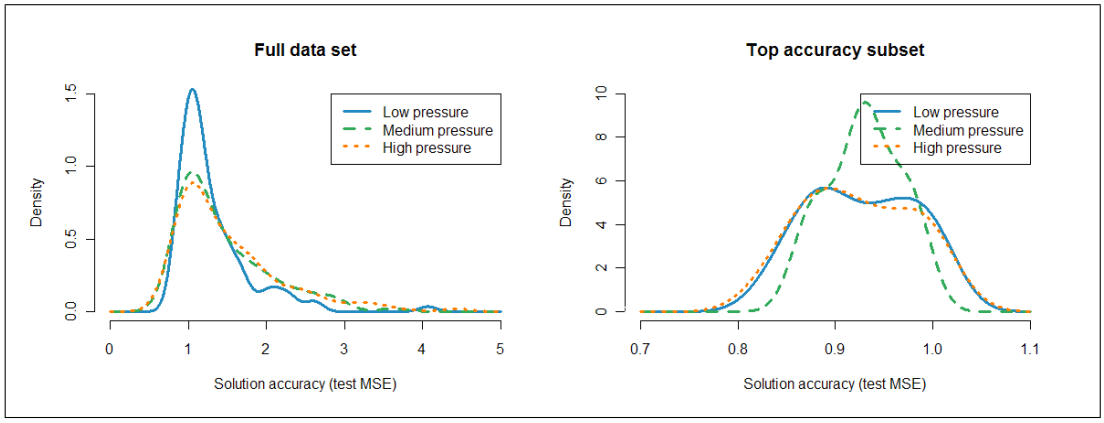


Figure 7.3: Comparative kernel density estimation of solution accuracy for three GP configurations with different parent selection pressures - A6 scenario

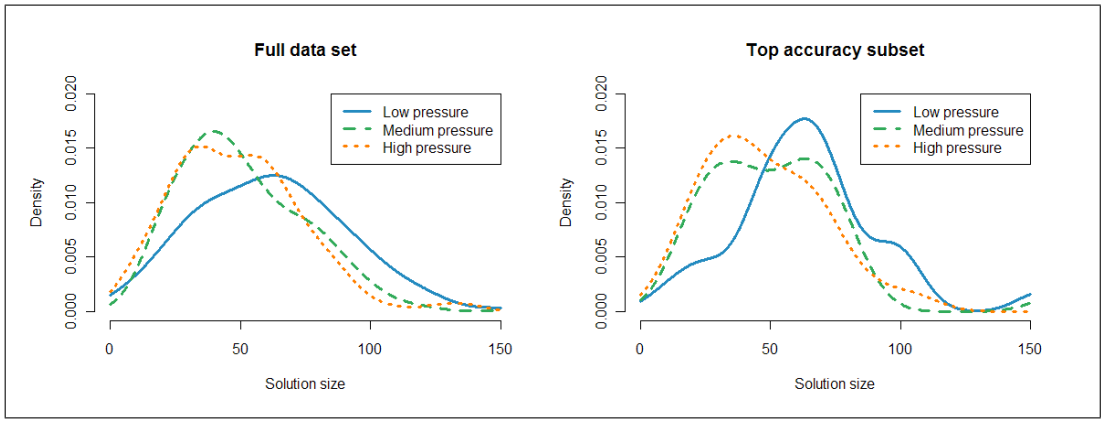


Figure 7.4: Comparative kernel density estimation of solution size for three GP configurations with different parent selection selection pressures - A6 scenario

For the presented statistics regarding the accuracy and size performance of the three chosen configurations, we have tested for significance with the *Mann-Whitney-Wilcoxon test* all the possible hypothesis (based on the *full solution data set*).

With regards to accuracy, the fact that the low pressure configuration has lower central tendencies than the high pressure configuration is marginally significant (one-tailed (p-*value*=0.0042)).

	Solution MSE on test set					
	Full data set			**Acc. subset**		
Configuration	μ	σ	$\mu_{1/2}$	μ	σ	$\mu_{1/2}$
Low pressure	1.308	0.490	1.14	0.926	0.054	0.93
Medium pressure	1.448	0.584	1.280	0.929	0.037	0.930
High pressure	1.5196	0.6927	1.280	0.923	0.056	0.920

Table 7.2: Solution accuracy statistics for three GP configurations with various parent selection pressures - A6 scenario

	Solution size					
	Full data set			**Acc. subset**		
Configuration	μ	σ	$\mu_{1/2}$	μ	σ	$\mu_{1/2}$
Low pressure	62.65	31.36	62.5	65.00	29.29	64
Medium pressure	52.60	28.11	47.00	53.36	29.94	52.00
High pressure	48.96	24.68	47.00	47.60	22.50	44.00

Table 7.3: Statistical information regarding the solution size of three GP configurations with various parent selection pressures - A6 scenario

With regards to the solution size distribution two observations are statistically significant:

1. the low pressure configuration has higher central tendencies than the medium pressure configuration - (one-tailed $p\text{-}value$=0.004);

2. the low pressure configuration has higher central tendencies than the high pressure configuration - (one-tailed $p\text{-}value$<0.001);

By analyzing the values of the presented statistics as well as the plots of the estimated densities with regards to the *top accuracy subsets*, the central tendency trends presented above don't seem to be that relevant. For example in Figure 7.3 the accuracy distributions of the low pressure and high pressure configurations are virtually identical with respect to the *top accuracy subsets*, whilst on the *full data sets*, the difference between these two distributions is marginally significant.

With regards to the empirical solution performance quantifiers described in Section 6.2.4, the high pressure configuration was able to produce 12 high accuracy models of which only one is small-sized, whilst the low pressure configuration was able to produce 10 high accuracy models of which two are also small-sized.

7.1.4 Discussion

From the series of tests we have described regarding offspring selection and based on the presented observations and the hypothesis that have been labeled as statistically significant, we can draw the following conclusions regarding the general behavior of this strategy:

- Offspring Selection helps GP perform significantly better in terms of accuracy while also determining a significant overall increase in solution size.

- Within the offspring selection strategy, the real driving force behind the evolutionary process is the offspring selection step. As such, the GP configuration that used a low parent selection pressure performed just as good in terms of accuracy (if not better) as the configurations that used medium and high parent selection pressures.

- Applying increased parent selection pressure has no positive influence on overall solution accuracy.

- Applying low parent selection pressure seems to drive a significant increase in overall solution size

The last two conclusions regarding the behaviour of the offspring selection strategy are very important as they motivate our decision to keep the medium parent selection pressure configuration (i.e., the InitialHL-GP configuration) as the base configuration for future tests aimed at improving GP solution parsimony.

The above conclusions are based on the performance of the various configurations with regards to the *full solution data sets*. During the presentation of the results we have also hinted that empirical observations on the *top accuracy subsets* don't appear to confirm all these preliminary conclusions.

Nevertheless, the fact that the enhanced HeuristicLab GP process (based on offspring selection) is far superior to a standard GP process in terms of general solution quality is undisputed. This is extremely important for the later stages of the modeling process (e.g., variable impact analysis) when an increased number of high quality solutions provides experts with a far better overall view of the problem at hand.

In light of the above observations, our main challenge has become a little more well defined: we must further enhance the InitialHL-GP process such as to reduce the apparent overall increase in solution size that offspring selection promotes, while at the

same time, ensuring that the significantly better accuracy performance of InitialHL-GP is not affected.

7.2 Possible Bloat Control Strategies for HeuristicLab

Based on the general reasoning presented in 5.4 and especially in 5.4.2, our strategy for ensuring solution parsimony in HeuristicLab GP (i.e. reducing bloat from a result orientated perspective) is to try to delay the bloating phenomenon or at least limit its magnitude without affecting the behavior of the evolutionary process.

For this reason we have chosen to investigate what popular bloat control techniques can be integrated in HeuristicLab and how they would perform. The main challange came from trying to smoothly integrate these bloat control techniques with the offspring selection strategy.

For instance, the anti-bloat selection methods (Section 5.3.3) are fairly hard to combine with the offspring selection enhancement because their most effective implementations are based on dynamically adjusting the control parameters: the parsimony coefficient (for the parsimony pressure method) or the weighting of the objectives (for the Pareto dominance method). Each such adjustment is based on a certain set of statistics regarding a given generation and, as such, coefficient and weight modifications are performed at most once every generation. But, as explained in Section 6.2.4, the offspring selection enhancement makes the GP process converge very fast (in terms of generations) to the best validation solution (see Table 6.6). This means that there are far fewer opportunities to dynamically adjust the control parameters, and the impact of this on the original bloat combating performance is hard to estimate. Furthermore, a modification of the offspring selection enhancement such as to somehow allow for intra-generation control parameter adjustment is a fairly complicated task. Based on this, we opted for keeping anti-bloat selection methods as a measure of last resort.

In the case of anti-bloat genetic operators (Section 5.3.2) we are not aware of any research that fully documents the influence of this method on the behavior of the evolutionary process (especially for tree-based GP algorithm used for solving complex modeling scenarios). Our educated guess is that in this case, the combination of size and/or depth limitations imposed to genetic operators and offspring selection has a high chance of leading to a fast drop in genetic diversity (i.e. premature convergence). This is due to the fact that we would have a combination between a method that restricts the number of offspring that are produced and a method that includes in the next generation only offspring that are more accurate than their respective parents.

As the initial HeuristicLab GP configuration already implements basic depth limitation (Section 6.2.2), our initial idea was to try to develop our bloat control strategy around this bloat control method. The decision was also motivated by the fact that there are no obstacles in trying to integrate this bloat control strategy with the offspring selection enhancement. Size limitations were not considered as there is strong evidence that using them can actually favor bloating in the early stages of the run [13].

After a set of initial experiments with static depth limitations we also decided to test the opportunity of using a dynamic depth limitation strategy.

Largely inspired by [1] and [57], and taking into consideration all the particularities of the GP process we are trying to enhance, we also decided to implement and test a bloat control strategy based on syntax-tree pruning.

7.2.1 Static Depth Limits

As mentioned in 5.3.1, the classical implementation of this strategy ([29]) eventually leads to a population that fills up with "bushy programs" that nearly infringe the limit. The well known method for combating this behavior (i.e. the avoidance of parent replication when an offspring violates the limit) is also implemented in the HeuristicLab GP process.

Even so, with static depth limiting, in the very late stages of the evolutionary process, the offspring selection strategy seems to promote the creation of "bushy" offspring as they seem to be the easiest means of obtaining marginal training accuracy improvements over already large parents.

Another major limitation of this basic bloat control method is the fact that it does nothing to control bloat until the limit is reached.

As expected, by considering the above, our attempts to control solution bloating by manipulating the value of the static depth limit (initially set at 15) were not very successful. Furthermore, the classic argument against static limitations (the value of the limit must be set accordingly to the specific problem at hand) makes this method an unlikely candidate for the robust bloat control system we set out to design, especially since estimating the proper limit value is harder when also using a validation test.

Nevertheless, the usage of a static depth limit with a high enough value (15 in our case) does seem to help in improving the speed of the GP process as the method prevents the

creation and subsequent manipulation of extremely large individuals that are highly bloated and/or extremely overfit.

7.2.2 Dynamic Depth Limits

In [48], Silva and Costa present a simple yet effective solution for overcoming most of the shortcomings of static depth limitation. Their idea is to dynamically adjust the value of the depth limit during the run.

Description

Compared to the original method, we have made a series of modifications in order to integrate dynamic depth limiting (DDL) into the HeuristicLab GP process. As such, the offspring selection criterion is slightly modified in order to incorporate a module that takes into consideration the dynamic depth limit (see Algorithm 7.1). In our implementation of the concept, the dynamic limit is initially set to a low value ($InitialDepthLimit$), usually 20-30% higher than that of the maxium depth in the initial population. Inside the offspring is automatically accepted in the next generation if it satisfies the accuracy constraint and at the same does not infringe the depth limit. If an offspring infringes the depth limit but is the best individual found so far (with regards to training accuracy) then it is accepted in the next generation if the increase in size is matched by the increase in accuracy. In this last case, the DDL is raised to match the depth of the new best-of-the-run individual. After several test runs we decided to also allow the limit to decrease. If during the run the best found individual at a given time has a depth that is significantly lower than the current DDL, the depth limit will be lowered, but it can not fall down to a value that is lower than its initialization value. (i.e. we implemented what Silva and Costa called the *heavy dynamic depth limit*).

The condition on Line 11 states that the DDL should be raised if each extra depth level is matched by an increase in training accuracy of at least C_{raise}%. Analogously, the condition on Line 3 states that the DDL should be lowerd if each decreased depth level is matched by an increase in trianing accuracy of at least C_{lower}%. As an empirical rule, tests have shown that the relation $C_{lower} = 2 * C_{raise}$ enables the DDL to have a stable behaviour throughout the run for all the considered test scenarios. Furthermore, we discovered that $C_{raise} = 0.015$ is also a stable setting for all test scenarios. Some initial tests revealed that the unusual stability of these settings is highly influenced by the usage of the linear scaled error measure and offspring selection enhancements.

Algorithm 7.1 The dynamic depth limiting module (DDL module)

```
1: AcceptStatus = true
2: if OffDepth ≤ DepthLimit then
3:    if (BestMSE / OffspringMSE - 1) ≥ (DLimit - OffspringDepth) * C_lower then
4:       if OffspringDepth > InitialDepthLimit then
5:          DLimit = OffspringDepth
6:       else
7:          DLimit = InitialDepthLimit
8:       end if
9:    end if
10: else
11:    if (BestMSE / OffMSE - 1) ≥ (OffspringDepth - DLimit) * C_raise then
12:       DLimit = OffspringDepth
13:    else
14:       AcceptStatus = false
15:    end if
16: end if
17: return AcceptStatus
```

Performance

In order to test the performance of this bloat control method, we have created a GP configuration that is identical in every way to the InitialHL-GP configuration with the exception that it also uses dynamic depth limiting - DDL-HL-GP in short. The static depth limit was kept at a level of 15.

The basic statistics regarding DDL-HL-GP solution performance are listed in Table 7.4. The comparative accuracy kernel density estimation plot is presented in Figure 7.5 whilst the comparative size kernel density estimation plot is presented in Figure 7.6.

	Full data set			**Acc. subset**		
Perf. criterium	μ	σ	$\mu_{1/2}$	μ	σ	$\mu_{1/2}$
Solution test MSE	1.514	0.643	1.345	0.919	0.049	0.940
Solution size	32.92	16.30	30.50	34.16	14.32	34.00

Table 7.4: DDL-HL-GP configuration performance - A6 scenario

With regards to solution accuracy, the central tendencies of the InitialHL-GP process (Table 6.3) are slightly better (smaller MSE) then the ones of DDL-HL-GP process both for the *full data sets* and for the *top accuracy subsets*. However, the hypothesis that this difference is statistically significant with regards to the *full solution data set* is rejected by the *Mann-Whitney-Wilcoxon test* (one-tailed *p-value*= 0.298).

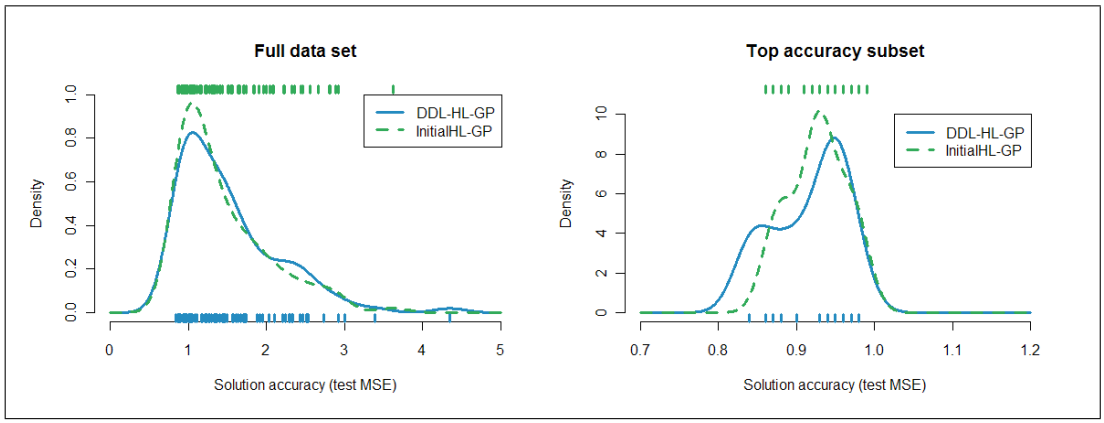


Figure 7.5: Comparative kernel density estimation of solution accuracy for DDL-HL-GP and InitialHL-GP - A6 scenario

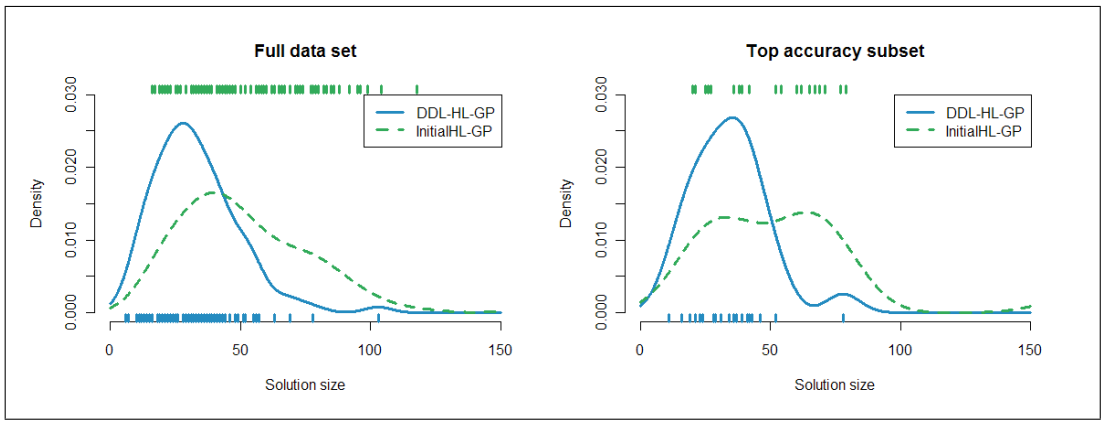


Figure 7.6: Comparative kernel density estimation of solution size for DDL-HL-GP and InitialHL-GP - A6 scenario

With regards to solution size, the central tendencies of the DDL-HL-GP process are a lot smaller then the ones of the InitialHL-GP process (Table 6.4) both for the *full data sets* and for the *top accuracy subsets*. The difference is also statistically significant with regards to the *full solution data set* (one-tailed $p\text{-}value < 0.0001$).

With regards to the empirical solution performance quantifiers described in Section 6.2.4, DDL-HL-GP was able to produce 10 high accuracy solutions of which 4 were also small-sized.

7.2.3 Pruning

Most bloat combating strategies adopt a preventive/passive stance in trying to control the phenomenon. This means that they do not directly try to modify bloated individuals

but they try to impose some restrictions on various parts of the evolutionary process such as to prevent the spreading (or conservation of bloat) in future generations. As we shall now see, there also exist more active ways of trying to combat bloat.

In machine learning, the term *pruning* is used to describe the act of systematically removing parts of decision trees. When used, regression or classification accuracy is decreased intentionally and traded for simplicity and better generalization. The concept of removing branches has also been transfered in (tree-based) GP. Several pruning methods for GP have been proposed over the years (e.g., [18] [14] [40]). In the context of GP, pruning is used for avoiding overfitting and as a bloat control method.

An exhaustive pruning technique is described in [57]. The method is based on the idea of systematically trying all pruning possibilities such as to maximize model complexity deterioration (i.e. reduction in size) while at the same time minimizing the deterioration coefficient (i.e. loss in accuracy). While the method has the advantage of ensuring the best possible pruning decision, it also has the main downside of being extremely computationally intensive.

Description

For this thesis, in order to evaluate the opportunity of using a pruning stage, we implemented a fairly simple pruning idea called iterated tournament pruning (ITP). The strategy is described in Algorithm 7.2. It is based on a couple of consecutive pruning tournaments. In each tournament we create several pruning candidates of the syntax-tree model we wish to prune in that tournament (the pruning base model). A pruning candidate for a model is created through a very simple process (Line 6) that randomly selects a subtree from the given model an replaces it with the mean value obtained by evaluating that subtree over all the records in the training set. The size of the excised subtree is limited to at most $MaxPruning\%$ with regards to the prunning base model. At the end of each tournament, we select as the prunning base for the next tournament, the prunned model with the highest accuracy (minimum MSE).

As we do not apply pruning on all individuals in the population or at every generation, the effectiveness of the pruning method is also determined by the *pruning criterion* (i.e. when and what models we choose to prune).

When considering all the three modeling scenarios, initial tests revealed that the method is not very stable with regards to all the parameters it supports. But, as a rule of the thumb, after initial testing, we can say that:

Algorithm 7.2 The iterated tournament pruning module (ITP module)

```
1: BestMSE = ∞
2: PrunSolution = Φ
3: PrunBase = OriginalModel
4: for i = 0 to IterationsCount do
5:   for j = 0 to TournamentSize do
6:     PrunCandidate = StochasticPrune(PrunBase, MaxPruning)
7:     PrunMSE = ComputeMSE(PrunCandidate)
8:     if PrunMSE < BestMSE then
9:       PrunSolution = PrunCandidate
10:      BestPrunMSE = PrunMSE
11:    end if
12:  end for
13:  PrunBase = PrunSolution
14: end for
15: return PrunSolution
```

- The $MaxPruning$ and $IterationsCount$ parameters should be set such as to generally avoid the possibility of completely reducing a model to a single node.

- The $TournamentSize$ should have a value > 50

- Better solution accuracy is obtained if pruning is not applied in the initial generations (e.g. the first 5 generations when using offspring selection) and then applied every two or three generations.

- Better solution accuracy is obtained if pruning is not performed on all individuals in the population. When ordering the population according to accuracy, pruning yielded best results when applied only to the individuals that were not among the best 10-15% (i.e. $PruningStartPercentile = 10 - 15\%$) nor among the worst 20-40% (i.e. $PruningEndPercentile = 60 - 80\%$).

- Because in HeuristicLab GP point mutation is the only method for the parametrization of constants and each pruning operation produces at least $IterationsCount$ new constants, an increased mutation rate of 25-50% helps in improving solution accuracy. This is also possible because, by design, the offspring selection strategy automatically rejects mutations that are not beneficial.

Performance

In order to test the pruning strategy we modified the InitialHL-GP configuration such as to also include the pruning module. Using the new configuration (ITP-HL-GP) we

made a series of tests in order to discover what parameter settings are more suitable for the A6 blast furnace scenario. The best found parameter configuration is presented in Table 7.5.

The basic statistics regarding ITP-HL-GP solution performance are listed in Table 7.6. The comparative accuracy kernel density estimation plot is presented in Figure 7.7 whilst the comparative size kernel density estimation plot is presented in Figure 7.8.

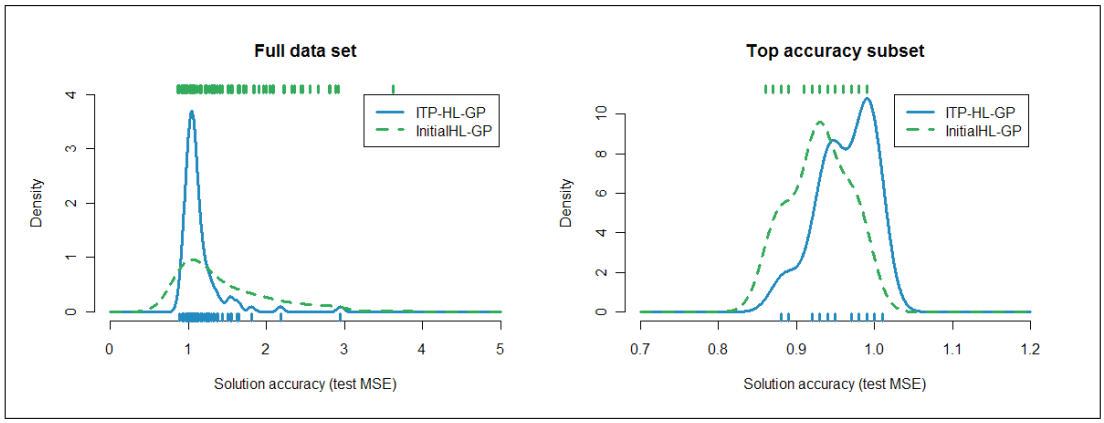


Figure 7.7: Comparative kernel density estimation of solution accuracy for ITP-HL-GP and InitialHL-GP - A6 scenario

With regards to solution accuracy, the central tendencies of the ITP-HL-GP process are better (smaller MSE) then the ones of InitialHL-GP process (Table 6.3) for the *full data sets*. This difference is also statistically significant one-tailed ($p\text{-}value{=}{<}0.0013$), two-tailed ($p\text{-}value{=}{<}0.0027$). Interestingly, with regards to the *top accuracy subsets*, the situation is reversed as the central tendencies of the ITP-HL-GP process are slightly better than the ones of the Initial-GP process.

Pruning parameter	**Value**
$IterationsCount$	3
$TournamentSize$	100
$MaxPruning$	30%
$PruningStartPercentile$	10%
$PruningEndPercentile$	75%
$MutationRate$	50%

Table 7.5: ITP-HL-GP best found parameter settings - A6 scenario

	Full data set			Acc. subset		
Perf. criterium	μ	σ	$\mu_{1/2}$	μ	σ	$\mu_{1/2}$
Solution test MSE	1.228	0.925	1.065	0.962	0.035	0.970
Solution size	39.34	18.65	35.00	42.16	18.43	38.00

Table 7.6: ITP-HL-GP configuration performance - A6 scenario

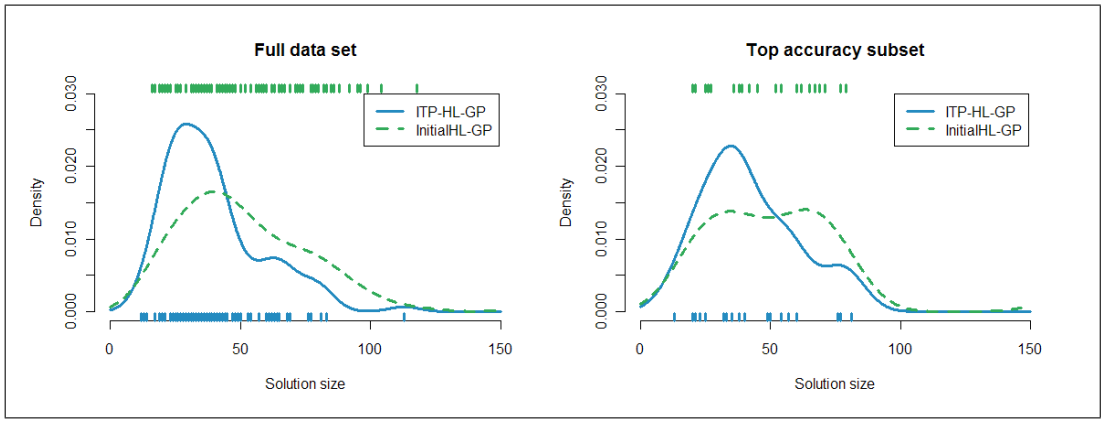


Figure 7.8: Comparative kernel density estimation of solution size for ITP-HL-GP and InitialHL-GP - A6 scenario

With regards to solution size, the central tendencies of the ITP-HL-GP process are considerably smaller then the ones of the InitialHL-GP process (Table 6.4) both for the *full data sets* and for the *top accuracy subsets*. The difference is also statistically significant with regards to the *full solution data sets* (one-tailed $p\text{-}value < 0.0001$).

With regards to the empirical solution performance quantifiers described in Section 6.2.4, ITP-HL-GP was able to produce 2 high accuracy solutions of which one was also small-sized.

7.2.4 Discussion

As a general observation both dynamic depth limiting and pruning have proven to be effective bloat control methods (from a result orientated perspective) as they enabled the HeuristicLab GP process to generally find more parsimonious solutions.

While the usage of dynamic depth limiting does not seem to have a great impact on the overall accuracy of the found solution, the usage of pruning determines a very interesting effect with regards to this performance criterion:

- Generally (i.e. *full solution data sets*) the method determines the GP process to create solutions that are tightly grouped together (i.e. small value of σ) around a fairly good value (when considering the general performance of ANN, SVM and GP regression for the A6 scenario - Table 6.2).

- Specifically (i.e. *top accuracy subsets* and empirical performance quantifiers) the method seems to somehow prevent the GP process from finding highly accurate solutions.

Even in the "lightweight" form that we presented in Section 7.2.3, pruning is a very computational intensive task and as such coupling the method with a static depth limit (Section 7.2.1) with a high enough value is very beneficial from a runtime point of view whilst having no negative impact on solution size or accuracy.

7.3 The Resulting HeuristicLab Bloat Control System

Although the improvements in solution size obtained with both dynamic depth limiting and pruning were quite satisfactory, we decided to test if we could somehow improve on these results by integrating both methods inside a single bloat control system.

Description

From the descriptions provided in the Section 7.2, one can observe that dynamic depth limiting and pruning have two complementary ways of fighting bloat. Whilst the former tries to prevent the creation and propagation of bloat, the latter directly attempts to remove superfluous code from the models. As such, trying to combine both methods in a single integrated solution aimed at combating bloating, seemed a very natural approach.

Thus, we have modified the InitialHL-GP configuration such as to include both the DDL module (see Algorithm 7.1) and the ITP module (see Algorithm 7.2). The static depth limit was also kept (mainly for reasons related to algorithm speed). We have named the resulting configuration BCS-HL-GP.

In our initial test, for the two bloat control methods, we decided to use the same parameter settings from the stand-alone configurations. To our surprise, this approach was quite successful.

Performance

The basic statistics regarding BCS-HL-GP solution performance for all the considered modeling scenarios are listed in Table 7.7 (accuracy) and Table 7.8 (Size). The comparative accuracy kernel density estimation plots are presented in Figure 7.9 whilst the comparative size kernel density estimation plots are presented in Figure 7.10.

	Solution MSE on test set					
	Full data set			**Acc. subset**		
	μ	σ	$\mu_{1/2}$	μ	σ	$\mu_{1/2}$
Blast furnace A5	1.062	0.118	1.065	0.960	0.031	0.960
Blast furnace A6	1.339	0.470	1.210	0.915	0.032	0.900
EvoComp problem	0.057	0.043	0.048	0.044	0.001	0.045

Table 7.7: Statistical information regarding the solution accuracy of BCS-HL-GP

	Solution size					
	Full data set			**Acc. subset**		
	μ	σ	$\mu_{1/2}$	μ	σ	$\mu_{1/2}$
Blast furnace A5	28.42	12.31	27.00	25.64	11.89	22.00
Blast furnace A6	29.17	11.18	28.00	26.36	10.21	23.00
EvoComp problem	18.16	6.81	18.00	21.32	9.44	18.00

Table 7.8: Statistical information regarding the solution size of BCS-HL-GP

With regards to solution accuracy, the central tendencies of the BCS-HL-GP process are slightly better (smaller MSE) then the ones of InitialHL-GP process both for the *full data sets* and the *top accuracy subsets* in all the considered scenarios. However, the *Mann-Whitney-Wilcoxon test* firmly rejected the hypothesis that any of these differences are statistically significant for the A6 and EvoComp scenarios (one-tailed *p-value*>0.2). The test confirmed that in the case of the A5 scenario the observed difference in central tendencies is marginally significant with a one-tailed *p-value* of 0.0409.

With regards to solution size, the central tendencies of the BCS-HL-GP process are considerably smaller then the ones of the InitialHL-GP process (Table 6.4) both for the *full data sets* and the *top accuracy subsets* in all the considered scenarios. In all three cases, the difference is also statistically significant with regards to the *full solution data sets* (one-tailed *p-value*< 0.0001).

Interestingly, when considering the A6 scenario, with regards to solution size, the central tendencies of the BCS-HL-GP configuration are also smaller then the central tendencies of both the DDL-HL-GP configuration and the ITP-HL-GP configuration. The difference with regards to DDL-HL-GP is statistically significant with a one-tailed *p-value* of 0.0183 whilst the difference with regards to ITP-HL-GP is statistically significant with a one-tailed *p-value* smaller then 0.0001.

Furthermore, also related to the A6 scenario, when considering the empirical solution performance quantifiers, the BCS-HL-GP configuration was able to produce 13 high accuracy solutions of which no fewer than 11 also have a small size.

Indicator	μ	σ
SolGen	13.24	2.12
MinModSize	11.12	4.80
AvgModSize	32.94	9.36
AvgSolSize	26.36	10.21

Table 7.9: BCS-HL-GP population dynamics indicators for the *top accuracy subset* of the A6 modeling scenario

For the A6 modeling scenario, the mean population dynamics indicators (see Section 6.2.4) for the top accuracy solution subset of the BCS-HL-GP configuration are presented in Table 7.9 together with the average solution size measure. When comparing these mean values with the ones reported for the InitialHL-GP configuration (Table 6.6) we observe that:

- The BCS-HL-GP configuration converges faster towards the best validation designated solution.
- The mean values of the *MinModSize* and *AvgModSize* indicators for BCS-HL-GP are twice as small as the ones for BCS-HL-GP.
- The average solution size for the BCS-HL-GP configuration is also twice as small.
- The σ values for all the presented indicators are significantly smaller in the case of the BCS-HL-GP configuration (i.e. the configuration is more robust).

These observations provide further empirical evidence that the proposed bloat control system is very efficient and enables the enhanced HeuristicLab GP process to find high quality parsimonious solutions.

Discussion

The results obtained after fitting the InitialHL-GP process with a bloat control system based on dynamic depth limiting and iterated stochastic pruning demonstrate the success of this approach with regards to the overall goal of this thesis: improving overall GP solution parsimony without impacting overall GP solution accuracy.

When considering the three test modeling scenarios presented in Section 4.1 and the results obtained by the classical regression methods and the InitialHL-GP process (Table 6.2) we can state that the presented bloat control system enables the GP process to produce very competitive, highly interpretable, regression models:

- For the A5 scenario, BCS-HL-GP found a solution of size 28 with a MSE on the test set of 0.88.
- For the A6 scenario, BCS-HL-GP found a solution of size 18 with a MSE on the test set of 0.86.
- For the EvoComp scenario, BCS-HL-GP found a solution of size 12 with a MSE on the test set of 0.039.

Without using any form of parameter tuning, the simple combination of dynamic depth limiting and pruning was able to significantly outperform the results of both bloat control methods considered separately. This is a very important observation as it allows for future performance improvement, whilst at the same providing strong proof that the approach of simultaneously trying to combat bloat from two complementary perspectives (prevention thorough DDL and active removal thorough ITP) is quite successful.

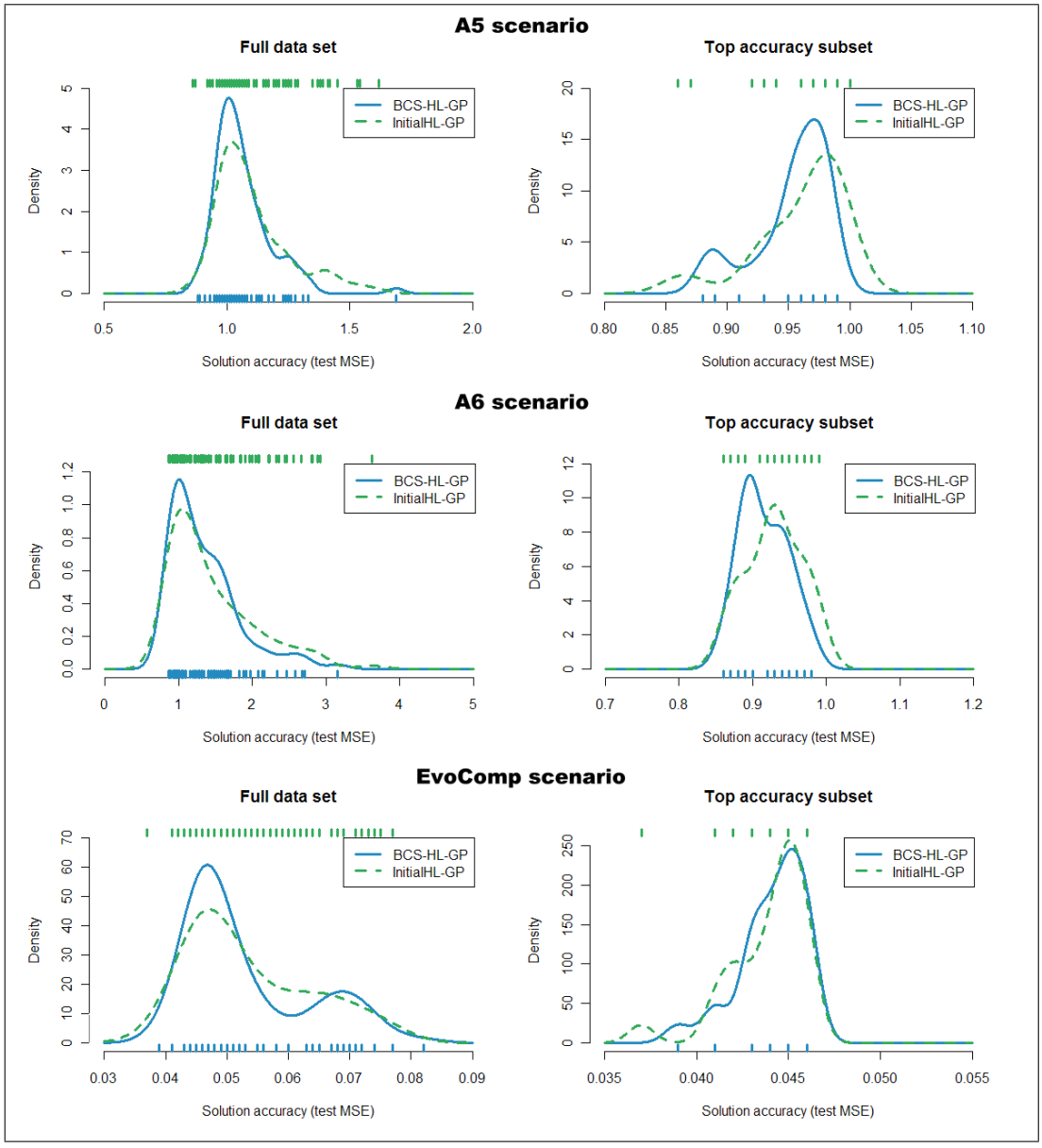


Figure 7.9: Comparative kernel density estimation of solution accuracy for BCS-HL-GP and InitialHL-GP - all test scenarios

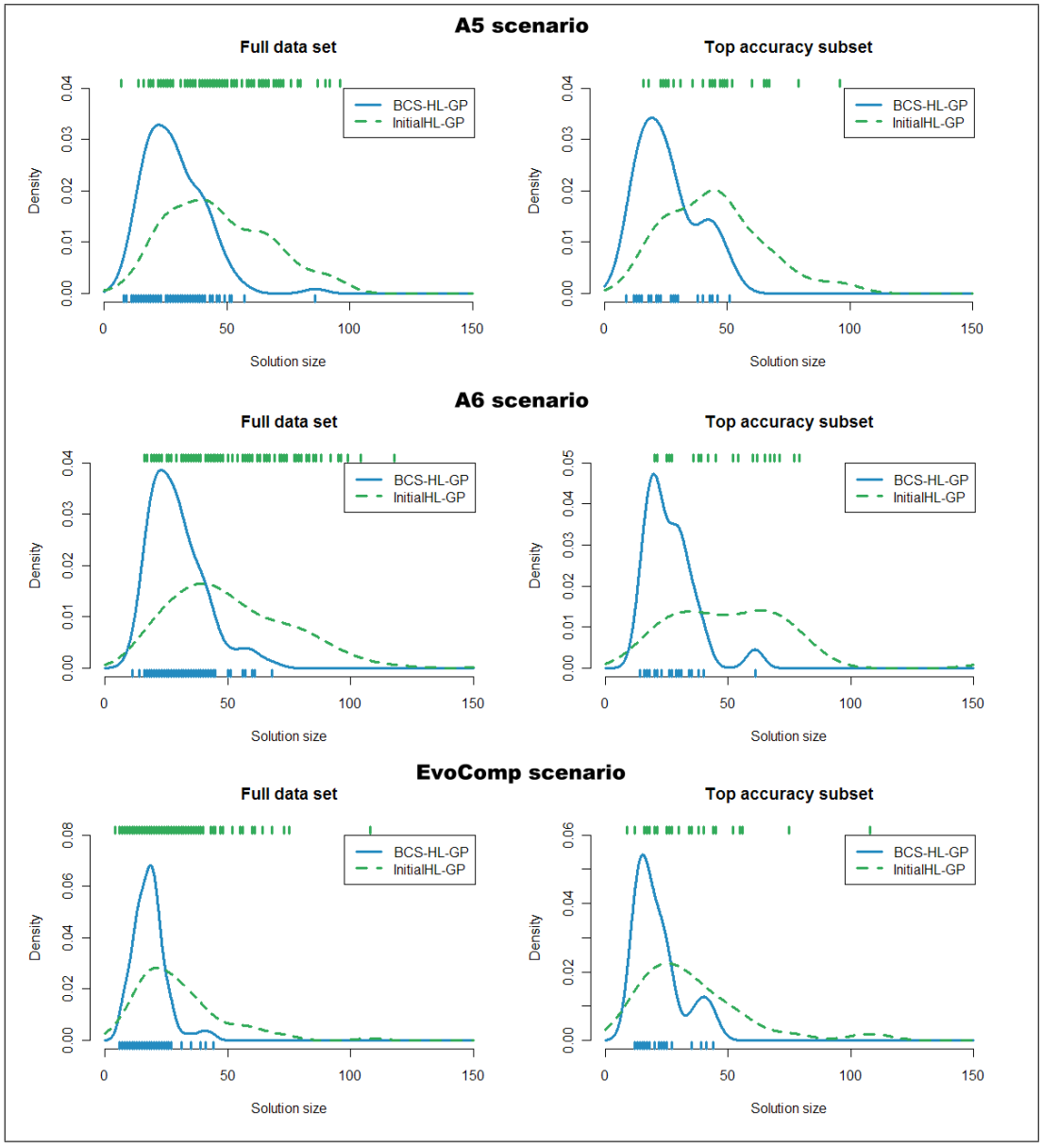


Figure 7.10: Comparative kernel density estimation of solution size for BCS-HL-GP and InitialHL-GP - all test scenarios

Chapter 8

Conclusion

8.1 Achievements

The main aim of our work was to investigate the possibility of reducing the size of the symbolic regression solutions produced by the enhanced genetic programming (GP) process implemented in HeuristicLab (an optimization framework developed by the Heuristic and Evolutionary Algorithms Laboratory (HEAL) of the Upper Austria University of Applied Sciences).

Before setting out to achieve our main goal, we first had to define a suitable methodology (Section 6.1.3) that would enable us to make an objective comparison between the results produced by different GP configurations. Thus, throughout our work, whenever comparing between GP configurations, we rely on average tendencies related to *full solution data sets* (100 solutions/configuration) and *top accuracy subsets* (25 solutions/configuration), on significance testing (the *Mann-Whitney-Wilcoxon* test) and on empirical solution performance quantifiers (Section 6.2.4).

One of the main enhancements the HeuristicLab GP process uses is the offspring selection strategy [2]. Although this enhancement is of great importance, as it enables the GP process to generally produce solutions of much higher quality than a standard GP process, the tests we have performed revealed that the offspring selection strategy also seems to determine a slight but statistically significant increase in average solution size (in comparison to a standard GP approach).

After initially having presented the influence of the code bloating phenomenon on the size of the solution produced by the HeuristicLab GP process, we continued by reviewing various bloat control methods and, in the end, we adapted, implemented and independently tested two different bloat control concepts that, by design, were very easy

to integrate with the HeuristicLab GP process in general and the offspring selection strategy in particular. Both techniques managed to determine a significant reduction in GP solution size.

The two bloat control methods we have considered propose different approaches in combating the phenomenon: prevention (dynamic depth limiting - DDL) and active removal (iterated tournament pruning - ITP). The natural observation that these two paradigms, though different, are also highly complementary, determined us to incorporate both in a single bloat control system. The resulting bloat control solution performed significantly better than the individual bloat control approaches even without any specific parameter tuning.

The fact that the HeuristicLab bloat control system is very robust and successful with regards to the goal that we initially set out to achieve is confirmed by the following observations:

- it helped reduce the mean size of GP symbolic regression models by $\approx 40\%$ on all three test scenarios;
- it helped reduce the standard deviation with regards to solution size by $\approx 40\%$ on the A5 scenario and by $\approx 60\%$ on the A6 and EvoComp scenarios
- it exhibited no negative impact on the solution accuracy distributions

8.2 Future Perspectives

In spite of the very good results obtained by the final bloat control system, we consider that, at the current stage of development, its main function is that of a proof of concept with regards to the successful combination of two bloat combating methods that are based on different paradigms.

Further development of the bloat control system can be firstly achieved by independently improving on its two major parts: the DDL module and the ITP module. In the form presented in this thesis, both modules use certain parameters for which we have empirically determined values that perform well for our studied modeling scenarios. While, especially in the case of DDL, the proposed values exhibited a remarkable stability, the need for a more context independent parametrization schema is very obvious. With regards to the ITP strategy we have proposed, we believe that a significant improvement in terms of performance and runtime can be achieved by dynamically

adjusting (some of) the used parameters during the run. An idea is to base these adjustments on specialized GP information regarding genetic propagation, population diversity as well as on population dynamics indicators.

Secondly, an overall improvement of the bloat control system may come from trying to better understand and improve the interaction between the DDL and ITP modules. Also, the opportunity of integrating other bloat control methods (like anti-bloat selection and anti-bloat genetic operators) should be studied in more detail.

Last but not least, we think that the general design of the bloat control system based on DDL and ITP will enable it to perform quite well in other GP application domains as well, and a study in this direction should be performed in the near future.

Bibliography

[1] Affenzeller, M. Personal communication, February 2010.

[2] Affenzeller, M., and Wagner, S. Offspring selection: A new self-adaptive selection scheme for genetic algorithms. In *Adaptive and Natural Computing Algorithms* (2005), B. Ribeiro, R. F. Albrecht, A. Dobnikar, D. W. Pearson, and N. C. Steele, Eds., Springer, pp. 218–221.

[3] Altenberg, L. The evolution of evolvability in genetic programming. In *Advances in Genetic Programming*, K. E. Kinnear, Jr., Ed. MIT Press, 1994, ch. 3, pp. 47–74.

[4] Angeline, P. J. Genetic programming and emergent intelligence. In *Advances in Genetic Programming*, K. E. Kinnear, Jr., Ed. MIT Press, 1994, ch. 4, pp. 75–98.

[5] Banzhaf, W., and Langdon, W. B. Some considerations on the reason for bloat. *Genetic Programming and Evolvable Machines 3*, 1 (Mar. 2002), 81–91.

[6] Bojarczuk, C. C., Lopes, H. S., and Freitas, A. A. Genetic programming for knowledge discovery in chest-pain diagnosis. *IEEE Engineering in Medicine and Biology Magazine 19*, 4 (July-Aug. 2000), 38–44.

[7] Buchberger, B. Personal communication, October 2009.

[8] Chang, C.-C., and Lin, C.-J. *LIBSVM: a library for support vector machines*, 2001. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm.

[9] Chen, S.-H., and Liao, C.-C. Agent-based computational modeling of the stock price-volume relation. *Information Sciences 170*, 1 (18 Feb. 2005), 75–100.

[10] Cramer, N. L. A representation for the adaptive generation of simple sequential programs. In *Proceedings of an International Conference on Genetic Algorithms and the Applications* (Carnegie-Mellon University, Pittsburgh, PA, USA, 24-26 July 1985), J. J. Grefenstette, Ed., pp. 183–187.

[11] Crawford-Marks, R., and Spector, L. Size control via size fair genetic operators in the PushGP genetic programming system. In *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference* (New York, 9-13 July 2002), W. B. Langdon, E. Cantú-Paz, K. Mathias, R. Roy, D. Davis, R. Poli, K. Balakrishnan, V. Honavar, G. Rudolph, J. Wegener, L. Bull, M. A. Potter, A. C. Schultz, J. F. Miller, E. Burke, and N. Jonoska, Eds., Morgan Kaufmann Publishers, pp. 733–739.

[12] Darwin, C. *The Origin of Species By Means of Natural Selection, or the Preservation of Favoured Races in the Struggle for Life.* Murray, London, 1859.

[13] Dignum, S., and Poli, R. Crossover, sampling, bloat and the harmful effects of size limits. In *Proceedings of the 11th European Conference on Genetic Programming, EuroGP 2008* (Naples, 26-28 Mar. 2008), M. O'Neill, L. Vanneschi, S. Gustafson, A. I. Esparcia Alcazar, I. De Falco, A. Della Cioppa, and E. Tarantino, Eds., vol. 4971 of *Lecture Notes in Computer Science*, Springer, pp. 158–169.

[14] Eggermont, J., Kok, J. N., and Kosters, W. A. Detecting and pruning introns for faster decision tree evolution. In *Parallel Problem Solving from Nature - PPSN VIII* (Birmingham, UK, 18-22 Sept. 2004), X. Yao, E. Burke, J. A. Lozano, J. Smith, J. J. Merelo-Guervós, J. A. Bullinaria, J. Rowe, P. T. A. Kabán, and H.-P. Schwefel, Eds., vol. 3242 of *LNCS*, Springer-Verlag, pp. 1071–1080.

[15] Eiben, A. E., and Smith, J. E. *Introduction to Evolutionary Computing.* Springer, 2003.

[16] Ekart, A., and Nemeth, S. Z. Selection based on the pareto nondomination criterion for controlling code growth in genetic programming. *Genetic Programming and Evolvable Machines 2*, 1 (Mar. 2001), 61–73.

[17] EvoStar, 2010. http://www.evostar.org/.

[18] Folino, G., Pizzuti, C., and Spezzano, G. Improving cooperative GP ensemble with clustering and pruning for pattern classification. In *GECCO 2006: Proceedings of the 8th annual conference on Genetic and evolutionary computation* (Seattle, Washington, USA, 8-12 July 2006), M. Keijzer, M. Cattolico, D. Arnold, V. Babovic, C. Blum, P. Bosman, M. V. Butz, C. Coello Coello, D. Dasgupta, S. G. Ficici, J. Foster, A. Hernandez-Aguirre, G. Hornby, H. Lipson, P. McMinn, J. Moore, G. Raidl, F. Rothlauf, C. Ryan, and D. Thierens, Eds., vol. 1, ACM Press, pp. 791–798.

[19] Forrest, S., Nguyen, T., Weimer, W., and Le Goues, C. A genetic programming approach to automated software repair. In *GECCO '09: Proceedings of the 11th Annual conference on Genetic and evolutionary computation* (Montreal, 8-12 July 2009), G. Raidl, F. Rothlauf, G. Squillero, R. Drechsler, T. Stuetzle, M. Birattari, C. B. Congdon, M. Middendorf, C. Blum, C. Cotta, P. Bosman, J. Grahl, J. Knowles, D. Corne, H.-G. Beyer, K. Stanley, J. F. Miller, J. van Hemert, T. Lenaerts, M. Ebner, J. Bacardit, M. O'Neill, M. Di Penta, B. Doerr, T. Jansen, R. Poli, and E. Alba, Eds., ACM, pp. 947–954. best paper.

[20] Forsyth, R. BEAGLE A Darwinian approach to pattern recognition. *Kybernetes 10*, 3 (1981), 159–166.

[21] Fukunaga, A., and Stechert, A. Evolving nonlinear predictive models for lossless image compression with genetic programming. In *Genetic Programming 1998: Proceedings of the Third Annual Conference* (University of Wisconsin, Madison, Wisconsin, USA, 22-25 July 1998), J. R. Koza, W. Banzhaf, K. Chellapilla, K. Deb, M. Dorigo, D. B. Fogel, M. H. Garzon, D. E. Goldberg, H. Iba, and R. Riolo, Eds., Morgan Kaufmann, pp. 95–102.

[22] Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., and Witten, I. H. The weka data mining software: An update. *SIGKDD Explorations Volume 11*, 1 (2009), 10–18.

[23] Howard, D., Roberts, S. C., and Ryan, C. Pragmatic genetic programming strategy for the problem of vehicle detection in airborne reconnaissance. *Pattern Recognition Letters 27*, 11 (Aug. 2006), 1275–1288. Evolutionary Computer Vision and Image Understanding.

[24] Kaboudan, M. Genetic programming forecasting of real estate prices of residential single family homes in southern california. *Journal of Real Estate Literature 16*, 2 (2008), 219–239.

[25] Keijzer, M. *Scientific Discovery using Genetic Programming*. PhD thesis, Danish Technical University, Lyngby, Denmark, Mar. 2002.

[26] Keijzer, M. Improving symbolic regression with interval arithmetic and linear scaling. In *Genetic Programming, Proceedings of EuroGP'2003* (Essex, 14-16 Apr. 2003), C. Ryan, T. Soule, M. Keijzer, E. Tsang, R. Poli, and E. Costa, Eds., vol. 2610 of *LNCS*, Springer-Verlag, pp. 70–82.

[27] Keijzer, M. Scaled symbolic regression. *Genetic Programming and Evolvable Machines 5*, 3 (Sept. 2004), 259–269.

[28] Kinnear, Jr., K. E. Evolving a sort: Lessons in genetic programming. In *Proceedings of the 1993 International Conference on Neural Networks* (San Francisco, USA, 28 Mar.-1 Apr. 1993), vol. 2, IEEE Press, pp. 881–888.

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

[30] Koza, J. R., Andre, D., Bennett III, F. H., and Keane, M. *Genetic Programming 3: Darwinian Invention and Problem Solving.* Morgan Kaufman, Apr. 1999.

[31] Koza, J. R., Keane, M. A., Streeter, M. J., Mydlowec, W., Yu, J., and Lanza, G. *Genetic Programming IV: Routine Human-Competitive Machine Intelligence.* Kluwer Academic Publishers, 2003.

[32] Kronberger, G., C.Feilmayr, Kommenda, M., Winkler, S., M.Affenzeller, and Burgler, T. System identification of blast furnace processes with genetic programming. In *Logistics and Industrial Informatics - LINDI* (2009), IEEE Press, pp. 1–6.

[33] Langdon, W. B. The evolution of size in variable length representations. In *1998 IEEE International Conference on Evolutionary Computation* (Anchorage, Alaska, USA, 5-9 May 1998), IEEE Press, pp. 633–638.

[34] Langdon, W. B., and Poli, R. Fitness causes bloat. In *Soft Computing in Engineering Design and Manufacturing* (23-27 June 1997), P. K. Chawdhry, R. Roy, and R. K. Pant, Eds., Springer-Verlag London, pp. 13–22.

[35] Langdon, W. B., and Poli, R. *Foundations of Genetic Programming.* Springer-Verlag, 2002.

[36] Langdon, W. B., and Poli, R. Evolutionary solo pong players. In *Proceedings of the 2005 IEEE Congress on Evolutionary Computation* (Edinburgh, UK, 2-5 Sept. 2005), D. Corne, Z. Michalewicz, M. Dorigo, G. Eiben, D. Fogel, C. Fonseca, G. Greenwood, T. K. Chen, G. Raidl, A. Zalzala, S. Lucas, B. Paechter, J. Willies, J. J. M. Guervos, E. Eberbach, B. McKay, A. Channon, A. Tiwari, L. G. Volkert, D. Ashlock, and M. Schoenauer, Eds., vol. 3, IEEE Press, pp. 2621–2628.

[37] Luke, S. Genetic programming produced competitive soccer softbot teams for robocup97. In *Genetic Programming 1998: Proceedings of the Third Annual Conference* (University of Wisconsin, Madison, Wisconsin, USA, 22-25 July 1998), J. R. Koza, W. Banzhaf, K. Chellapilla, K. Deb, M. Dorigo, D. B. Fogel, M. H. Garzon, D. E. Goldberg, H. Iba, and R. Riolo, Eds., Morgan Kaufmann, pp. 214–222.

[38] Luke, S. Two fast tree-creation algorithms for genetic programming. *IEEE Transactions on Evolutionary Computation 4*, 3 (Sept. 2000), 274–283.

[39] Luke, S. Modification point depth and genome growth in genetic programming. *Evolutionary Computation 11*, 1 (Spring 2003), 67–106.

[40] Maeda, Y., and Kawaguchi, S. Redundant node pruning and adaptive search method for genetic programming. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2000)* (Las Vegas, Nevada, USA, 10-12 July 2000), D. Whitley, D. Goldberg, E. Cantu-Paz, L. Spector, I. Parmee, and H.-G. Beyer, Eds., Morgan Kaufmann, p. 535.

[41] McPhee, N. F., Jarvis, A., and Crane, E. F. On the strength of size limits in linear genetic programming. In *Genetic and Evolutionary Computation – GECCO-2004, Part II* (Seattle, WA, USA, 26-30 June 2004), K. Deb, R. Poli, W. Banzhaf, H.-G. Beyer, E. Burke, P. Darwen, D. Dasgupta, D. Floreano, J. Foster, M. Harman, O. Holland, P. L. Lanzi, L. Spector, A. Tettamanzi, D. Thierens, and A. Tyrrell, Eds., vol. 3103 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 593–604.

[42] McPhee, N. F., and Miller, J. D. Accurate replication in genetic programming. In *Genetic Algorithms: Proceedings of the Sixth International Conference (ICGA95)* (Pittsburgh, PA, USA, 15-19 July 1995), L. Eshelman, Ed., Morgan Kaufmann, pp. 303–309.

[43] Nordin, P., and Banzhaf, W. Complexity compression and evolution. In *Genetic Algorithms: Proceedings of the Sixth International Conference (ICGA95)* (Pittsburgh, PA, USA, 15-19 July 1995), L. Eshelman, Ed., Morgan Kaufmann, pp. 310–317.

[44] Nordin, P., and Banzhaf, W. Programmatic compression of images and sound. In *Genetic Programming 1996: Proceedings of the First Annual Conference* (Stanford University, CA, USA, 28–31 July 1996), J. R. Koza, D. E. Goldberg, D. B. Fogel, and R. L. Riolo, Eds., MIT Press, pp. 345–350.

[45] Poli, R., Langdon, W. B., and Dignum, S. On the limiting distribution of program sizes in tree-based genetic programming. In *Proceedings of the 10th European Conference on Genetic Programming* (Valencia, Spain, 11 - 13 Apr. 2007), M. Ebner, M. O'Neill, A. Ekárt, L. Vanneschi, and A. I. Esparcia-Alcázar, Eds., vol. 4445 of *Lecture Notes in Computer Science*, Springer, pp. 193–204.

[46] Poli, R., Langdon, W. B., and McPhee, N. F. *A field guide to genetic programming*. Published via http://lulu.com and freely available at http://www.gp-field-guide.org.uk, 2008. (With contributions by J. R. Koza).

[47] Ritchie, M. D., White, B. C., Parker, J. S., Hahn, L. W., and Moore, J. H. Optimization of neural network architecture using genetic programming improves detection and modeling of gene-gene interactions in studies of human diseases. *BMC Bioinformatics 4*, 28 (2003).

[48] Silva, S., and Costa, E. Dynamic limits for bloat control in genetic programming and a review of past and current bloat theories. *Genetic Programming and Evolvable Machines 10*, 2 (2009), 141–179.

[49] Soule, T. *Code Growth in Genetic Programming*. PhD thesis, University of Idaho, Moscow, Idaho, USA, 15 May 1998.

[50] Soule, T., and Foster, J. A. Code size and depth flows in genetic programming. In *Genetic Programming 1997: Proceedings of the Second Annual Conference* (Stanford University, CA, USA, 13-16 July 1997), J. R. Koza, K. Deb, M. Dorigo, D. B. Fogel, M. Garzon, H. Iba, and R. L. Riolo, Eds., Morgan Kaufmann, pp. 313–320.

[51] Soule, T., and Foster, J. A. Removal bias: a new cause of code growth in tree based evolutionary programming. In *1998 IEEE International Conference on Evolutionary Computation* (Anchorage, Alaska, USA, 5-9 May 1998), IEEE Press, pp. 781–186.

[52] Soule, T., and Heckendorn, R. B. An analysis of the causes of code growth in genetic programming. *Genetic Programming and Evolvable Machines 3*, 3 (Sept. 2002), 283–309.

[53] Spector, L. *Automatic Quantum Computer Programming: A Genetic Programming Approach*, vol. 7 of *Genetic Programming*. Kluwer Academic Publishers, Boston/Dordrecht/New York/London, June 2004.

[54] Vapnik, V. N. *The nature of statistical learning theory.* Springer-Verlag New York Inc., New York, NY, USA, 1995.

[55] Wagner, S., and Affenzeller, M. Heuristiclab: A generic and extensible optimization environment. In *Adaptive and Natural Computing Algorithms* (2005), B. Ribeiro, R. F. Albrecht, A. Dobnikar, D. W. Pearson, and N. C. Steele, Eds., Springer, pp. 538–541.

[56] Wagner, S., and Affenzeller, M. SexualGA: Gender-specific selection for genetic algorithms. In *Proceedings of the 9th World Multi-Conference on Systemics, Cybernetics and Informatics (WMSCI)* (2005), N. Callaos, W. Lesso, and E. Hansen, Eds., vol. 4, International Institute of Informatics and Systemics, pp. 76–81.

[57] Winkler, S. M. *Evolutionary System Identification.* PhD thesis, Johannes-Kepler-Universitat, Linz, Austria, 2008.

[58] Winkler, S. M., Affenzeller, M., and Wagner, S. On the reliability of nonlinear modeling using enhanced genetic programming techniques. In *Topics on Chaotic Systems - Selected Papers from CHAOS 2008 International Conference* (2009), C. S. C. Skiadas, I. Dimotikalis, Ed., World Scientific Publishing, pp. 398–405.

[59] Winkler, S. M., Affenzeller, M., and Wagner, S. Using enhanced genetic programming techniques for evolving classifiers in the context of medical diagnosis. *Genetic Programming and Evolvable Machines 10*, 2 (2009), 111–140.

[60] Xie, H., Zhang, M., and Andreae, P. Genetic programming for automatic stress detection in spoken english. In *Applications of Evolutionary Computing, EvoWorkshops2006: EvoBIO, EvoCOMNET, EvoHOT, EvoIASP, EvoInteraction, EvoMUSART, EvoSTOC* (Budapest, 10-12 Apr. 2006), F. Rothlauf, J. Branke, S. Cagnoni, E. Costa, C. Cotta, R. Drechsler, E. Lutton, P. Machado, J. H. Moore, J. Romero, G. D. Smith, G. Squillero, and H. Takagi, Eds., vol. 3907 of *LNCS*, Springer Verlag, pp. 460–471.

[61] Zhang, B.-T., and Mühlenbein, H. Balancing accuracy and parsimony in genetic programming. *Evolutionary Computation 3*, 1 (1995), 17–38.

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Masterarbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt bzw. die wörtlich oder inhaltlich entnommenen Stellen deutlich als solche kenntlich gemacht habe.

Hagenberg, Juli 2010 Alexandru-Ciprian Zăvoianu