UNIVERSITY OF MILANO BICOCCA

DOCTORAL SCHOOL OF SCIENCE

PHD THESIS

Measures and Methods for Robust Genetic Programming

Defended by

Mauro CASTELLI

Thesis Advisor: Leonardo VANNESCHI

Thesis Supervisor: Sara SILVA

defended on February, 2012

Acknowledgments

I would like to show my deep and sincere gratitude to my supervisor, professor Leonardo Vanneschi for the continuous support of my PhD study and research, for his patience, motivation, enthusiasm, and immense knowledge. His guidance helped me in all the time of research and writing of this thesis. I could not have imagined having a better supervisor and friend for my PhD study.

I thank my lab mates: Stefano, Luca, Enrico, Carlo for the stimulating discussions and for all the fun we have had in the last three years. Special thanks to Stefano: we spent together the last 8 years in university, he is a true and irreplaceable friend. I could not have imagined having a better companion of adventure. Thank you for bear and support me!

Also I am heartily thankful to my friend Laura, because she has made available her support in a number of ways and for all the fun we have had in the last three years. When I became too serious, her words allowed me to laugh and lightened my perspective. Thank You Laura!

Other friends I must mention are Ilaria, Gaia and Yuri. Thank you for your support and most of all your humor. You both kept things light and me smiling. Thank you for all the fun we have had during our breakfasts!

Thanks also to all the people I met during these years of university: Christian, Mara, Lorenza, Luca and all the people who I forget to mention. Thank you for everything you taught me and for the moments spent together.

Thanks to Diego: we spent together the last 8 years in university, he was a colleague and now he is above all a friend.

I am grateful to my parents for their understanding, endless patience and encouragement when it was most required. Both have instilled many admirable qualities in me and given me a good foundation with which to meet life. They have taught me about hard work respect and persistence.

And last but not least I must acknowledge with tremendous and deep thanks Lara, not necessarily for coming along at the right time, but for the very special person she is. I still have not found the words that describe or express how I feel for this woman and what her presence in my life has meant. Without her love and encouragement, I would not have finished this thesis. Thank you with all my heart.

Contents

List of Tables

List of Figures

CHAPTER 1

Introduction

The aim is...to get machines to exhibit behaviour, which if done by humans, would be assumed to involve the use of intelligence.

Arthur Samuel

Contents

1.1 Introduction

This thesis focuses on genetic programming (GP), a machine learning technique inspired by Darwin's theory of evolution.

According to [83] GP is a systematic method for getting computers to automatically solve a problem starting from a high-level statement of what needs to be done. GP is a domain-independent method that genetically breeds a population of computer programs to solve a problem. Specifically, GP iteratively transforms a population of computer programs into a new generation of programs by applying analogs of naturally occurring genetic operations. A flow chart of the standard GP algorithm is reported in figure 1.1. A detailed description of the GP algorithm is presented in chapter 2.

GP is considered a bio-inspired method and, like all the evolutionary computation methods, it is inspired by Darwin's theory of evolution. According to this theory the vast majority of the history of life can be fully accounted for by physical processes operating on and within populations and species. These processes are reproduction, mutation, competition, and selection. Over many generations, these processes shape the behaviours of individuals and species to fit the demands of their surroundings. While evolution has no intrinsic purpose, it is capable of engineering solutions to the problems of survival that are unique to each individual's circumstance and, by any measure, quite ingenious. According to [36], combining the evolutionary process within a computer could provide a means for addressing complex engineering problems that traditional algorithms have been unable to conquer. Indeed, the field of evolutionary computation is one of the fastest growing areas of computer science and engineering for just this reason; it is addressing many problems that were previously beyond reach,

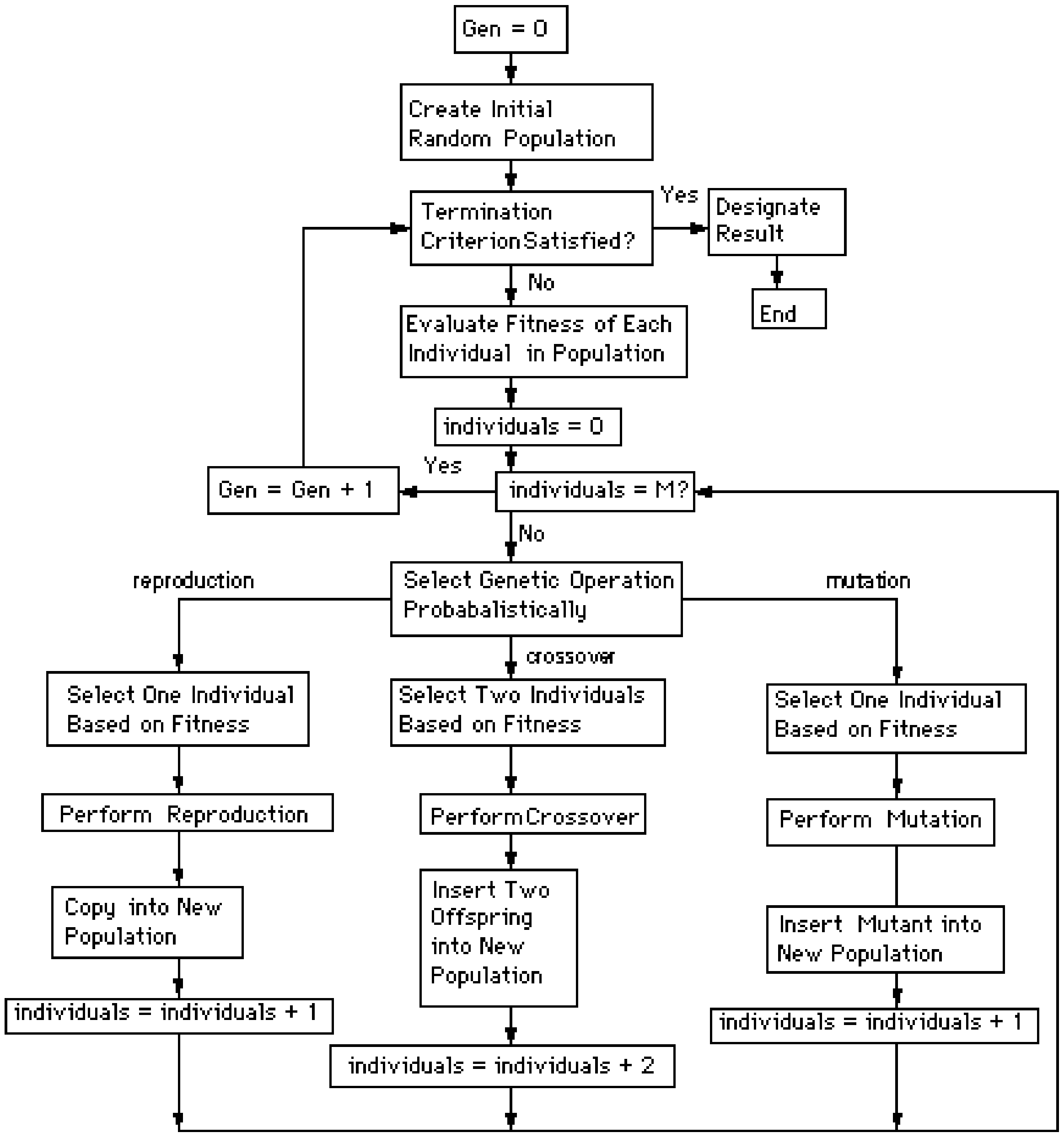

Gen = 0
Create Initial Random Population
Termination Criterion Satisfied?
Yes
Designate Result
End
No
Evaluate Fitness of Each Individual in Population
individuals = 0
Yes
Gen = Gen + 1
individuals = M?
No
reproduction
Select Genetic Operation Probabalistically
mutation
crossover
Select One Individual Based on Fitness
Select Two Individuals Based on Fitness
Select One Individual Based on Fitness
Perform Reproduction
Perform Crossover
Perform Mutation
Copy into New Population
Insert Two Offspring into New Population
Insert Mutant into New Population
individuals = individuals + 1
individuals = individuals + 2
individuals = individuals + 1

Figure 1.1: Genetic Programming flowchart. *M* is the size of the population.

such as rapid design of medicines. Potentially, the field may fulfill the dream of artificial intelligence: a computer that can learn on its own and become an expert in any chosen area.

1.2 Contributions

This thesis presents contributions in different areas of the field of GP. These contributions are reported in the following sections.

1.2.1 Measures for Genetic Programming

While GP has been successfully used in different domains, there a still a lot of open problems. In particular, the study of different (and maybe related) phenomena that characterize GP deserves an accurate analysis. These phenomena, that are discussed below, are *bloat*, *overfitting* and the *functional complexity* of the solutions.

The most accepted definition of bloat is “program growth without (significant) return in terms of fitness” (definition taken from [82], page 101). This definition is clearly a very fuzzy one and in many bibliographic references GP problems are analyzed in terms of “presence” or “absence” of bloat, without any numerical quantification. This is the case, among many other references, of [95] where the authors state “Standard GP bloats” and “Dynamic Operator does not bloat”, or of [82] where the authors use informal expressions like “bloat is more marked” (page 41), “reduce bloat” (page 78) or “bloat can ... happen” (page 106). These expressions are really too informal to permit a deep study of bloat. Hence, the first contribution of this thesis is a simple measure to quantify the “amount of bloat of a GP run” with a single number.

Following [74] (page 67), overfitting can be defined as follows: “Given a hypothesis space H, a hypothesis $h \in H$ is said to *overfit* the training data if there exists some alternative hypothesis $h' \in H$, such that h has smaller error than h' over the training samples, but h' has smaller error than h over the entire distribution of instances".

According to [13], there are basically three approaches to avoid overfitting: (1) penalizing complexity or biasing toward simplicity, (2) limiting the number of models considered, (3) using a validation data set. Penalizing complexity or biasing toward simplicity may involve post-pruning of models or generating and hypothesizing models in the order from simple to complex or searching in the space of solutions from general to specific and using some stopping criterion. There have been a number of studies ([17], [23], [73]) where accuracy has not been reduced or has even been improved as a result of simplifying trees by pruning. There have also been theoretical arguments in favor of what has sometimes been referred to as Occam's Razor, namely that simpler models have greater predictive power and lead to less generalization error ([15], [48]). Following Jensen and Cohen [53], Domingos in [34] regards the number of models considered rather than the complexity of the models as leading to overfitting. For GP, overfitting is often avoided by limiting the number of generations, and limiting the size of the population would also result in reducing the number of models considered. A validation data set can be used to directly test generalization errors and thus directly decide between different models. It can be used to cutoff search, thus limiting the number of models considered. Such a cutoff can also prevent complexity when the search is biased from simple to complex.

All these approach only use qualitative information because there is no measure that can quantify the amount of overfitting with a single number. Defining a measure for overfitting is clearly a hard task, given that, according to the definition of [74], it requires the exploration of all the hypothesis space H, looking for such a hypothesis h'.

Another contribution of this thesis is the definition of an overfitting measure that can quantify the amount of overfitting that affect a GP run. Having a quantitative measure can result in more sophisticated and effective methods to counteract overfitting.

The issue of generalization in GP has received a growing attention in the last few years (see [61] for a survey). In 1995 Zhang and Mühlenbein investigated the relationship between generalization and parsimony and proposed an adaptive learning method that automatically balances these two factors. One year later, in [37], a new GP system called Compiling GP System was introduced and in [7], Banzhaf and coworkers showed the positive effect of an extensive use of the mutation operator on generalization in GP using sparse data sets. More recently, in [39], Gagné and coworkers have investigated two methods to improve generalization in GP: the selection of individuals using a three data sets methodology, and the application of parsimony pressure to reduce the size of the solutions. In the last few years the idea of quantitatively studying the relationship between generalization and solution complexity was tackled in several contributions. For instance, in [2] authors propose a theoretical analysis of GP from the perspective of statistical learning theory and prove the advantage of a parsimonious fitness using Vapnik-Chervonenkis theory. Another important contribution, even though not explicitly focused on GP, but on evolutionary algorithms in general, is presented in [42], where authors measure behavioral complexity explicitly using compression, and use it as a separate objective to be optimized. In [111], the authors define two measures of complexity for GP individuals: a genotypic measure and a phenotypic one. While the genotypic measure is related to counting the number of nodes of a tree and its subtrees, the phenotypic one, called *order of nonlinearity*, is related to functional complexity and consists in calculating the degree of the Chebyshev polynomial approximation of the function. The authors then use these two measures as criteria in a multi-objective system and they show that this system is able to counteract both bloat and overfitting.

In this thesis different complexity measures have been proposed. The first measure is inspired by the concept of curvature [20] but, differently from curvature it is always computable. A second measure, called graph based complexity (GBC), to quantify the functional complexity of GP individuals has also been proposed. Compared to the firstly defined measure, GBC is rotationally invariant and it has a higher correlation with the quality of GP solutions on out-of-sample data. A third measure to quantify the ability of GP to learn "difficult" points has also been proposed and discussed.

1.2.2 Generalization Ability of Genetic Programming

As stated in section 1.2.1 the issue of generalization has received a growing attention in the last few years. While one contribution of this thesis focuses on the definition of measures to quantify overfitting and related phenomena, an other important contribution is the definition of methods to reduce overfitting.

The first contribution is based on a simple idea. The idea is to re-use genetic material from older generations. Even though similar to the concept of "short-term" memory, which is typical of Tabu Search [41], the proposed method is new. Individuals belonging to "old" generations are allowed to participate again to the selection process at given time intervals, thus giving them a second chance to take part in mating. Motivations for introducing this method are the following. First of all it is possible hypothesize that, generation by generation, the GP individuals become more and more specialized. Even if this behavior can be a good one, the GP individuals could also overfit training data. The insertion of earlier (and probably less specialized) individuals with a good fitness can allow the population diversity to increase, reducing at the same time the risk of overfitting training data. The second motivation is that the processes of selection, crossover and mutation of GP can discard (the former one) or disrupt (the latter ones) good genetic material. Allowing a reinsertion, the probability of good genetic material to be propagated increases. The third motivation is that, also on training data themselves, the evolutionary process may lead the population to convergence towards local optima. The insertion of earlier individuals with a good fitness, while allowing the population diversity to increase, should also reduce the risk of premature convergence and stagnation in local optima.

The second important contribution is a study on the generalization ability of a multi objective GP framework (MOGP). In optimization problems the search bias usually involves only one fitness function. This is equivalent to the use of only one criterion to estimate the quality of a solution. On the other hand, in many situations a good solution is achieved by a compromise between multiple criteria [27]. The usage of only one criterium to guide the evolutionary process results, in several applications, in solutions affected by overfitting. Hence, the performances achieved on the training instances could not be replicated on unseen data. It has also been shown in [95,110] that considering a completely bloat free GP framework does not improve the generalization ability of the solutions. In particular the GP framework used in [95,110] produces solutions that, even though rather smaller than the ones produced by standard GP, are not able to generalize better than them. The aim of the study presented in this thesis is to understand whether or not the use of a second criterium, combined with the criterium used as a typical fitness function, could enhance the generalization ability of the solutions. In this light, several auxiliary criteria that are thought to have an effect on the generalization ability have been chosen.

1.2.3 The Role of Semantic in Genetic Programming

GP has been widely investigated in the last decades. The analysis of a GP system is mainly based on the study of genotypic properties. While the study of genotype can be useful to capture particular phenomena related to the evolutionary process, it is not able to describe the entire dynamic of the process.

Thus incorporating semantic awareness in the GP process could improve performances, extending the applicability of GP to problems that are difficult with purely syntactic approaches. For this reason the study of GP systems has been extended to phenotypic aspects. This type of analysis does not consider the structural characteristics of a tree, but other properties related to the fitness. So a phenotypic study is strongly based on the fitness of the individuals involved in the evolutionary process.

A key factor in the success or otherwise of a GP population in evolving towards a solution is the extent of diversity amongst its members. Diversity may be viewed in genotypic (structural) or in phenotypic (behavioural) terms, but the latter has received less attention. Overviews of diversity measures can be found in [78] and [18], while Burke *et al.* in [19] give a more extensive analysis of these measures and of how they relate to fitness.

Behavioural or phenotypic diversity metrics are based on the functionality of individuals, i.e. the execution of program trees rather than their appearance. Usually, behavioural diversity is viewed in terms of the spread of fitness values obtained on evaluating each member of the population [88]. One way of measuring such a diversity is by considering the fitness distribution as an indicator of entropy, or disorder, in the population [116] [87].

Other approaches consider sets or lists of fitness values and use them in combination with genotypic measures [29]. For certain types of problems it may be possible to achieve the effect of behavioural diversity without invoking the fitness function, via the use of semantic sampling schemes [65]. Semantic analysis of programs is also used in the diversity enhancing technique described by Beadle and Johnson [11].

Gustafson in [45] developed two edit distances to sample semantic diversity in GP and conducted an analysis comparing behavioural diversity measures with changes in fitness. One of the limitations of the edit distance method is that it is defined on representations that differ substantially from the standard one.

Semantic analysis methods are starting to appear in combination with crossover. McPhee *et al.* [71] used truth tables to analyze behavioural changes in crossover for boolean problems. They consider the semantics of two components in each tree: semantic of subtrees and semantic of context (the remainder of an individual after removing a subtree). They experimentally measured the variation of these semantic components throughout the GP evolutionary process. They paid special attention to fixed-semantic subtrees: subtrees where the semantic of the tree does not change when this subtree is replaced by another subtree. They showed that there may be many such fixed semantic subtrees when the tree size increases during evolution; thus it becomes very difficult to change the semantic of trees with crossover and mutation once the trees have become large.

While it is possible to represent behaviours using truth tables, a more efficient technique is that of using reduced ordered binary decision diagrams (ROBDDs) [16] to create reduced canonical representations to measure behavioural differences.

In [10] semantic is used to define an algorithm called Semantically Driven Crossover (SDC). The SDC algorithm has been developed based on the analysis of the behavioural changes caused by crossover. The key feature of this method is the use of a canonical representation of members of the population (reduced ordered binary decision diagrams-ROBDDs) to check for semantic equivalence without having to access the fitness function. Two trees are semantically equivalent if and only if they reduce to the same ROBDD. This is used to determine which participating individuals are copied to the next generation. If the offsprings are semantically equivalent to their parents, the children are discarded and the crossover is restarted. This process is repeated until semantically new children are found. The authors argue that this results in increased semantic diversity in the evolving population, and a consequent improvement in the GP performances.

Other works involving semantic are reported in section 7.2.

In this thesis a semantic niching method is proposed. The idea consists in building, maintaining and updating generation by generation a semantic distribution. Although original, the idea is inspired by the well-known control bloat method proposed in [92]. The similarities between the two methods are limited to the acceptance criteria of new individuals produced by the genetic operators. Beyond this, the basic idea that characterizes the two methods is different: while the work in [92] builds and uses a distribution based on the syntax of the individuals, the work proposed in this thesis focuses on semantics. The main aim is to increase the performances of GP by maintaining population diversity.

1.3 Thesis Overview

Chapter 2 introduces basic concepts about the field of GP. In the first part of the chapter the field of Evolutionary Computation is presented and a discussion regarding advantages (and drawbacks) of Evolutionary Algorithms over other approaches is proposed. The main concepts about GP are covered in section 2.3, where all the components of a standard GP algorithms are described.

In Chapter 3 current issues in GP are discussed and a literature review is proposed. The literature review focuses on three relevant phenomena that affects GP. Programs bloat, overfitting and programs complexity are discussed in detail with references to recent literature.

Chapter 4 presents different measures to quantify bloat, overfitting and complexity of candidate solutions. Strengths and weakness of the proposed measures are discussed and the relations between these phenomena are investigated. In section 4.4 a new measure of functional complexity for GP solutions is proposed. The measure is called Graph Based Complexity (GBC) and it overcomes some of the problems that affect the complexity measure propose in section 4.3. A natural extension of GBC was Graph Based Learning Ability (GBLA) defined in section 4.4. Its goal is to quantify the ability of GP to learn "difficult" training points.

Chapter 5 presents an experimental study of the generalization ability of a set of multi-optimization GP frameworks comparing them with standard GP and Dynamic Operator Equalization [93], a bloat control technique. Section 5.2 introduces Operator Equalization and gives some details regarding how it works; section 5.3 presents basic concepts about multi objective optimization, while in section 5.4 the multi-optimization framework used is presented. Section 5.5 presents the experimental setting, including a description of the datasets and of the parameters used; section 5.6 presents the obtained experimental results. Finally Section 5.7 concludes the chapter and discusses ideas for future research.

Chapter 6 presents a method to increase the generalization ability of GP. The idea, that consists in giving a second chance of mating to individuals belonging to "old" generations, is presented in section 6.2. In section 6.3 experimental settings and results are presented and discussed. Section 6.4 concludes the chapter and suggests some ideas for future works

Chapter 7 presents a semantic niching method to increase the performance of GP. The idea consists in building, maintaining and updating generation by generation a semantic distribution. Section 7.2 presents the state of the art method regarding the study of semantic in GP. The section focuses on the different approaches used to study the dynamic of the evolutionary process, presenting strengths and weaknesses of semantic and syntactical methods. Section 7.3 outlines the details of the proposed method, while section 7.4 and section 7.5 present the test functions and the experimental settings respectively. Results are discussed in section 7.6 and in section 7.7. Section 7.8 discusses some problems that affect the proposed semantic method while section 7.9 concludes the chapter and gives some possible directions for future research.

Chapter 8 summarizes the main contributions of this work.

Chapter 9 concludes this thesis presenting some hints for future research.

CHAPTER 2

Genetic Programming: Introduction

Contents

The aim of this chapter is to introduce the basic concepts about the field of Genetic Programming. Section 2.1 introduces Evolutionary Computation (EC) while section 2.2 outlines the possible advantages (and disadvantages) in using such family of bio-inspired techniques. The main concepts about GP are covered in section 2.3, where all the components of a standard GP algorithm are described. Most of this chapter is taken from [82] [9] [5].

2.1 Evolutionary Algorithms

Evolutionary algorithms (EAs) are stochastic optimization techniques based on the principles of natural evolution. An EA is a stochastic iterative procedure for generating tentative solutions for a certain problem Π. There are different EA models: Evolutionary Programming, Evolutionary Strategies and Genetic Algorithms. A complete description of EA families can be found in [6]. These families have not grown in complete isolation from each other and new variants have emerged. One of this variant is GP and it is described in section 2.3.

2.2 Advantages (and drawbacks) of evolutionary algorithms over other approaches

Since according to the no-free-lunch (NFL) theorem [115], there cannot exist any algorithm for solving all (e.g. optimization) problems that is generally (on average) superior to any competitor, the question of whether evolutionary algorithms (EAs) are inferior/superior to any alternative approach is senseless. What could be claimed solely is that EAs behave better than other methods with respect to solving a specific class of problems (with the consequence that they behave worse for other problem classes).

The NFL theorem can be corroborated in the case of EAs versus many classical optimization methods insofar as the latter are more efficient in solving linear, quadratic, strongly convex, unimodal, separable, and many other special problems. On the other hand, EAs do not give up so early when discontinuous, nondifferentiable, multimodal, noisy, and otherwise unconventional response surfaces are involved. Their effectiveness (or robustness) thus extends to a broader field of applications, of course with a corresponding loss in efficiency when applied to the classes of simple problems classical procedures have been specifically devised for.

Looking into the historical record of procedures devised to solve optimization problems, especially around the 1960s (see [90]), when a couple of direct optimum-seeking algorithms were published, for example, in the Computer Journal, a certain pattern of development emerges. Author A publishes a procedure and demonstrates its suitability by means of tests using some test functions. Next, author B comes along with a counterexample showing weak performance of A's algorithm in the case of a certain test problem. Of course, he also presents a new or modified technique that outperforms the older one(s) with respect to the additional test problem. This game could in principle be played $ad infinitum$.

A better means of clarifying the scene ought to result from theory. This should clearly define the domain of applicability of each algorithm by presenting convergence proofs and efficiency results. Unfortunately. however, it is possible to prove abilities of algorithms only by simplifying them as well as the situations to which they are confronted. The huge remainder of questions must be answered by means of (always limited) test series, and even that cannot tell much about an actual real-world problem-solving situation with yet unanalyzed features, that is, the normal case in applications.

Again unfortunately, there does not exist an agreed-upon test problem catalogue to evaluate old as well as new algorithms in a concise way. It is doubtful whether such a test bed will ever be agreed upon, but efforts in that direction would be worthwhile.

Finally, what are the truths and consequences? First, there will always remain a dichotomy between efficiency and general applicability, between reliability and effort of problem-solving, especially optimum-seeking, algorithms. Any specific knowledge about the situation at hand may be used to specify an adequate specific solution algorithm, the optimal situation being that one knows the solution in advance. On the other hand, there cannot exist one method that solves all problems effectively as well as efficiently. These goals are contradictory. If there is already a traditional method that solves a given problem, EAs should not be used. They cannot do it better or with less computational effort.

To develop a new solution method suitable for a problem at hand may be a nice challenge to a theoretician, who will afterwards get some merit for his effort, but from the application point of view the time for developing the new technique has to be added to the computer time invested. In that respect, a non specialized, robust procedure (and EAs belong to this class) may be, and often proves to be, worthwhile.

A warning should be given about a common practice: the linearization or other decomplexification of the situation in order to make a traditional method applicable. Even a guaranteed globally optimal solution for the simplified task may be a long way off and thus greatly inferior to an approximate solution to the real problem.

The best one can say about EAs, therefore, is that they present a methodological framework that is easy to understand and handle, and is either usable as a black-box method or open to the incorporation of new or old recipes for further sophistication, specialization or hybridization. They are applicable even in dynamic situations where the goal or constraints are moving over time or changing, either exogenously or self-induced, where parameter adjustments and fitness measurements are disturbed, and where the landscape is rough, discontinuous, multimodal, even fractal or cannot otherwise be handled by traditional methods, especially those that need global prediction from local surface analysis.

There exist EA versions for multiple criterion decision making (MCDM) and many different parallel computing architectures. Almost forgotten today is their applicability in experimental (non-computing) situations.

Sometimes striking is the fact that even obviously wrong parameter settings do not prevent fairly good results: this certainly can be described as robustness. Not yet well understood, but nevertheless very successful are those EAs which self-adapt some of their internal parameters, a feature that can be described as collective learning of the environmental conditions. Nevertheless, even selfadaptation does not circumvent the NFL theorem.

2.3 Genetic Programming

GP is an evolutionary computation technique that automatically solves problems without requiring the user to know or specify the form or structure of the solution in advance. At the most abstract level GP is a systematic, domain-independent method for getting computers to solve problems automatically starting from a high-level statement of what needs to be done. In GP a population of computer programs is evolved. That is, generation by generation, GP stochastically transforms populations of programs into new, hopefully better, populations of programs. The important features shared by most

GP systems are

- Stochastic decision making. GP uses pseudo-random numbers to mimic the randomness of natural evolution. As a result, GP uses stochastic processes and probabilistic decision making at several stages of program development.
- Program structures. GP assembles variable length program structures from basic units called functions and terminals. Functions perform operations on their inputs, which are either terminals or output from other functions. The actual assembly of the programs from functions and terminals occurs at the beginning of a run, when the population is initialized
- Genetic operators. GP transforms the initial programs in the population using genetic operators. Crossover between two individual programs is one principal genetic operator in GP. Other important operators are mutation and reproduction.
- Simulated evolution of a population by means of fitness based selection. GP evolves a population of programs in parallel. The driving force of this simulated evolution is some form of fitness-based selection. Fitness-based selection determines which programs are selected for further improvements.

2.3.1 Terminals and Functions: the Primitives of Genetic Programs

The functions and terminals are the primitives with which a program in GP is built. Functions and terminals play different roles. The terminal set is comprised of the inputs to the GP program, the constants supplied to the GP program, and the zero-argument functions with side-effects executed by the GP program. The function set is composed of the statements, operators, and functions available to the GP system. Loosely speaking, terminals *provide* a value to the system while functions *process* a value already in the system. Together, functions and terminals are referred to as *nodes*.

Choosing the Function and Terminal Set

The functions and terminals used for a GP run should be powerful enough to be able to represent a solution to the problem. For example, a function set consisting only of the addition operator will probably not solve many very interesting problems. On the other hand, it is better not to use too large a function set. This enlarges the search space and can sometimes make the search for a solution harder. An approximate starting point for a function set might be the arithmetic and logic operations.

Another important property of the function set is that each function should be able to handle gracefully all values it might receive as input. This is called the *closure* property. The most common example of a function that does not fulfill the closure property is the division operator. The division operator cannot accept zero as an input. Division by zero will normally crash the system, thereby terminating a GP run. This is of course unacceptable. Instead of a standard division operator one may define a new function called protected division. Protected division is just like normal division except for zero denominator inputs. In that case, the function returns something else, i.e., a very big number or zero. All functions (square root and logarithms are other examples) must be able to accept all possible inputs because if there is any way to crash the system, the boiling genetic soup will certainly hit upon it.

2.3.2 Executable Program Structures

The primitives of GP (the functions and terminals) are not programs. Functions and terminals must be assembled into a structure before they may execute as programs. The evolution of programs is, of course, common to all genetic programming. Programs are structures of functions and terminals together with rules or conventions for when and how each function or terminal is to be executed.

The choice of a program structure in GP affects execution order, use and locality of memory, and the application of genetic operators to the program. There are really two very separate sets of issues here. Execution and memory locality are phenomic issues (that is, issues regarding the behavior of the program). On the other hand, mutation and crossover are genomic issues (that is, how the "DNA" of the program is altered. In most tree-based GP systems, there is no separate phenotype. Therefore, it appears that structural issues of execution, memory, and variation are the same. But that similarity exists only because of an implicit choice to blend the genome and the phenome.

The three principal program structures used in GP are tree, linear, and graph structures. However, GP program structures are often virtual structures. For example, tree and graph structures are executed and altered as i/they were trees or graphs. But how a program executes or is varied is a completely different question from how it is actually held in computer memory. Many tree-based systems do not actually hold anything that looks like a tree in the computer. Of the three fundamental structures, tree structures are the commonest in GP. In section 2.3.2, trees structure will be described.

Tree Structure Execution and Memory

Figure 2.1 is a diagram of a tree-based phenome. It has many different symbols that could be executed in any order. But there is a convention for executing tree structures.

The standard convention for tree execution is that it proceeds by repeatedly evaluating the leftmost node for which all inputs are available. This order of execution is referred to as postfix order because the operators appear after the operands. Another convention for execution is called prefix order. It is the precise opposite of postfix order and executes the nodes close to the root of the tree before it executes the terminal nodes. The advantage of prefix ordering is that a tree containing nodes like *IF/THEN* branches can often save execution time by evaluating first whether the *THEN* tree must be evaluated. Applying postfix order to Figure 2.1, the execution order of the nodes is: d -> e -> OR -> a-> b-> c -> + -> MUL -> -.

This same tree structure also constrains the usage of memory on execution. Figure 2.1 uses only local memory during execution. Why? Local memory is built into the tree structure itself. For example, the values of b and c are local to the + node. The values of b and c are not available to any other part of the tree during execution. The same is true for every value in the tree.

2.3.3 Initialization Methods

The first step in actually performing a GP run is to initialize the population. That means creating a variety of program structures for later evolution.

One of the principal parameters of a GP run is the maximum size permitted for a program. For trees in GP, that parameter is expressed as the maximum depth of a tree

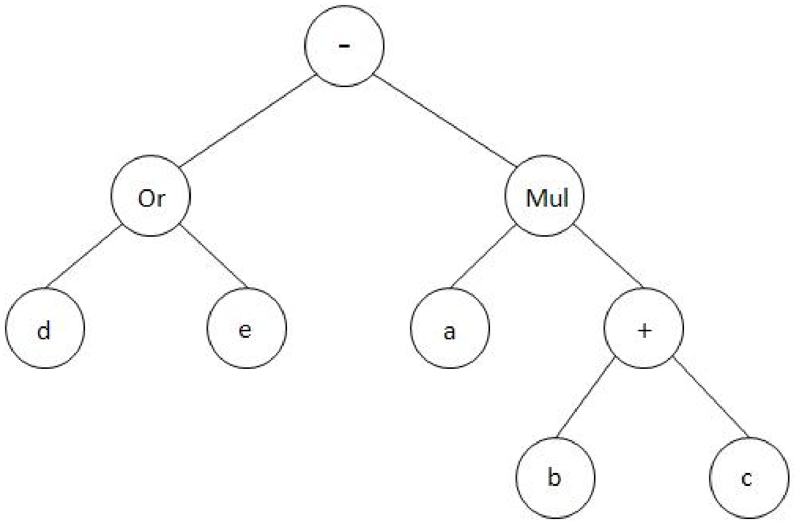


Figure 2.1: A tree structure phenome

or the maximum total number of nodes in the tree. The depth of a node is defined as the minimal number of nodes that must be traversed to get from the root node of the tree to the selected node. The *maximum depth parameter* (MDP) is the largest depth that will be permitted between the root node and the outermost terminals in an individual.

Grow and full method

There are two different methods for initializing tree structures in common use. They are called full and grow [57].

Grow produces trees of irregular shape because nodes are selected randomly from the function and the terminal set throughout the entire tree (except the root node, which uses only the function set). Once a branch contains a terminal node, that branch has ended, even if the maximum depth has not been reached.Because the incidence of choosing terminals is random throughout initialization, trees initialized using grow are likely to be irregular in shape.

Instead of selecting nodes randomly from the function and the terminal set, the full method chooses only functions until a node is at the maximum depth. Then it chooses only terminals. The result is that every branch of the tree goes to the full maximum depth.

The Ramped Half-and-Half Method

Diversity is valuable in GP populations. By itself, the above method could result in a uniform set of structures in the initial population because the routine is the same for all individuals. To prevent this, the "ramped-half-and-half" technique has been devised. It is intended to enhance population diversity of structure from the outset [58]. In trees the technique is like this. Suppose the maximum depth parameter is 6. The population is divided equally among individuals to be initialized with trees having depths 2, 3, 4, 5, and 6. For each depth group, half of the trees are initialized with the full technique and half with the grow technique.

2.3.4 Genetic Operators

An initialized population usually has very low fitness. Evolution proceeds by transforming the initial population by the use of genetic operators. In machine learning terms, these are the search operators. While there are many genetic operators, the three principal GP genetic operators are:

- Crossover
- Mutation

- Reproduction

Crossover Operator

The crossover operator combines the genetic material of two parents by swapping a part of one parent with a part of the other. Tree-based crossover is described graphically in Figure 2.2. The parents are shown in the left half of the figure while the children are shown in the right half. More specifically, tree-based crossover proceeds by the following steps:

- Choose two individuals as parents, based on mating selection policy. The two parents are shown on the left of Figure 2.2.
- Select a random subtree in each parent. The selection of subtrees can be biased so that subtrees constituting terminals are selected with lower probability than other subtrees.
- Swap the selected subtrees between the two parents. The resulting individuals are the children. They are shown on the right in Figure 2.2.

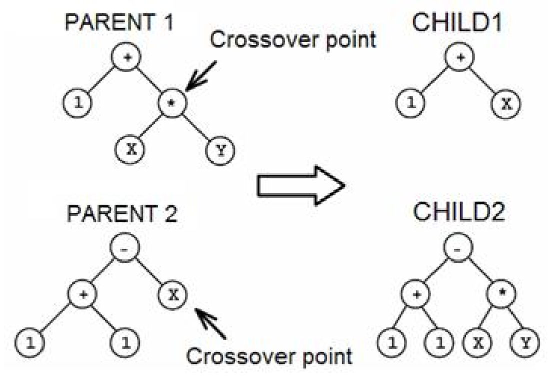


Figure 2.2: Crossover Operator

Mutation Operator

The mutation operation introduces random changes in structures in the population. While there are many different mutation operators, the three principal GP mutation operators are:

- Subtree mutation.
- Swap mutation.
- Point mutation.

The subtree mutation operator works as follows: the mutation operation begins by selecting a point at random within the tree. This mutation point can be an internal (i.e., function) point or an external (i.e., terminal) point of the tree. The mutation operation then removes whatever is currently at the selected point and whatever is below the selected point and inserts a randomly generated subtree at that point. This operation is controlled by a parameter that specifies the maximum size (measured by depth) for the newly created subtree that is to be inserted. This parameter typically has the same value as the parameter for the maximum initial size of trees in the initial random population.

Swap Mutation operates by randomly selecting two subtrees which are then swapped.

Finally, point mutation operates by replacing a randomly selected node with a node of the same *arity*.

Reproduction

The reproduction operator is straightforward. An individual is selected. It is copied, and the copy is placed into the new population.

2.3.5 Fitness and Selection

GP neither is a hill climbing system (which searches only one path through the search space) nor does it conduct an exhaustive search of the space of all possible computer programs. Rather, GP is a type of beam search. The GP population is the beam - the collection of points in the search space from which further search may be conducted.

Of course, GP must choose which members of the population will be subject to genetic operators such as crossover, reproduction, and mutation. In making that choice, GP implements one of the most important parts of its model of organic evolutionary learning, *fitness-based selection*. Fitness-based selection affects both the ordering of the individuals in the beam and the contents of the beam.

GP's evaluation metric is called a fitness function and the manner in which the fitness function affects the selection of individuals for genetic operators may be referred to as the GP selection algorithm. Fitness functions are very problem specific. There are a number of different selection algorithms used in GP.

2.3.5.1 The Fitness Function

Fitness is the measure used by GP during simulated evolution of how well a program has learned to predict the output(s) from the input(s) (that is, the features of the learning domain).

The goal of having a fitness evaluation is to give feedback to the learning algorithm regarding which individuals should have a higher probability of being allowed to multiply and reproduce and which individuals should have a higher probability of being removed from the population.

The fitness function should be designed to give graded and continuous feedback about how well a program performs on the training set.

A *continuous fitness function* is any manner of calculating fitness in which smaller improvements in how well a program has learned the learning domain are related to smaller improvements in the measured fitness of the program, and larger improvements in how well a program has learned the learning domain are related to larger improvements in its measured fitness.

Such continuity is an important property of a fitness function because it allows GP to improve programs iteratively. Two more definitions about fitness function will be useful.

Standardized fitness is a fitness function or a transformed fitness function in which zero is the value assigned to the fittest individual. Standardized fitness has the administrative feature that the best fitness is always the same value (zero), regardless of what problem one is working on.

Normalized fitness is a fitness function or a transformed fitness function where fitness is always between zero and one.

2.3.5.2 The Selection Algorithm

After the quality of an individual has been determined by applying a fitness function, we have to decide whether to apply genetic operators to that individual and whether to keep it in the population or allow it to be replaced. This task is called selection and assigned to a special operator, the selection operator.

There are various different selection operators, and a decision about the method of selection to be applied under particular circumstances is one of the most important decisions to be made in a GP run. Selection is responsible for the speed of evolution and is often cited as the culprit in cases where premature convergence stalls the success of an evolutionary algorithm.

Selection in general is a consequence of competition between individuals in a population. This competition results from an overproduction of individuals which can withstand the competition to varying degrees. The degree to which they can withstand the competition is regulated by the selection pressure, which depends on the ratio of offspring to individuals in the population.

The three principal selection operators are:

- Fitness-Proportional Selection.
- Ranking Selection.
- Tournament Selection.

Fitness-Proportional Selection.

Fitness-proportional selection is employed in a GA scenario for generational selection and specifies probabilities for individuals to be given a chance to pass offspring into the next generation. An individual i is given a probability of

$$p_i = f_i \sum_j f_j$$

for being able to pass on traits. Depending on the variation operator used, this might result (i) in a copy of that individual, or (ii) in a mutated copy, or (iii) in case two individuals have been selected in the way mentioned, two offspring with mixed traits being passed into the next generation.

Ranking Selection.

Ranking selection [43] [112] is based on the fitness order, into which the individuals can be sorted. The selection probability is assigned to individuals as a function of their rank in the population. Mainly, linear and exponential ranking are used. For linear ranking, the probability is a linear function of the rank:

$$\frac{1}{N}\left[p^- + (p^+ - p^-)\frac{i-1}{N-1}\right]$$

where p^-/N is the probability of the worst individual being selected, and p^+/N the probability of the best individual being selected, and

$$p^- + p^+ = 2$$

should hold in order for the population size to stay constant.

For exponential ranking, the probability can be computed using a selection bias constant c,

$$p_i = \frac{c-1}{c^{N-1}} c^N - i$$

with $0 < c < l$.

Tournament Selection.

Tournament selection is not based on competition within the full generation but in a subset of the population. A number of individuals, called the tournament size, is selected randomly, and a selective competition takes place. The traits of the better individuals in the tournament are then allowed to replace those of the worse individuals. In the smallest possible tournament, two individuals compete. The better of the two is allowed to reproduce with mutation. The result of that reproduction is returned to the population, replacing the loser of the tournament.

The tournament size allows researchers to adjust selection pressure. A small tournament size causes a low selection pressure, and a large tournament size causes high pressure.

Tournament selection has recently become a mainstream method for selection, mainly because it does not require a centralized fitness comparison between all individuals. This not only accelerates evolution considerably, but also provides an easy way to parallelize the algorithm.

2.3.6 The Basic GP Algorithm

It is now possible to assemble all of the individual elements (functions, terminals, fitness-based selection, genetic operators, variable length programs, and population initialization) into an overall algorithm for a basic GP run. There are two ways to conduct a GP run, a generational approach and a steady-state approach. In generational GP, an entire new generation is created from the old generation in one cycle. The new generation replaces the old generation and the cycle continues. In steady-state GP, there are no generations. We will present an algorithm for each approach.

First, however, we will review the preparatory steps for making a GP run. Then we will discuss the two basic ways to approach the GP run algorithm itself.

2.3.6.1 Summary of Preparatory Steps.

Here are the preliminary steps in a GP run, which has been described in detail in this chapter.

1. Define the terminal set.
2. Define the function set.
3. Define the fitness function.
4. Define parameters such as population size, maximum individual size, crossover probability, selection method, and termination criterion (e.g., maximum number of generations).

Once these steps are completed, the GP run can commence. How it proceeds depends on whether it is generational or steady state.

2.3.6.2 Generational GP Algorithm.

Traditionally, GP uses a *generational* evolutionary algorithm. In generational GP, there exist well-defined and distinct generations. Each generation is represented by a complete population of individuals. The newer population is created from and then replaces the older population. The execution cycle of the generational GP algorithm includes the following steps:

1. Initialize the population.
2. Evaluate the individual programs in the existing population. Assign a numerical rating or fitness to each individual.
3. Until the new population is fully populated, repeat the following steps:
    - Select an individual or individuals in the population using the selection algorithm.
    - Perform genetic operations on the selected individual or individuals.
    - Insert the result of the genetic operations into the new population.

    4. If the termination criterion is fulfilled, then continue. Otherwise, replace the existing population with the new population and repeat steps 2-4. 5. Present the best individual in the population as the output from the algorithm.

2.3.6.3 Steady-State GP Algorithm.

The steady-state or tournament selection model is the principal alternative to generational GP. In this approach there are no fixed generation intervals. Instead, there is a continuous flow of individuals meeting, mating, and producing offspring. The offspring replace existing individuals in the same population.

Here is an example of a basic GP algorithm using the steady-state method.

1. Initialize the population.
2. Randomly choose a subset of the population to take part in the tournament (the competitors).
3. Evaluate the fitness value of each competitor in the tournament.
4. Select the winner or winners from the competitors in the tournament using the selection algorithm.
5. Apply genetic operators to the winner or winners of the tournament.
6. Replace the losers in the tournament with the results of the application of the genetic operators to the winners of the tournament.
7. Repeat steps 2-7 until the termination criterion is met.
8. Choose the best individual in the population as the output from the algorithm.

CHAPTER 3

Open Issues in Genetic Programming

Contents

3.1 Bloat

Bloat is one of the most important problem that affects GP. According to [9] bloat (or code bloat) is defined as code growth without a significant improvement in terms of fitness. In recent years several theories have been developed to trying to explain the origin of this phenomenon. Methods to limit or prevent the bloat have also been proposed. In this section a review of the state of the art methods and theories concerning is proposed.

3.1.1 Bloat Theories

As stated before, several theories have been developed to explain the occurrence of bloat. In 1994 in [3] author noted the appearance of redundancy in evolved genetic programs. Several solutions evolved by Koza [57] contain semantically extraneous components. By extraneous, he meant that if those code segments were removed from the solution, this would not alter the result produced by the solution. According to [3] the ubiquity of redundant constructs suggests a more basic truth about their nature. There is an advantage from the viewpoint of the evolving genetic program for syntactically redundant constructions that are semantically transparent. Because the redundant functions do not effect the semantics of the sub-expression, if a crossover operation is performed such that it splits the redundant components, the semantics of the removed subtree are still preserved. These redundant functions serve as introns for the evolving subexpression giving the blind crossover operator a higher probability of capturing the complete subexpression intact. An intron is a portion of a chromosome that is never expressed and provides spacing between the genes.

Hitchhiking defined by Tackett in [98] is a second theory that focuses on the presence of introns. Tackett hypothesized that GP bloat was caused by blocks of code in GP individuals that, while they had little merit on their own, happened to be in close proximity to high fitness blocks of code. Tackett referred to this phenomenon as hitchhiking.

Protection from deletion [1] [14], [9], states that introns are required to protect fit programs from the mostly destructive crossover operation. In [9] authors pointed out that while introns do not affect the fitness of the individual, they do affect the likelihood that the individual's descendants will survive. They referred to this new concept of fitness, which includes the survivability of an individual's offspring, as *effective fitness*. The better the parent can protect its children from being the results of destructive genetic operators, the better the effective fitness of the parent. Introns help parents do that. The concept of effective fitness will is clearly a function of at least two factors:

- The fitness of the parent. The fitter the parent, the more likely it is to be chosen for reproduction.
- The likelihood that genetic operators will affect the fitness of the parent's children.

Introns emerge as a result of competition among parents with respect to the second item. Hence successful individuals are able to survive generation to generation due to the fact that potentially destructive crossovers take place in intron areas of the program, and as such does not degrade their fitness for selection.

Removal bias [97] occurs when inviable subtrees near the leaves of programs more frequently are replaced by larger subtrees with no effect on fitness. As most crossover is destructive, these programs survive due to their fitness being unchanged, whereas, more programs that had crossovers in areas of active code are culled at selection. In [97] authors also pointed out that at least two factors contribute to code growth in GP: the evolutionary advantage of having protective, inviable code and a bias in favor of the removal of small subtrees. Of these two causes removal bias contributes somewhat less to the overall growth, although the amount of growth it produces is still significant. They presented these two forms of growth as being independent and they showed that each can act in the absence of the other. However, it also seems likely that in a normal GP they act in concert. For example, removal bias creates inviable code which is preserved because of its protective value.

While hitchhiking, protection from deletion and removal bias require the presence of introns, in [63] authors concentrate their attention on the fitness function. The fitness causes bloat theory basically states that with a variable-length representation there are many different ways to represent the same program, long and short, and a static evaluation function will attribute the same fitness to all, as long as their behavior is the same. Given that crossover is a destructive process, when more fit programs become hard to find, then programs of equal fitness are favored. Because there are many more longer ways to represent a program than shorter ways, a natural drift towards longer solutions occurs, causing bloat.

Another explanation for bloat in tree-based GP was advanced by Luke in [66]. Luke has observed that when a genetic operator modifies a parent to create an offspring, there is a link between the depth of the modified node and its effect on the fitness of the offspring when compared to the parent. Given that most crossover is destructive, programs that experience swaps near the leaves of the trees suffer less decrease in their fitness compared to trees that include a crossover near the root. As a result, more code gets added near the leaves of the tree and longer trees are favoured in terms of survival when the fitness function and selection are applied. Moreover, the deeper the modification point, the smaller the branch that is removed, thus creating a removal bias. This may be regarded as a generalization of the original removal bias theory [97].

The crossover bias theory [31] is the most recent theory concerning bloat. It explains code growth in tree-based GP by the effect that standard subtree crossover has on the distribution of tree sizes in the population. In particular it states that shorter programs will be more frequently sampled as a result of crossover. Because very small individuals are generally unfit, selection tends to reject them in favor of the larger individuals, causing an increase in mean tree size. It is the proliferation of these small unfit individuals, perpetuated by crossover, that ultimately causes bloat. Dignum and Poli [31]support the crossover bias theory providing strong theoretical evidence showing how the distribution of program sizes follows a Lagrange distribution of the second kind [51] [52] [79].

One of the key issues behind bloat appears to be the selection mechanism. As shown by Tackett [98] random selection results in no bloat. The moment the fitness function is applied, underlying bias becomes present and have numerous effects on GP performance and program size.

3.1.2 Method to control bloat

Although several theories have been defined to explain the causes of the appearance of bloat in GP there is not a widely accepted theory. Because of this fact GP practitioners have developed several techniques to reduce and control bloat. In [68] several bloat control methods are discussed.

The first method to control bloat has been proposed by Koza in [57] and the idea is very simple. A depth limit of 17 is fixed, and if the depth of a program is greater than 17 after a crossover, the parent program was used instead of the child program.

The Tarpeian method [81] introduces a parameter which represents a probability of assigning a very poor fitness value to a large program without assessing the fitness of the program. Such individuals are not evaluated further, and such a low fitness dramatically reduced any chance of an individual being selected for breeding (it was still possible to be selected if the tournament selection procedure happened to select only individuals of this type, but the chances of this happening were slim). The crucial feature of the Tarpeian method is that it reduces the number of evaluations necessary. This feature makes Tarpeian different from the other bloat control methods, which all evaluate every individual generated.

Parsimony pressure is a general family of methods which consider size as part of the selection process. Historically such methods work by computing the fitness g of an individual as a function of the individual's raw fitness f and its size s. We refer to such methods as parametric parsimony pressure, as they result in a fitness model which uses the exact values of f and s as parameters in a statistical model of selection.

The most widely-used approach to parametric parsimony pressure is to treat the individual's size as a linear factor in fitness, that is, $g = xf + ys$, where x and y are parameter settings.

The main difficulty with this method is that without scaling, one element of the equation $g = xf + ys$ will overwhelm the other element creating bias in the fitness function. Consequently, the fitness and size components are scaled in an effort to prevent this bias to occur. The choice of coefficients in order to balance fitness and program size is difficult, however, in [85] authors show how to apply Price's covariance theorem to dynamically set and optimize the parsimony coefficients.

Variants of the canonical linear parametric parsimony pressure method are proposed in [22], [99], [118] and [55]. In [76] authors use a non-linear parametric parsimony pressure that increases the penalty proportionally to the fitness.

Research has shown that beyond a certain minimum program length the distributions of program functionality and fitness converge to a limit [64]. Before that limit, however, there may be program-length classes with a higher or lower average fitness than that achieved beyond the limit. Ideally, therefore, GP search should be limited to program lengths that are within the limit and that can achieve optimum fitness. This has the dual benefits of providing the simplest/smallest solutions and preventing GP bloat thus shortening run times.

Operator equalisation is a bloat control technique proposed in [33] that allows accurate control of the program length distribution during a GP run. By filtering which individuals are allowed in the population, it can easily bias the search towards smaller or larger programs. The technique controls the distribution of sizes inside the population by probabilistically accepting each newly created individual based on its size. The probabilities of acceptance are calculated considering a predetermined target distribution. A maximum allowed program length, and the desired number of length classes, or bins, are used to calculate the width of the bins. This technique achieved promising results with different predetermined target length distributions, using a conservative program length limit.

In [93] authors improve operator equalisation by giving it the ability to automatically determine and follow the ideal length distribution for each stage of the run, unconstrained by a fixed maximum limit. Results show that in most cases the new technique performs a more efficient search and effectively reduces bloat, by achieving better fitness and/or using smaller programs.

3.2 Overfitting and Complexity

The main goal of a GP system is to find a generally valid set of rules or patterns from a set of observations (training set). Two different concept may occur when the set of observation is used for the learning process: generalization and overfitting.

According to [9] generalization occurs when learning that occurs on the training data remains, to some extent, valid on test data. In other words, generalization is a problem of drawing a sample from a population and making predictions about other samples from the same population. The noise from one sample to the next is probably different, perhaps a lot different. In other words, sampling error can be a big problem.

Another equally difficult problem is overfitting. Overfitting affects several machine learning techniques and occurs when a model begins to memorize the training data rather than learning to generalize from the model. According to [74] given a hypothesis space H, a hypothesis $h \in H$ is said to overfit the training data if there exists some alternative hypothesis $h' \in H$ such that h has smaller error than h' over the training examples, but h' has a smaller error than h over the entire distribution of instances.

Considering GP, the ideal learning process would learn the true relationship between the inputs and the outputs and would ignore the noise. But GP is guided by its fitness function to lower the error on the training data. After GP has modeled the true relationship, GP individuals can and do continue to improve their fitness by learning the noise unique to the sample that comprises the training data (that is, by overfitting the training data) [9].

One of the approaches used to detect if overfitting occurs is to use a validation set. In particular all the solutions in the population are evaluated on the validation set additionally to the training set. Instead of returning the best solution on the training set, the best k solutions on the training set are selected and, among these solutions, the solution with the best fitness on the validation set is returned as the final result [40], [113]. This approach can also be extended to store a set of Pareto optimal solutions regarding validation fitness and model size in an archive [96], [89].

The effectiveness of a validation method strongly depends on the observations contained in the validation partition. Both the training and validation partition should contain a representative sample of all possible observations to make it possible to create a model that is also applicable to new observations. Thus the choice of training and validation partitions is crucial for any data-based modeling approach.

Other statistical learning approaches to counteract overfitting tune the algorithm parameters in iterative steps [46]. The objective is to find a parameter settings which result in a model that generalizes well using an estimator for the expected generalization error of the model. The expected generalization error can be estimated through a cross-validation.

An important topic that has been discussed is the relation between bloat and overfitting. It has been recently observed that overfitting can occur in absence of bloat, and vice versa. Thus, it has been suggested that overfitting and bloat are two separate phenomena in GP [107], [95]. This suggests that overfitting and bloat should be controlled also by separate mechanisms.

Nevertheless, some technique to counteract overfitting are based on the assumption that bloat and overfitting are related phenomena. For example in [60] authors use the covariant parsimony pressure method [80], which was initially proposed for bloat control, to reduce overfitting. In their contribution authors propose an overfitting detection criterion for symbolic regression. This criterion is used in combination with covariant parsimony pressure to control the average program length. Parsimony pressure is applied in the overfitting state to gradually reduce model length. As long as no overfitting is detected the program length is allowed to grow.

In section 4.2 a measure to quantify overfitting during a GP run is proposed.

It is often stated that simpler solutions will be more robust and will generalize better than complex ones, with the latter being more likely to overfit the training data [74], [107].

An immediate question concerns how to measure program complexity. In section 4.3 a measure of functional complexity has been proposed. The measure is called Slope-based Functional Complexity (SFC). SFC in inspired by the concept of curvature, and represents the first measure of complexity that is explicitly intended to be an indicator of program overfitting. While SFC is based on a reasonable intuition, the methodological approach it requires can become complicated for multidimensional problems.

In [101] authors proposed a measure of complexity based on the concept of Hölderian regularity [100], and they call it Regularity-based Functional Complexity (RFC). This measure captures the underlying justification of SFC overcoming some of its practical difficulties.

Both the cited complexity measures show almost no correlation with program overfitting (at least for the considered test problems).

The study of the relation between program complexity and overfitting deserves further analysis. In fact, the measures cited in this section only represent a first attempt to measure complexity and to use this value as an indicator of GP overfitting. Hence a more comprehensive evaluation of these, and possibly other, measures of complexity is fundamental.

CHAPTER 4

Measures of Overfitting, Bloat and Functional Complexity

Contents

The goal of this chapter is to define measures to quantify the amount of bloat, the amount of overfitting and the functional complexity of the solutions in a GP system. We introduce these measures pointing out their main advantages and drawbacks and trying to propose ways to improve them. These measures, or their improved versions, should be used to study and better understand the relationship between bloat, overfitting and functional complexity of solutions in the future.

4.1 Bloat Measure

The most accepted definition of bloat is “program growth without (significant) return in terms of fitness” (definition taken from [82], page 101). This definition is clearly a very fuzzy one and in many bibliographic references GP problems are analyzed in terms of “presence” or “absence” of bloat, without any numerical quantification. This is the case, among many other references, of [95] where the authors state “Standard GP bloats” and “Dynamic Operator does not bloat”, or of [82] where the authors use informal expressions like “bloat is more marked” (page 41), “reduce bloat” (page 78) or “bloat can ... happen” (page 106). We believe that these expressions are really too informal to permit a deep study of bloat and we introduce a simple measure to quantify the “amount of bloat of a GP run” with a single number. To the best of our knowledge, such a measure has never been defined so far.

According to the proposed measure, in case of minimization problems (i.e. problems were a small fitness value is better than a large one) the “amount of bloat” at generation g is defined as:

$$bloat(g) = \frac{(\overline{\delta}(g) - \overline{\delta}(0))/\overline{\delta}(0)}{(\overline{f}(0) - \overline{f}(g))/\overline{f}(0)} \tag{4.1}$$

where $\overline{\delta}(g)$ is the average program length[1] in the population at generation g and $\overline{f}(g)$ is the average fitness (calculated using training data) in the population at generation g.

We can informally say that $bloat(g)$ is an expression of the relationship between the average length growth and the average fitness improvement up to generation g compared to the respective values at generation zero. If we assume that at generation zero

[1]Following [57], with the expression *length of a program* we mean the total number of nodes in its tree-like representation.

we have no bloat[2], then this is a very intuitive formalization of the definition of bloat. Furthermore, if we assume that, for a given $g > 0$, $\overline{\delta}(g)$ is larger than $\overline{\delta}(0)$ and (for minimization problems) $\overline{f}(g)$ is smaller than $\overline{f}(0)$, which is what normally happens in an usual GP run, then we have that $bloat(g)$ often assumes positive values[3]. Finally, if g is large, then we usually have that small improvements in fitness correspond to large improvements in length. For this reason, we have normalized the numerator and denominator of $bloat(g)$ by $\overline{\delta}(0)$ and $\overline{f}(0)$. This normalization mitigates the effect of the two different orders of magnitude of these improvements.

4.2 Overfitting Measure

Following [74] (page 67), overfitting can be defined as follows: "Given a hypothesis space H, a hypothesis $h \in H$ is said to *overfit* the training data if there exists some alternative hypothesis $h' \in H$, such that h has smaller error than h' over the training samples, but h' has smaller error than h over the entire distribution of instances". Under this perspective, defining a measure for overfitting is clearly a hard task, given that it requires the exploration of all the hypothesis space H, looking for such a hypothesis h'. Nevertheless, we believe that a good indication of the "amount of overfitting" of a GP system can be given by the relationship between fitness on the training set (training

[2]If we assume that the definition of bloat is "program *growth* without (significant) return in terms of fitness", then it comes natural to assert that at generation zero programs did not *grow* yet, and thus there can be no bloat. Furthermore, it is worth pointing out that at generation zero, i.e. after the initialization phase, usually programs have a strong size limitation given by the initialization method used. In this work we use Ramped Half and Half initialization [57] with a depth limit equal to 6.

[3]This has a known exception for symbolic regression problems, where the average population size usually shrinks in the very first generations, before increasing progressively in the rest of the run. For this reason, we did not force this constraint to necessarily hold, thus assuming that $bloat(g)$ can also have negative values.

fitness from now on) and the fitness on the test set (test fitness from now on). Thus, at a given generation `g`, we quantify overfitting by the `overfit(g)` value calculated by the pseudocode in Figure 4.1, where: `btp` stands for “best test point” and it represents the best test fitness reached until the current generation, excluding the generations (usually at the beginning of the run) where the best individual on the training set has a better test than training fitness; `tbtp` stands for “training at best test point”, i.e. the training fitness of the individual that has test fitness equal to `btp`; `training_fit(g)` is a function that returns the best training fitness in the population at generation `g`; `test_fit(g)` returns the test fitness of the best individual on the training set at generation `g`.

```
overfit(0) = 0
btp = test_fit(0)
tbtp = training_fit(0)

for each generation g>0
    if (training_fit(g) > test_fit(g))
        overfit(g) = 0
    else
        if (test_fit(g) < btp)
            overfit(g) = 0
            btp = test_fit(g)
            tbtp = training_fit(g)
        else
            overfit(g) =
              |training_fit(g)-test_fit(g)| - |tbtp-btp|
        endif
    endif
endfor
```

Figure 4.1: Pseudocode for calculating the overfitting measure in case of minimization problems.

In synthesis, the ideas that inspire this algorithm are: if, at a given generation `g`, test fitness is better than training fitness, then there is no overfitting (`overfit(g)=0`); if test fitness is better than the best test point, then there is no overfitting (`overfit(g)=0`); otherwise overfitting is quantified by the difference of the distance between training and test fitness at generation `g` and the distance between training and test fitness at the generation where the best test point has been found. Given that we use elitism in our GP runs, training fitness is constantly improving, or in the worst case it stays constant. For this reason: training_fit(g)-test_fit(g) $\geq$ tbtp-btp. Thus we have that, for all generations `g`, `overfit(g)` ≥ 0, but this may not be the case in the absence of elitism.

We are aware that this definition has the clear and strong limitation that it dramatically depends on how training and test data are chosen and we believe that a more sophisticated definition, where training and test data are alternated in a crossvalidation-like way is suitable. Nevertheless, this work represents a first and preliminary study of this measure and thus we prefer to keep things simple and we adopt the algorithm in Figure 4.1.

We also point out that using the same idea as in equation (4.1) for measuring the relationship between training and test fitness (i.e. comparing the values at the current generation with the corresponding ones at generation zero) is definitely unsuitable. In fact, at generation zero the population has been created by the (typically random) initialization algorithm and the solutions in the population have not undergone any evolution yet. For this reason, there is typically no reason why training fitness should be better than test fitness at generation zero. On the other hand, these two values are typically quite similar and being the test set often smaller than the training set, it is even more likely that the average training fitness is worse than the average test fitness at generation zero. For this reason, using a formula like equation (4.1) to quantify overfitting would produce a measure that is more affected by the values of training and test fitness at generation zero than by the course of the learning process. As a consequence of the fact that training and test fitness values are very similar to each other at generation zero (or test fitness is even better than training fitness), a measure like equation (4.1) would very often produce very high overfitting values, even when it is clearly not the case.

4.3 Multi-Slope Complexity Measure

Previous and Related Work

The relationship between code growth and functional complexity in GP has been only lightly investigated so far. In [111], the authors define two measures of complexity for the GP individuals: a genotypic measure and a phenotypic one. While the genotypic measure is related to counting the number of nodes of a tree and its subtrees, the phenotypic one, called *order of nonlinearity*, is related to functional complexity and consists in calculating the degree of the Chebyshev polynomial approximation of the function. The authors then use these two measures as further criteria (besides fitness) in a multi-objective system based on Pareto optimization, and they show that this system is able to counteract bloat and overfitting.

In this work, complexity is used differently: the proposed measure will not be used as an optimization criterium, but just reported for the current best individual (in the training set) along the evolution, in order to discover its relationship with bloat and/or overfitting. Furthermore, the goal is not to find the approximating polynomial of the function, but rather to work directly on the function itself. For this reason, a measure inspired by the concept of curvature as an indicator of the functional complexity is proposed.

Curvature

Informally, the curvature [20] of a function is the amount by which its geometric representation deviates from being *flat*, or *straight*. For a plain curve in 3 dimensions, given parametrically as $c(t) = (x(t), y(t))$, the curvature is defined as [20]:

$$k = \frac{x'y'' - y'x''}{(x'^2 + y'^2)^{3/2}}$$

Suppose to solve a tridimensional symbolic regression problem using GP, and suppose to report the average curvature of the individuals in the population at each generation. This task is computationally very hard. In fact it implies the calculation of simple and second derivatives of the function along two dimensions. In case the coding function of a GP individual is not derivable, the exact calculus may even be impossible, and an approximation would require the previously mentioned [111] calculation of the approximating polynomial of that function. Furthermore, the dimension of the feature space of usual symbolic regression problems is by far larger than three. The theory of generalized curvature measures to multidimensional spaces has a long history, that has recently been gathered in [75].

On the other hand, in this work an indicator of functional complexity that is inspired by the intuition of the concept of curvature is defined. This indicator has the advantage of being calculable in polynomial time with the number of fitness cases. The intuition of this indicator is very simple, even though its formal definition may seem a bit complicated. For this reason, this measure is firstly presented from an intuitive viewpoint, then a simple example of its calculation is proposed and finally its formal definition is given.

Intuition

Consider the three simple bidimensional functions in Figure 4.2.

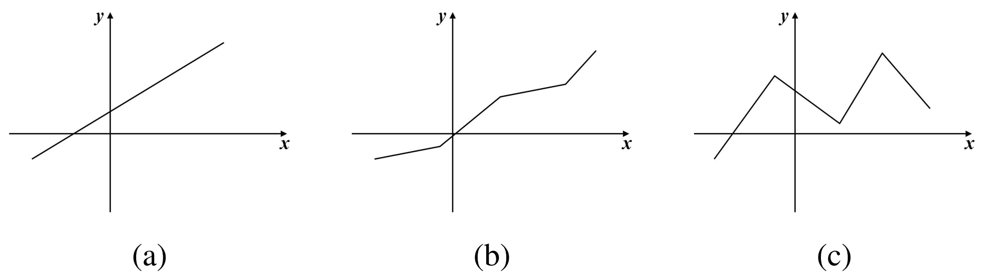


Figure 4.2: The graphical representation of three simple functions used to explain the proposed complexity measure from an intuitive viewpoint. According to the proposed measure, the function represented in plot (a) must be less complex than the function represented in plot (b) and the function represented in plot (b) must be less complex than the function represented in plot (c) (see the text for further explanation).

The objective is to define a measure to quantify the fact that the function represented in Figure 4.2(c) is more complex than the one represented in Figure 4.2(b), which is more complex than the one represented in Figure 4.2(a). In fact, all the three functions are polylines, but in Figure 4.2(a) all the segments forming the polyline have the same slope, while in Figure 4.2(b) the segments have different slopes, even though all the slopes have the same sign; finally, segments in Figure 4.2(c) have different slopes of different signs. In other words, what is wanted is to express the complexity of a function by counting the number of different slopes and by assigning a higher weight to inversions in the slope sign. It is worth pointing out that such a measure has formally absolutely no relationship with the curvature measure. It is only inspired by an informal intuition of the concept of curvature.

Considering bidimensional regression problems, one can imagine the graphical representation of a GP individual as a polyline, by plotting the points that have fitness cases on the abscissas and the corresponding values of the function coding the GP individual as ordinates and joining those points by segments. Thus, in case of bidimensional problems, all that must be done to calculate the measure is to sort all fitness cases (from the smaller value to the larger one) and to consider the values assumed by the GP individual on those fitness cases. Then the slope of each segment joining these points is calculated. Let $(s_1, s_2, s_3, s_4, s_5, \ldots)$ be those slopes. The measure is simply calculated as: s_1 - s_2 + s_2 - s_3 + s_3 - s_4 + s_4 - s_5 + ... In this way, if the slopes are all identical, the value of the measure is zero (minimal possible complexity) and if the signs of the slopes of all the consecutive segments change, the contribution of each segment is maximal (because all slopes' absolute values are summed). On the other hand, if two consecutive segments have different slopes with the same sign, their contribution is the subtraction of their respective absolute values (i.e. a contribution larger than zero, but smaller than in the case where the slope sign changes).

In case of multidimensional spaces of features, the projection of the function coding the GP individual on each single dimension is considered. The same calculus as above is executed for each dimension separately and then the average is calculated.

Example

Let the fitness cases of a bidimensional regression problem be $(0, \pi/2, 3/2\pi, 2\pi)$ and let $g(x) = sin(x)$ be a GP individual. Then the points of the polyline are:

$\{(0,0),(\pi/2,1),(3/2\pi,-1),(2\pi,0)\}$. In this case, the slope of the segment that joins the points $(0,0)$ and $(\pi/2,1)$, the one of the segment that joins the points $(\pi/2,1)$ and $(3/2\pi,-1)$ and the one of the segment that joins the points $(3/2\pi,-1)$ and $(2\pi,0)$ must be calculated. Let s_1, s_2 and s_3 be these three slopes respectively. Hence: $s_1 = 1/(\pi/2) = 2/\pi$, $s_2 = -2/(3/2\pi - \pi/2) = -2/\pi$ and $s_3 = 1/(2\pi - 3/2\pi) = 2/\pi$. The value of the proposed measure is: s_1 - s_2 $+$ s_2 - s_3 $= 8/\pi$.

Formal Definition

Let $\mathbf{X} = (\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n)$ be the vector of fitness cases of a regression problem, where for each $i = 1,2,...,n$: $\mathbf{x}_i = (x_{i1}, x_{i2}, ..., x_{im})$ is an m-dimensional vector of floating point numbers (i.e. the feature space is m-dimensional). Let $g : \mathbb{R}^m \to \mathbb{R}$ be the function coding a GP individual and let $g_i = g(\mathbf{x}_i)$ be the value assumed by g on the i^{th} fitness case. Given a $j = 1,2,...,m$, let $\mathbf{p}_j = (x_{1j}, x_{2j}, ..., x_{nj})$ be the vector containing all the values of feature j in $\mathbf{X}$. Now let $\mathbf{q}_j = (y_{1j}, y_{2j}, ..., y_{nj})$ be a vector that contains the same values as $\mathbf{p}_j$, except that they are sorted (i.e. $\mathbf{q}_j$ is a permutation of $\mathbf{p}_j$, such that the values in $\mathbf{q}_j$ are sorted). Let $\phi : \{1,2,...,n\} \to \{1,2,...,n\}$ be a function that, applied to an index in $\mathbf{q}_j$ returns the position of the corresponding element in $\mathbf{p}_j$. In other words, for all $k,h = 1,2,...,n$: $y_{kj} = x_{hj}$ if and only if $\phi(k) = h$. Then it is possible to define:

$$pc_j = \begin{cases} \displaystyle\sum_{i=1}^{n-2} \text{abs} \frac{g_{\phi(i+1)} - g_{\phi(i)}}{y_{(i+1)j} - y_{ij}} - \frac{g_{\phi(i+2)} - g_{\phi(i+1)}}{y_{(i+2)j} - y_{(i+1)j}}, & \text{if } n \geq 3 \\ 0, & \text{otherwise} \end{cases}$$

where pc_j stands for *partial complexity* on the j^{th} dimension. Finally, the complexity measure is defined as the average of all the partial complexities on all the dimensions of the feature space, i.e.:

$$complexity = \left(\sum_{j=1}^{m} pc_j\right)/m$$

Experimental Setting

A total of 30 runs were performed both with Standard GP (StdGP) and Dynamic Operator Equalization (DynOpEq) [93]. A brief description of DynOpEq is proposed in chapter 5. All the runs used populations of 500 individuals allowed to evolve for 100 generations. Tree initialization was performed with the Ramped Half-and-Half method [57] with a maximum initial depth of 6. The function set contained the four binary operators $+$, $-$, $\times$, and $/$, protected as in [57]. The terminal set contained one floating point variable for the F_7 test function, 241 floating point variables for the %F dataset and 626 floating point variables for the LD50 dataset. In all these test problems no random constants were added to the terminal set. Because the cardinalities of the function and terminal sets were so different, a balanced choice between functions and terminals has been imposed when selecting a random node. Fitness was calculated as the root mean squared error between outputs and targets. Selection for reproduction used Lexicographic Parsimony Pressure [67] tournaments of size 10. Very similar to a regular tournament, the lexicographic tournament chooses smaller individuals when their fitness is the same. The reproduction (replication) rate was 0.1, meaning that each selected parent has a 10% chance of being copied to the next generation instead of being engaged in breeding. Standard tree mutation and standard crossover (with uniform selection of crossover and mutation points) were used with probabilities of 0.1 and 0.9, respectively. The new random branch created for mutation has maximum depth equal to 6. Selection for survival used elitism (i.e. unchanged copy of the best individual in the next population). Regarding the parameters specific to each technique, StdGP used a fixed maximum depth of 17, and DynOpEq used a bin width of 1.

Experimental Results

Experimental results are shown in Figures 4.3, 4.4 and 4.5 for the F_7 test function, and the %F and LD50 test problems, respectively.

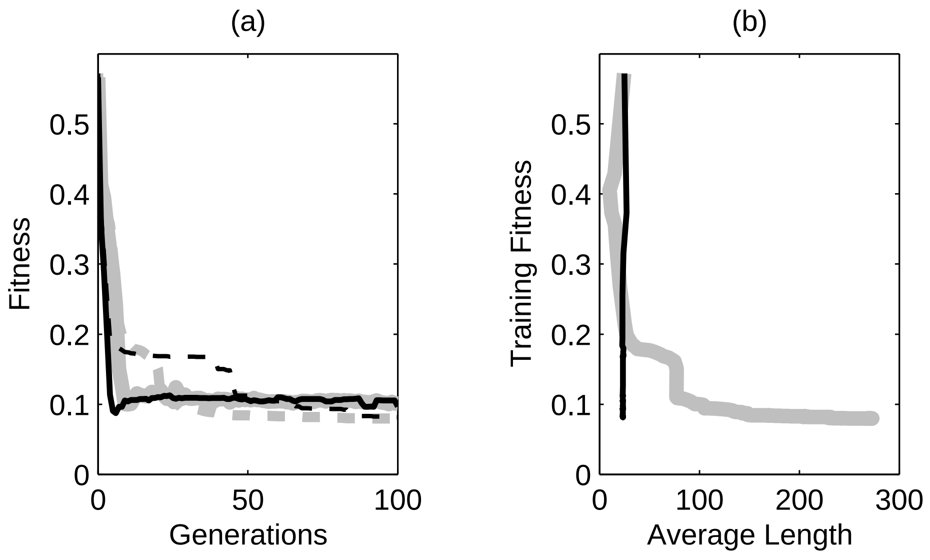


Figure 4.3: Test Function: $F_7(x) = log(x)$. Plot (a): best training and test fitness vs. generations. Plot (b): best training fitness vs. average program length. Plot (c): bloat measure described in Section 4.1 vs. generations. Plot (d): overfitting measure described in Section 4.2 vs. generations. Plot (e): complexity measure described in Section 4.3 vs. generations. In all cases, median values over 30 independent runs are reported. Legend: grey thick lines = StdGP; black thin lines = DynOpEq. (In Plot (a): solid lines = test fitness; dashed lines = training fitness).

In all these figures plots (a) report the best training fitness and the test fitness of

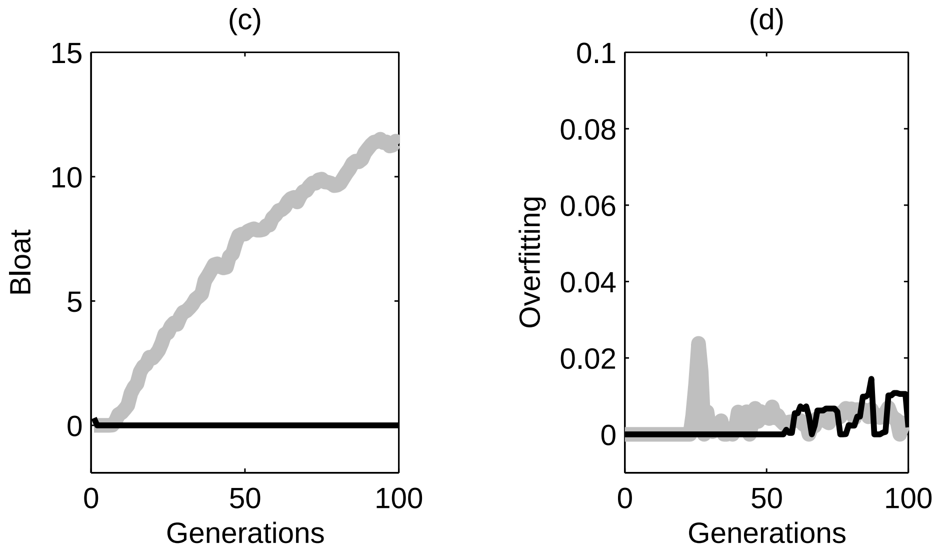


Figure 4.3: Test Function: $F_7(x) = log(x)$. Plot (a): best training and test fitness vs. generations. Plot (b): best training fitness vs. average program length. Plot (c): bloat measure described in Section 4.1 vs. generations. Plot (d): overfitting measure described in Section 4.2 vs. generations. Plot (e): complexity measure described in Section 4.3 vs. generations. In all cases, median values over 30 independent runs are reported. Legend: grey thick lines = StdGP; black thin lines = DynOpEq. (In Plot (a): solid lines = test fitness; dashed lines = training fitness).

the best individual on the training set vs. generations. These plots should give an intuition of the generalization ability of the solutions found by the different GP models and their amount of overfitting. In plots (b) some unconventional curves are reported: best training fitness vs. average program length. Depending on how fast the fitness improves with the increase of program length, the lines in the plot may point downward (south), or they may point to the right (east). Lines pointing south represent a rapidly improving fitness with little or no code growth. Lines pointing east represent a slowly improving fitness with strong code growth. These plots should help us to make inferences about bloat: for a method that does not bloat, but at the same time is able

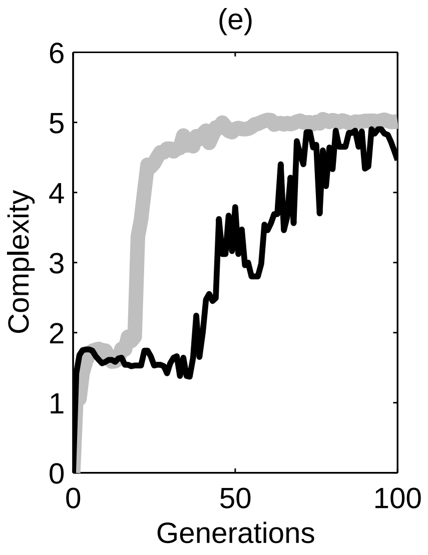


Figure 4.3: Test Function: $F_7(x) = log(x)$. Plot (a): best training and test fitness vs. generations. Plot (b): best training fitness vs. average program length. Plot (c): bloat measure described in Section 4.1 vs. generations. Plot (d): overfitting measure described in Section 4.2 vs. generations. Plot (e): complexity measure described in Section 4.3 vs. generations. In all cases, median values over 30 independent runs are reported. Legend: grey thick lines = StdGP; black thin lines = DynOpEq. (In Plot (a): solid lines = test fitness; dashed lines = training fitness).

to improve fitness, these lines should point as south as possible. Plots (c) report the bloat measure described in Section 4.1 vs. generations. Plots (d) report the overfitting measure described in Section 4.2 vs. generations[4]. Finally plots (e) report the complexity measure described in Section 4.3 vs. generations. In all cases, median values over the 30 runs are reported. When comparing curves, the term "correlation" is used only from a visual, not statistical, point of view.

Let begin by analyzing the results of the F_7 test function (Figure 4.3). Plot 4.3(a) shows that DynOpEq does not overfit in the first part of the run, because test fitness

[4]For an easier comparison of overfitting between the three problems, each y axis was drawn so that its maximum value is roughly one fifth of the maximum y axis value of the respective plot (a).

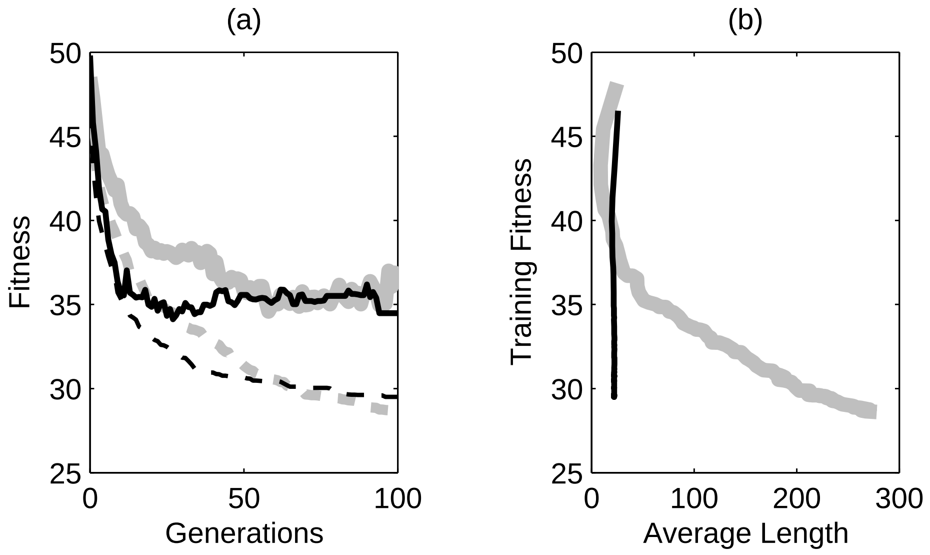


Figure 4.4: Test Function: *bioavailability (%F) regression*. All the rest as in Figure 4.3.

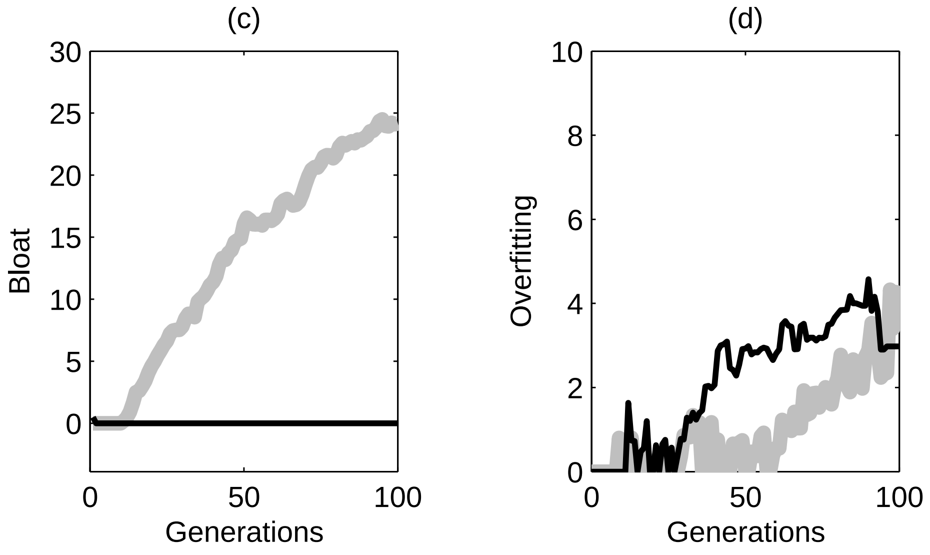


Figure 4.4: Test Function: *bioavailability (%F) regression*. All the rest as in Figure 4.3.

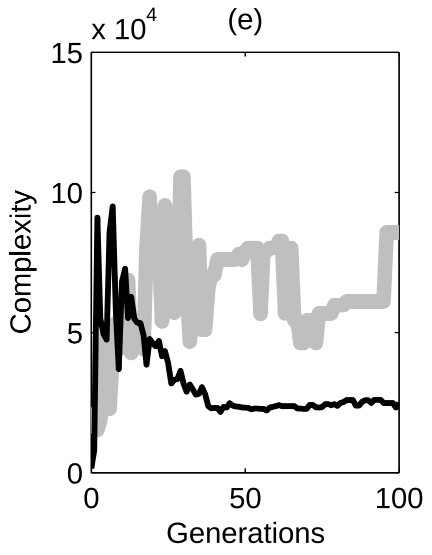


Figure 4.4: Test Function: *bioavailability (%F) regression*. All the rest as in Figure 4.3.

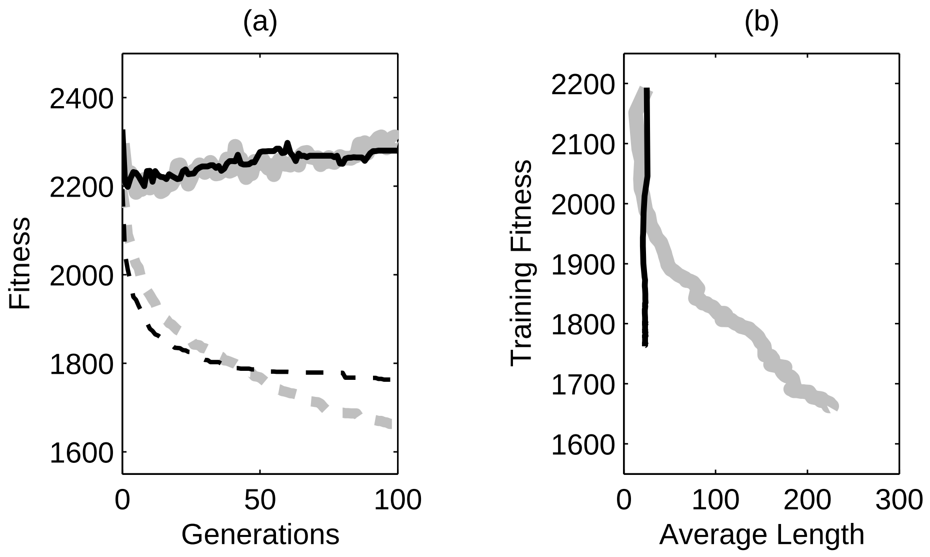


Figure 4.5: Test Function: *toxicity (LD50) regression*. All the rest as in Figure 4.3.

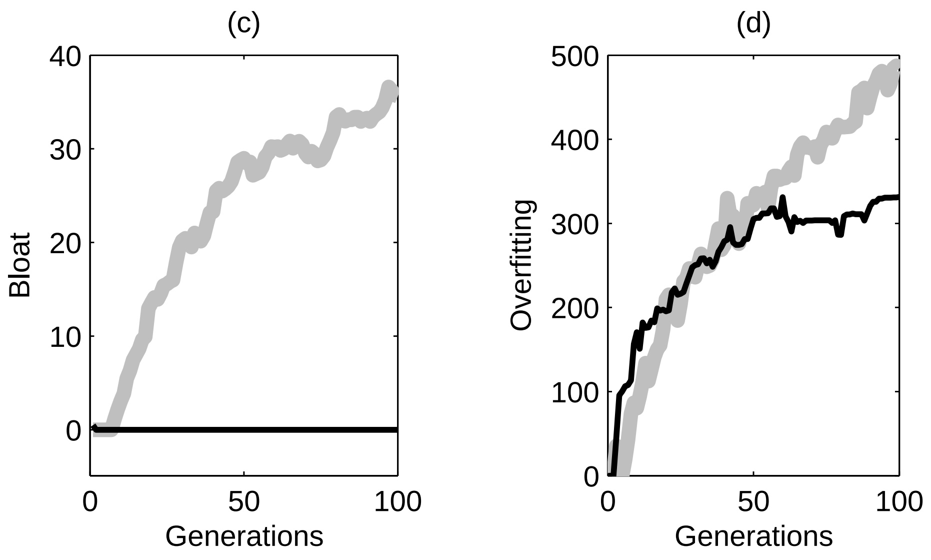


Figure 4.5: Test Function: *toxicity (LD50) regression*. All the rest as in Figure 4.3.

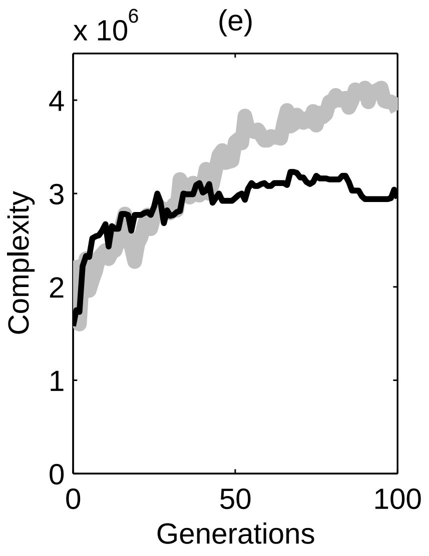


Figure 4.5: Test Function: *toxicity (LD50) regression*. All the rest as in Figure 4.3.

is better than training fitness, while it slightly overfits in the second part, where training fitness improves and test fitness remains approximately constant. StdGP shows a similar behavior, with a slightly more marked overfitting in the first part of the run. It is interesting to point out that the overfitting measure (plot 4.3(d)) captures this fact, given that for both methods it stays constantly equal to zero, except for some small oscillations in the second part of the run and for some larger oscillations for StdGP in the first part. By observing plot 4.3(e), it is possible see that the functional complexity of the individuals found by StdGP and DynOpEq is more or less the same at the end of the run, but StdGP increases its values sooner and faster than DynOpEQ. There is also a clear correlation between complexity and the best training fitness (plot 4.3(a)). Plot 4.3(b) clearly shows that StdGP bloats, while DynOpEq does not bloat. In fact, as training fitness keeps improving, the average length in the StdGP population increases, while it remains approximately constant in the DynOpEq population. Consistently, after a small variation in the first five generations, the bloat measure (plot 4.3(c)) is constantly equal to zero for DynOpEq, while it steadily grows for StdGP.

Now let us focus on the results obtained on the %F dataset (Figure 4.4). Plot 4.4(a) shows that DynOpEq starts overfitting earlier than StdGP. Now observe the behavior of the overfitting measure (plot 4.4(d)). The measure has larger values for DynOpEq from the beginning of the run, but they are met by the StdGP values towards the end of the run. This is partially captured by the complexity measure (plot 4.4(e)), where the values for DynOpEq increase sooner than the values for StdGP. Despite the overfitting, the complexity values soon decline and remain surprisingly low for DynOpEq. Plot 4.4(b) clearly shows that StdGP bloats while DynOpEq does not bloat. Consistently, the bloat measure remains constantly equal to zero (except for a small oscillation in the first three generations) for DynOpEq, while it steadily keeps growing during the whole evolution for StdGP.

Finally, let us focus on the results obtained on the LD50 dataset (Figure 4.5). Plot 4.5(a) shows that both methods overfit, but StdGP overfits more than DynOpEq, in particular in the second part of the run. In fact, while training fitness keeps improving for both methods during the whole evolution, test fitness has a minimum and then, for both methods, it keeps deteriorating. This minimum is around generation 10 for both methods, but this deterioration is clearly stronger for StdGP. In fact the fitness on the test set keeps deteriorating until the end of the evolution for StdGP, while it remains approximately constant after generation 60 for DynOpEq. The behavior of the overfitting measure (plot 4.5(d)) is consistent with this. It keeps growing during the whole evolution for StdGP, while for DynOpEq it grows in the first phase of the evolution and then stabilizes on a more or less constant value, lower than the values of StdGP. Also the complexity measure behaves consistently (plot 4.5(e)). In fact, at the end of the evolution DynOpEq starts producing less complex individuals than StdGP. Plot 4.5(b) clearly shows that also for this test problem StdGP bloats while DynOpEq does not bloat. Consistently, the bloat measure (plot 4.5(c)) is always equal to zero for DynOpEq, while it steadily keeps growing during the whole evolution for StdGP.

Conclusions

Even though promising, the results that have been obtained do not have to be emphasized. In fact, to use each proposed measures to make strong inferences about bloat, overfitting, functional complexity and their mutual relationship, it is necessary to overcome some of their drawbacks For instance:

(1) The bloat measure compares the fitness and the length of programs at a given generation with the fitness and the length of programs at generation zero. This is a good

starting point, given that it is possible to assume that there is no bloat at generation zero (as described above). Nevertheless, the initialization algorithm (that defines the population at generation zero) has biases both on program length and on fitness. These biases clearly affect the measure and should be taken into account. Or otherwise, a measure that does not take generation zero as a reference deserves to be defined and investigated.

(2) As already discussed above, the overfitting measure clearly depends on how training and test sets have been chosen. Given that at each GP run a different (random) partitioning of the dataset into training and test sets has been used, it may happen that exactly the same population has two different values of the overfitting measure in two different runs. As a good starting point, 30 independent runs of each experiment have been performed, and median values have been reported, hoping that this mitigates the bias of the measure given by its dependence on the training/test partitioning. Furthermore, it must be pointed out that having exactly the same population in two different runs is a very unlikely event. Nevertheless the problem exists and it is important. New versions of this measure deserve to be defined, for instance alternating training and test data in a crossvalidation-like way.

Last but not least (3), the functional complexity measure has formally *no relationship* with curvature. It is an empirical indicator, that captures the idea of curvature only from an intuitive viewpoint. It is a good staring point, because compared to other (more formal) measures it has the advantage of being simple and computationally cheap. Furthermore, the number of slope changes and (even more) the number of slope sign changes are clearly related to the empirical intuition of the concept of functional complexity. Nevertheless, the suspect is that the lack of formality of this measure may cause problems in the future and more formal measures, although hopefully still computationally cheap, deserve investigation.

4.4 Graph Based Complexity and Graph Based Learning Ability

The measure proposed in this section has three different, but related, goals:

- defining a new measure of functional complexity, that is rotationally invariant and that overcomes the limitations of the measure proposed in [107];
- defining a new measure to quantify the ability of GP to learn "difficult" points (my intuition of what a "difficult" point is will be explained in Section 4.4 where the measure will be defined) and studying its correlation with generalization;
- defining a new fitness function (inspired by the two previously defined measures) to improve GP generalization ability in those cases where standard GP (i.e. GP that calculates fitness using the root mean squared error between outputs and targets) has poor generalization ability.

Proposed Measures

The complexity measure proposed in this section and presented in [21], as the one introduced in [107] and described in section 4.3, is inspired by the idea that complex functions should have a larger curvature than simple ones. But, contrarily to [107], in this work the idea of curvature is quantified using the following intuition: let

g be a GP individual; the curvature of g can be expressed by counting the number of pairs of "*close*" training points $\mathbf{x}$ and $\mathbf{y}$ for which the corresponding values $g(\mathbf{x})$ and $g(\mathbf{y})$ are "far". In more formal terms, given a GP individual g, let $S = \{(\mathbf{x_1}, g(\mathbf{x_1})), (\mathbf{x_2}, g(\mathbf{x_2})), \ldots, (\mathbf{x_m}, g(\mathbf{x_m}))\}$ be the set of training points $\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_m}$ associated with the corresponding values assumed by g on them. Let $\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_m} \in \mathbf{X}$ and $g(\mathbf{x_1}), g(\mathbf{x_2}), \ldots, g(\mathbf{x_m}) \in \mathbf{Y}$, with both $\mathbf{X}$ and $\mathbf{Y}$ being metric spaces equipped with metrics $d_{\mathbf{X}}$ and $d_{\mathbf{Y}}$ respectively. For any $i = 1, 2, ..., m$, and for any prefixed constant value δ, let $B_\delta(\mathbf{x_i})$ be the open ball of radius δ centered on $\mathbf{x_i}$ in metric space $\mathbf{X}$, i.e. $B_\delta(\mathbf{x_i}) = \{\mathbf{x_j} \| \, d_{\mathbf{X}}(\mathbf{x_i}, \mathbf{x_j}) < \delta\}$. Analogously, for any $i = 1, 2, ..., m$, and for any prefixed constant value ε, let $B_\varepsilon(g(\mathbf{x_i}))$ be the open ball of radius ε centered on $g(\mathbf{x_i})$ in metric space $\mathbf{Y}$. For every training point $\mathbf{x_i}$, it is possible to define the set

$$V(\mathbf{x_i}) = \{\mathbf{x_j} \in B_\delta(\mathbf{x_i}) \, \| \, g(\mathbf{x_j}) \notin B_\varepsilon(g(\mathbf{x_i})) \text{ and } \mathbf{x_j} \neq \mathbf{x_i}\}. \tag{4.2}$$

The set $V(\mathbf{x_i})$ contains all the points of the sample set that are close (i.e. nearer than a given δ) to $\mathbf{x_i}$ in the $\mathbf{X}$ metric space, but whose values under g are not close (i.e. farther than a given ε) to the value of $g(\mathbf{x_i})$ in the $\mathbf{Y}$ metric space. Consider now the set $V = \cup_{i=1}^{m} V(\mathbf{x_i})$. V can be addressed as the set of points in $\mathbf{X}$ in which the function represented by the GP individual g is *rugged*, thus the fraction of training points that belong to set V can be used as an intuitive measure of curvature: $\frac{\mathsf{V}}{m}$. Clearly, values near 1 denote a very rugged function, while values near 0 indicate flat (or straight) and thus less complex functions. Now, it is interesting to consider not only the set $V(\mathbf{x_i})$ containing the *points* of ruggedness since, in the union $\cup_{i=1}^{m} V(\mathbf{x_i})$ the information on what are the *pairs* of points that are close, and the corresponding function values far, is lost. So, the set $E = \cup_{i=1}^{m} (\{\mathbf{x_i}\} \times V(\mathbf{x_i}))$ is introduced. Now, if defining the set $E_{\text{tot}} = \{(\mathbf{x_i}, \mathbf{x_j}) \parallel \mathbf{x_j} \in B_\delta(\mathbf{x_i}) \setminus \{\mathbf{x_i}\}\}$, it is possible to define the complexity measure as:

$$\text{GBC} = \mathsf{E/E\_\{tot\}}$$

It is important to remark that, in case of symbolic regression problems, what usually happened is that $\mathbf{X} \subseteq \mathbb{R}^n$ and $\mathbf{Y} \subseteq \mathbb{R}$. Thus, it is possible to calculate GBC, for instance, using the Euclidean distance as the $d_\mathbf{X}$ and $d_\mathbf{Y}$ metrics. Thus GBC is rotationally invariant, contrarily to what happens for the complexity measure defined in [107]. The acronym chosen as the name of this measure (GBC, which stands for Graph Based Complexity) depends on the fact that it is possible to represent it in terms of counting operations on a graph. Let $G = (\{\mathbf{x_1}, \ldots, \mathbf{x_m}\}, E_{\text{tot}})$ be a graph defined on the training points, where two vertices are connected if their distance on the metric space $\mathbf{X}$ is less than δ. Now consider the subgraph $G_\varepsilon = (\{\mathbf{x_1}, \ldots, \mathbf{x_m}\}, E)$ which only contains the edges $(\mathbf{x_i}, \mathbf{x_j})$ of G such that the distance between $g(\mathbf{x_i})$ and $g(\mathbf{x_j})$ in the $\mathbf{Y}$ metric space is greater or equal than ε. The GBC measure is clearly equal to the ratio between the number of connections in G_ε and the number of connections in G.

The GBC function can be used to quantify the complexity of GP individuals. However, it is also clear that the same calculation can be performed using the known target values (instead of using the values assumed by the learned function) on the different training points. In particular, indicating by $f(\mathbf{x_i})$ the target value on a training point $\mathbf{x_i}$, it is possible to define a set V' (analogous to the set V previously defined) as follows: $V'(\mathbf{x_i}) = \{\mathbf{x_j} \in B_\delta(\mathbf{x_i}) \parallel f(\mathbf{x_j}) \notin B_\varepsilon(f(\mathbf{x_i}))$ and $\mathbf{x_j} \neq \mathbf{x_i}\}$. And thus, it is also possible to define a set E' as follows: $E' = \cup_{i=1}^{m} (\{\mathbf{x_i}\} \times V'(\mathbf{x_i}))$. Following the same idea that used to define the GBC measure, it is possible to state that $\text{GBC}_{target} = \text{E'/E\_\{tot\}}$ is a measure that can be used to quantify the ruggedness of the target function.

Furthermore, *both* information coming from sets E and E' can also be used to quantify the ability of a GP individual to learn "difficult" points. For this aim the following measure, called GBLA (Graph Based Learning Ability), is defined:

$$\text{GBLA} = (\|E \bigtriangleup E'\|)/\|E_{\text{tot}}\|$$

where $\bigtriangleup$ represents the operator of symmetrical difference between sets. GBLA quantifies the number of training points where the target function is rugged and the learned function is flat, plus the number of training points where the learned function is rugged and the target function is flat. For simplicity, these points are called "difficult" points.

Both the definitions of GBC and GBLA are based on the definition of the $V(\mathbf{x_i})$ set (equation (4.2)). The elements of $V(\mathbf{x_i})$ depend on the choice of the two parameters δ and ε. Consequently, also the values of GBC and GBLA depend on these two parameters. Nevertheless, a set of preliminary experiments have indicated an interesting fact concerning these two parameters: considering many series composed by the values of the GBC of the best individual in the population for each iteration for several pairs of values of δ and ε, all these series have a positive value of their mutual cross correlation coefficient, with a magnitude of this coefficient approximately equal to 1. The same fact has also been observed for GBLA. Given that in the experimental study presented in Section 4.4 the main objective is to understand the *correlation* of GBC and GBLA with other quantities during the GP runs (rather than the particular values of GBC and GBLA), it is possible to assert that parameters δ and ε do not qualitatively affect the results of Section 4.4, neither the conclusions that can be draw from them. All the experiments reported in Section 4.4 are also repeated for several other pairs of values of δ and ε and all the experimental results have confirmed that the values of δ and ε do not affect the qualitative interpretation of the results.

Experimental setting

A total of 120 runs where performed to obtain the results reported in this section. All the runs used a population of 200 individuals. The number of generations performed was equal to 100 for the LD50 and %F datasets and to 500 for %PPB (the reason why a larger number of generations for %PPB has been executed will be clear next in the current section). Tree initialization was performed with the Ramped Half-and-Half method [57] with a maximum initial depth of 6. The function set contained the four binary operators $+$, $-$, $\times$, and $/$, protected as in [57]. No random constants were added to the terminal set. Because the cardinalities of the function and terminal sets were so different, a balanced choice between functions and terminals when selecting a random node has been imposed . Unless where explicitly pointed out, fitness was calculated as the root mean squared error (RMSE) between outputs and targets. Tournament selection was used with size 10. The reproduction (replication) rate was 0.1, meaning that each selected parent has a 10% chance of being copied to the next generation instead of being engaged in breeding. Standard tree mutation and standard crossover (with uniform selection of crossover and mutation points) were used with probabilities of 0.1 and 0.9, respectively. Selection for survival used elitism (i.e. unchanged copy of the best individual in the next population). A fixed maximum depth equal to 17 was used for the trees in the population. These parameters are absolutely identical to the ones used in [107]. Given that, as pointed out previously, the δ and ε parameters do not affect the qualitative interpretation of the results contained in this section, the results obtained for two arbitrary values, i.e.: $\delta = 0.06$ and $\varepsilon = 0.05$ are reported. All distance values have been normalized into the range [0,1] before comparing them with the values of δ and ε.

Experimental results: GBC and GBLA

In Figure 4.6 the median over 120 independent runs of the RMSE on the training set, the RMSE on the test set, the value of GBC, the value of 1-GBLA and finally the complexity and overfitting measures introduced in [107] for all the performed generations (first column: LD50; second column: %F; third column: %PPB) are reported. From now on the terms “complexity” and “overfitting” are used to indicate the complexity and overfitting measures introduced in [107] while the terminology “RMSE on the test set” indicates the RMSE on the test set of the individual with the best RMSE on the training set.

The relationship between RMSE on the training and test set for the studied problems is firstly considered. For LD50 the RMSE on the training set steadily decreases during the whole evolution, while the RMSE on the test set keeps increasing after generation 30, also showing an irregular and oscillating behaviour. For %F both the RMSE on training and test set are steadily decreasing during the studied 100 generations. For %PPB the RMSE on the training set steadily decreases during the whole evolution, while the RMSE on the test set is decreasing until generation 50, and then increasing until generation 500, also showing some oscillations. It is possible to conclude that GP has a worse generalization ability for LD50 and %PPB than for %F.

Now the curves representing GBC and 1-GBLA are considered. Both GBC and 1-GBLA are increasing after generation 30 for LD50, steadily decreasing (except for the initial part of the run) for %F and increasing (except for the first 50 generations) for %PPB. These results hint a relationship between the trend of the RMSE on the test set and the GBC and GBLA measures for all the studied problems. In particular, GBC seems to have a positive correlation with the RMSE on the test set and GBLA seems to have a negative correlation with the RMSE on the test set. Before studying in details these correlations, the trend of the complexity and overfitting measures is considered. The complexity measure seems to have a less clear relationship with the RMSE on the test than GBC and GBLA for LD50 and %F. On the other hand, for %PPB the complexity measure seems to be growing with a higher speed than GBC and 1-GBLA, and thus it seems to have a stronger correlation with the RMSE on the test set. Finally, it is possible to note that the overfitting measure has a more oscillating and less regular behavior than the other measures. Nevertheless, its general trend seems related to GBC and to the complexity measure for LD50 and to GBC, GBLA and complexity for %PPB. On the other hand, no clear relationship appears between the overfitting measure and any of the other measures for %F.

In order to better understand the mutual relationships between the quantities plotted in Figure 4.6, in Figure 4.7 the median values of the cross correlation at delay zero between them are reported. Cross correlation is a standard method of estimating the degree to which two series $S_1 = \{x_1, x_2, ..., x_n\}$ and $S_2 = \{y_1, y_2, ..., y_n\}$ are correlated. It is defined by $r = (\sum_{i=d+1}^{n} \|(x_i - \overline{S_1})(y_{i-d} - \overline{S_2})\|)/(\sigma_{S_1}\sigma_{S_2})$, where $\overline{S_1}$ and $\overline{S_2}$ are the averages of the elements in the series S_1 and S_2 respectively, σ_{S_1} and σ_{S_2} are their respective standard deviations, and d is the delay (in this work $d = 0$). An introduction to cross correlation can be found, for instance, in [35]. Assume the objective is to calculate the correlation between GBC and RMSE on the test set, as it is the case, for instance, in the left-top plot of Figure 4.7. This can be done by considering as S_1 the series composed by the GBC values of the best individuals in the population at each generation and as S_2 the series of the respective RMSE values on the test set of the same individuals. Applying the same method, it is possible to calculate the cross correlation of any pair of measures reported in Figure 4.6. In Figure 4.7, two horizontal lines at the values -0.15 and 0.15 are also drawn, empirically identified as thresholds between the presence of a correlation (either positive or negative) and the absence of correlation in [54]. Even though these thresholds are quite arbitrary, they are used in the interpretation of the presented results.

For LD50: during the whole run, GBC has a positive cross correlation with the RMSE on the test set and overfitting and a negative cross correlation with the RMSE on the training set. GBLA has a negative cross correlation with the RMSE on the test set, a positive cross correlation with the RMSE on the training set and cross correlation approximately equal to zero with overfitting. Complexity has a positive cross correlation with overfitting and a negative cross correlation with the RMSE on the training set. The cross correlation between complexity and RMSE on the test set is approximately equal to zero, except for some small oscillations at the end of the run.

For %F: GBC has a positive cross correlation with RMSE both on the training and test set (and this value is steadily increasing during the run) and a cross correlation approximately equal to zero with overfitting. GBLA has a negative (and steadily decreasing) cross correlation both with the RMSE on the training and on the test set and a cross correlation approximately equal to zero with overfitting. Finally, complexity has a cross correlation approximately equal to zero with overfitting and a positive cross corrlation with both RMSE on the training and test set (even though in both cases the cross correlation becomes larger than 0.15 only in the final part of the run).

For %PPB: the value of the cross correlation between GBC and the RMSE on the training and test set is negative, but steadily increasing, in the first 100 generations. For this reason, simulations until generation 500 have been executed, to see if some of these correlations became positive later in the evolution. It is possible to see that the correlation between GBC and the RMSE on the test set becomes positive more or less at generation 350, and it keeps on growing, although without becoming larger than 0.15. Because of the steadily growing trend of the curve of the cross correlation between GBC and RMSE on the test set, it is possible to hypothesize that this cross correlation would become positive later in the run (this hypothesis will be verified in the future by performing runs for a larger number of generations than 500). On the other hand, the cross correlation between GBC and RMSE on the training set is clearly negative during the whole run. Finally, the cross correlation between GBC and overfitting is positive (it becomes larger than 0.15 around generation 280, and it remains larger than 0.15 until the end of the run). GBLA has a negative cross correlation with both overfitting and RMSE on the test set and a positive cross correlation with RMSE on the training set. Finally, complexity has a positive cross correlation with both overfitting and RMSE on the test set and a negative cross correlation with RMSE on the training set.

Summarizing: GBC is positively correlated with the RMSE on the test set and GBLA is negatively correlated with the RMSE on the test set. These facts seem independent on the generalization ability of GP (i.e. they hold for all the studied problems). Furthermore, the magnitude of the correlation with the RMSE on the test set is larger for GBC and GBLA than for the complexity measure for all studied problems except %PPB.

The negative values of the correlation between GBLA and RMSE on the test set can be interpreted as follows: GBLA quantifies the ability of GP to learn the “difficult” training points. It is intuitive that a good learning of those points leads GP to a poor generalization ability, because the solutions are too specialized on training data and thus overfit them. This can be caused, for instance, by the fact that those “difficult” points correspond to “noise”, or even errors in the training data, or they are generally not useful to reliably reconstruct the target function.

The existing relationship between GBC and GBLA with the RMSE on the test set seems to hint that the ideas used to define GBC and GBLA could be useful to build a new fitness function able to reduce the error on the test set. This is the goal of the next paragraph.

Experimental results: New Fitness Function

Using either GBC or GBLA as new fitness functions does not allow to obtain interesting results. Consider, for instance, the case of GBLA: each function able to learn some particular points (the ones that are not considered as difficult) would have a good fitness, and thus it would receive a high probability of surviving and mating in the GP population, independently of the distance of that function from the target one. Besides the intuition, also a set of preliminary experimental results confirm that using either GBC or GBLA as fitness functions does not allow to obtain better results than standard GP (i.e. GP that uses the RMSE as fitness) on the test set.

Nevertheless, the ideas used to define GBC and GBLA can be used to define a new fitness function, assuming to integrate them with the error between learned values and target ones. A possibility could be to use them together with RMSE in a multi-objective method. Even though the idea is interesting, and it definitely deserves to be investigated in the future, in this work the aim is to define *one* new fitness function able to incorporate both the information derived from the RMSE and from the new measures. The idea is to give a *weight* to the error in each training point. For this reason, the new fitness function is called "weighted_fitness". The weight should depend on how rugged the learned function is in that point, reducing the weight of the rugged points. The new fitness measure is:

$$\text{weighted\_fitness}(g) = \sum_{i=0}^{m} \frac{(f(\mathbf{x_i}) - g(\mathbf{x_i}))^2}{1 + \|V(\mathbf{x_i})\|} \tag{4.3}$$

where $\mathbf{x_1}$, $\mathbf{x_2}$, ..., $\mathbf{x_m}$ are the training points, g is a GP individual, the values $f(\mathbf{x_1})$, $f(\mathbf{x_2})$, ..., $f(\mathbf{x_m})$ represent the targets on those points and $V(\mathbf{x_i})$ is the set defined in equation (4.2). The fact that the denominator in equation (4.3) is $1 + \|V(\mathbf{x_i})\|$ instead of $\|V(\mathbf{x_i})\|$ is due to the fact that the value of $\|V(\mathbf{x_i})\|$ could be equal to zero, and thus adding 1 prevents from the eventuality of an error.

In Figure 4.8 an experimental comparison between standard GP and GP that uses the new fitness function defined in equation (4.3) is reported. The median of the RMSE on test data for each performed generation for both these models is reported. For LD50, GP that uses the new fitness function is able to obtain better results. For both %F and %PPB, GP using the new fitness function seems to return very similar results than standard GP.

It is possible to conclude that GP using the proposed fitness function is able to better generalize (compared to standard GP) for some problems where standard GP has a poor generalization ability (as it is the case of the LD50), while it behaves comparably to standard GP when standard GP itself has a good generalization ability (like for %F). Nevertheless, problems where standard GP has a poor generalization ability and the new fitness function is not able to improve it exist (it is the case of %PPB). But at least, it has been shown that in this last case, the new fitness function does not worsen the results. These results indicate that the proposed fitness function could be always used, given that in some cases it gives an advantage when standard GP has poor generalization, and when it doesn't, at least, it does not give any disadvantage.

Conclusions

A study of GP learning ability has been presented in section 4.4, offering the three following contributions: first, a new measure (GBC) to quantify the functional complexity of GP individuals has been defined . Compared to another complexity measure defined in [107], GBC is rotationally invariant and it has a higher correlation with the quality of GP solutions on out-of-sample data. Secondly, we have presented a new measure (GBLA) aimed at quantifying the ability of GP to learn "difficult" points and we have shown that this measure is negatively correlated with the quality of GP solutions on out-of-sample data. Based on these ideas, the third contribution consisted in defining a new fitness function for GP. Experimental results have shown that this new fitness function allows GP to better generalize for some problems where standard GP has a poor generalization, without worsening the results in all other cases. This seems to indicate the suitability of this fitness function in any possible case. The new fitness function has to be further studied in the future. In particular, we will test GP with the new fitness function on several other datasets of different complexities and also on a set of hand-tailored symbolic regression problems of various different difficulties. Furthermore, we will consider several possible extensions and improvements of the new fitness function.

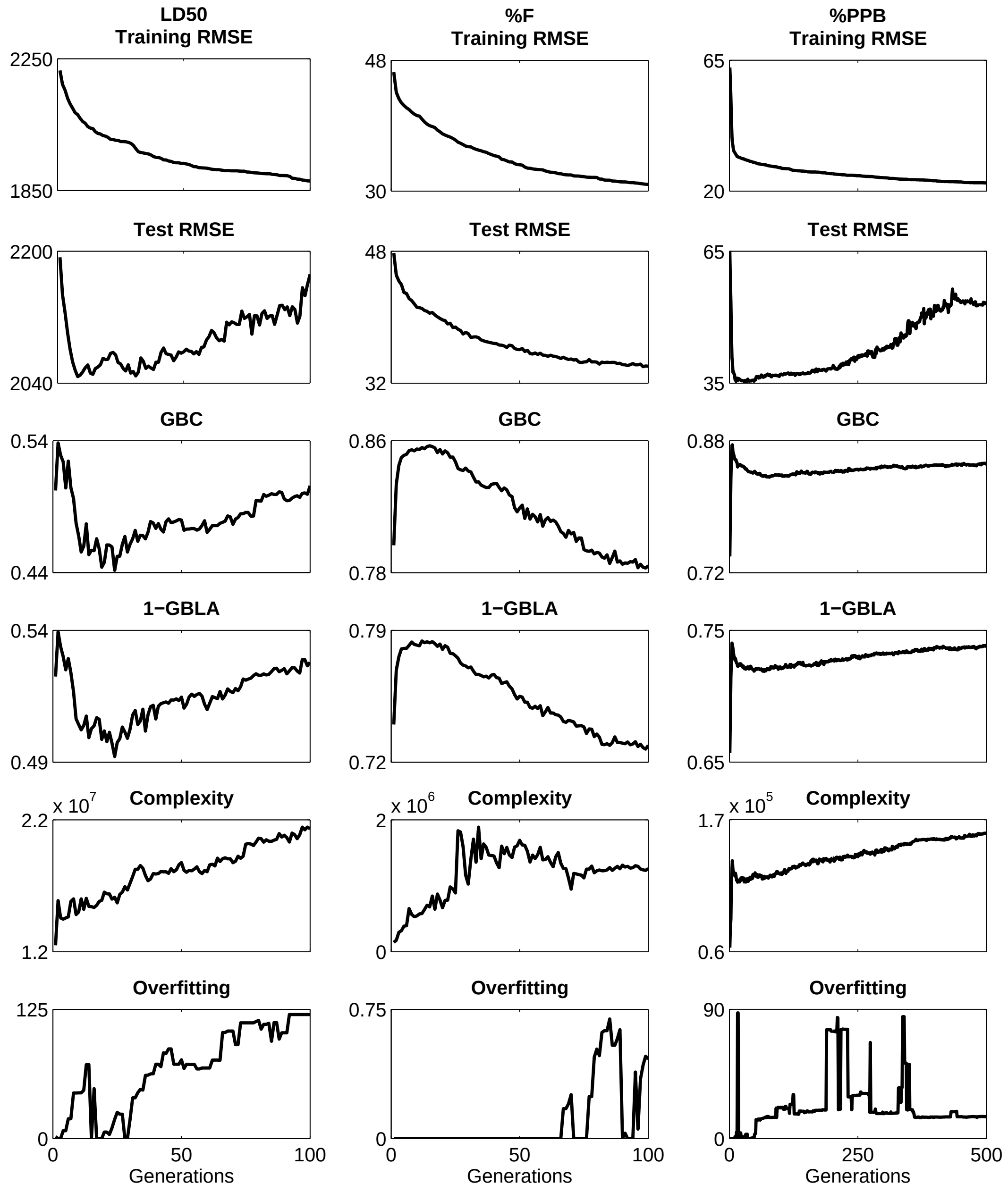


Figure 4.6: The first (respectively second and third) column reports the results for LD50 (respectively %F and %PPB). For each column, from top to bottom, the RMSE on the training set, the RMSE on the test set, the GBC, 1-GBLA and the values of the complexity and the overfitting measures introduced in sections 4.2 and 4.3 are reported. All these results are reported against generations and they are medians of the value assumed by the best individual over 120 independent runs.

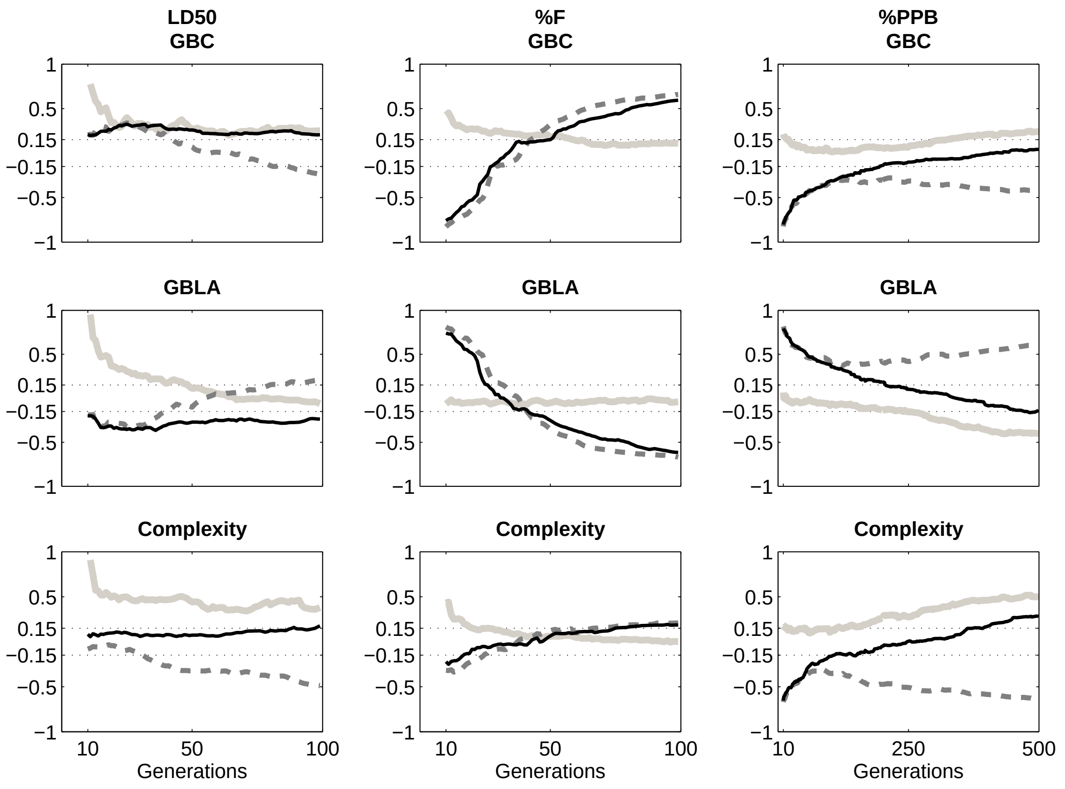


Figure 4.7: The first (respectively second and third) column reports the results for LD50 (respectively %F and %PPB). For each column, from top to bottom, the cross correlation of GBC, GBLA and complexity with the RMSE on the training set (dashed dark grey line), the RMSE on the test set (black line) and the overfitting measure (solid light grey line) are reported. All these results are reported against generations and they are medians over 120 independent runs.

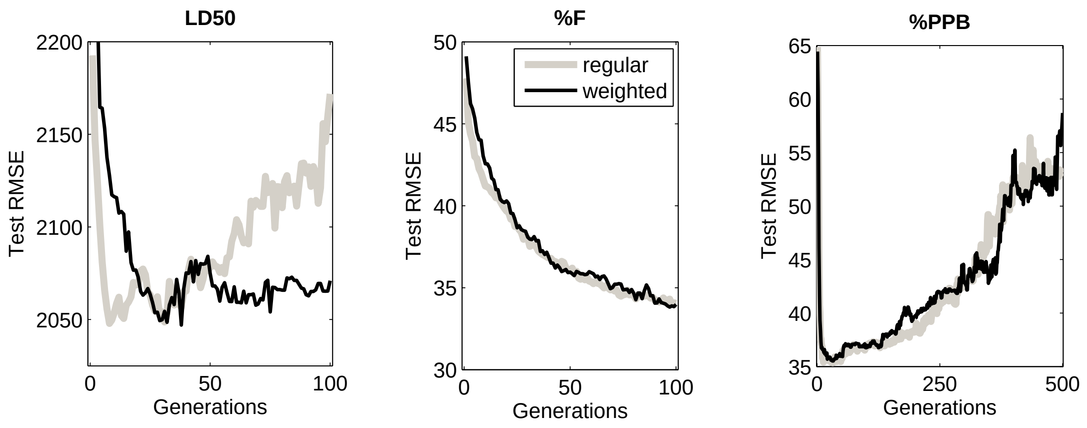


Figure 4.8: RMSE on the test set of standard GP (i.e. GP that uses the RMSE as fitness, indicated by "regular" in figure) and of GP that uses the fitness function defined in equation (4.3) (indicated by "weighted" in the figure). Left: LD50. Middle: %F. Right: %PPB.

CHAPTER 5

Multi Objective Optimization in Genetic Programming

Contents

5.1 Introduction

In this chapter an experimental study of the generalization ability of a set of multi-optimization GP frameworks is presented, comparing them with standard GP and Dynamic Operator Equalization [93]. The idea of optimizing more than one criterion on training data to produce more general solutions is not new: a related idea has been used in [39] in the domain of binary classification (where a two-objective sort is performed in order to extract a set of non-dominated individuals) and in [105] for regression on simple benchmark problems characterized by a small dimensionality of the feature space. Furthermore, in [111], the authors define two measures of complexity for the GP individuals, they use them as further criteria (besides fitness) in a multi-objective system based on Pareto optimization, and they show that this system is able to counteract bloat and overfitting.

Compared to those contributions, in this work different objectives are used, we test our methods on a real-life problem and we make use of the NSGA-II [28] multi-objective evolutionary method.

The functions we use in the multi-optimization algorithm are the fitness of the problem and one or more other functions that can intuitively be related to overfitting,described later.

The test problem we use consists in predicting the value of an important pharmacokinetic parameter (*Median Lethal Dose* or LD50 from now on) of a set of candidate drug compounds on the basis of their molecular structure. This application is an example of regression problem characterized by a large space of features, and it has already been introduced in [106].

This chapter is structured as follows: in Section 5.2 operator equalization is introduced;

in Section 5.3 basic concepts about multi objective optimization are introduced; in Section 5.4 the multi-optimization framework used is presented; Section 5.5 presents the experimental setting, including a description of the dataset and of the parameters used; in Section 5.6 we present and discuss the obtained experimental results; finally Section 5.7 concludes the chapter and discusses ideas for future research.

5.2 Operator Equalization

Developed alongside the crossover bias theory [30, 32, 79, 84], Operator Equalisation (Op.Eq.) is a recent technique to control bloat that allows an accurate control of the program length distribution inside a population during a GP run. To better explain how it works, we use the concept of a histogram. Each bar of the histogram can be imagined as a bin containing those programs whose length is within a certain interval. The width of the bar determines the range of lengths that fall into this bin, and the height specifies the number of programs allowed within. We call the former *bin width* and the latter *bin capacity*. All bins are the same width, placed adjacently with no overlapping. Each length value, l, belongs to one and only one bin b, identified as follows:

$$b = \left\lfloor \frac{l-1}{bin\_width} \right\rfloor + 1 \tag{5.1}$$

For instance, if $bin\_width = 5$, bin 1 will hold programs of lengths 1,..,5, bin 2 will hold programs of lengths 6,..,10, etc. The set of bins represents the distribution of program lengths in the population. Op.Eq. biases the population towards a desired target distribution by accepting or rejecting each newly created individual into the population (and into its corresponding bin). The original implementation of Op.Eq. [33], where the user was required to specify the target distribution and maximum program length, rapidly evolved to a self adapting implementation called DynOpEq [93,95,110], where both these elements are automatically set and dynamically updated to provide the best setting for each stage of the evolutionary process. There are two tasks involved in DynOpEq: calculating the target (in practical terms, defining the capacity of each bin) and making the population follow it (making sure the individuals in the population fit the set of bins).

5.2.1 Calculating the Target Distribution

In DynOpEq the dynamic target length distribution simply follows fitness. For each bin, the average fitness of the individuals within is calculated, and the target is directly proportional to these values. Bins with higher average fitness will have higher capacity, because that is where search is proving to be more successful. Formalizing, the capacity, or target number of individuals, for each bin b, is calculated as:

$$bin\_capacity_b = round\left(n \times (\bar{f}_b / \sum_i \bar{f}_i)\right)$$

where $\bar{f}_i$ is the average fitness in the bin with index i, $\bar{f}_b$ is the average fitness of the individuals in b, and n is the number of individuals in the population.

Initially based on the first randomly created population, the target is updated at each generation, always based on the fitness measurements of the current population.

This creates a fast moving bias towards the areas of the search space where the fittest programs are, avoiding the small unfit individuals resulting from the crossover bias, as well as the excessively large individuals that do not provide better fitness than the smaller ones already found.

5.2.2 Following the Target Distribution

In DynOpEq every newly created individual must be validated before eventually entering the population. In particular, DynOpEq rejects the individuals that do not fit the target: individuals from the population are selected for mating and the application of genetic operators allow us to create new individuals, as in standard GP. After that, the length of each new individual is measured, and its corresponding bin is identified using Equation (5.1). If this bin already exists and is not full (meaning that its capacity is higher than the number of individuals already there), the new individual is immediately accepted. If the bin still does not exist (meaning it lies outside the current target boundaries) the fitness of the individual is measured and, in case we are in the presence of the new best-of-run (the individual with best training fitness found so far), the bin is created to accept the new individual, becoming immediately full. Any other non existing bins between the new bin and the target boundaries also become available with capacity for only one individual each. The criterion of creating new bins whenever needed to accommodate the new best-of-run individual is inspired by the successful Dynamic Limits [91] bloat control technique.

When the intended bin exists but is already at its full capacity, or when the intended bin does not exist and the new individual is not the best-of-run, DynOpEq evaluates the individual and, if we are in the presence of the new best-of-bin (meaning the individual has better fitness than any other already in that bin), the bin is forced to increase its capacity and accept the individual. Otherwise, the individual is rejected. Permitting the addition of individuals beyond the bin capacity allows a clever overriding of the target distribution, by further biasing the population towards the lengths where the search is having a higher degree of success. In the second case, when the bin does not exist and the individual is not the best-of-run, rejection always occurs[1].

5.3 Multi Objective Optimization

The main concepts of the majority of multi-optimization algorithms are *dominance* and *Pareto optimality*. Consider a set $\mathcal{S}$ called *solution space*. The *fitness* is a function $f : \mathcal{S} \mapsto \mathbb{R}$. Let $f_1, f_2, \ldots, f_n$ be n fitness functions. It is possible to build the function $F : \mathcal{S} \mapsto \mathbb{R}^n$ where $F(s) = (f_1(s), f_2(s), \ldots, f_n(s))$. This function combines n fitness functions to remap a solution into a vector inside a n-dimensional space. For the sake of simplicity suppose, without loss of generality, that the image space of all the n fitness functions is $\mathbb{R}_+$ and the optimal fitness is 0 for all the functions.

A solution s_1 is said to be dominated by a solution s_2 if $\forall i \in \{1, \ldots, n\} \quad F(s_1)_i = f_i(s_1) > f_i(s_2) = F(s_2)_i$. In other words, a solution s_1 is dominated by s_2 if s_2 is better than s_1 in respect to all the selected criteria. The set of solutions that are not dominated by any other solution is called *Pareto set*. A solution from this set is said to be *Pareto optimal*.

[1]We point out that there is an obvious computational overhead in evaluating so many individuals that end up being rejected. This subject has been extensively addressed in previous work [93] where the main conclusion was that most rejections happen in the beginning of the run and refer to very small individuals.

5.3.1 NSGA II

The main problem in using multi-optimization techniques is to find a way to combine the fitness values given by all the chosen criteria. The majority of the multi-optimization algorithms use in some way the concept of Pareto set.

The NSGA and NSGA-II [28] algorithms share the same basic idea. The NSGA-II is an improvement of NSGA that addresses some of the NSGA main criticisms. In particular it reduces the computational complexity, it introduces elitism and removes the need to specify an additional parameter to ensure diversity in population.

The main idea of the algorithm is to create a total order between a particular family of disjoint subsets of the population. A naive iterative algorithm to create this family of subsets is the following: (1) given a set $\mathcal{P}$ of individuals, find the *Pareto set* A of $\mathcal{P}$; (2) remove A from $\mathcal{P}$. For every set B removed before A we will have that $B \succ A$; (3) iterate the previous steps until $\mathcal{P} = \emptyset$.

The previous ordering is consistent in respect to the notion of dominance: if $B \succ A$ then B dominates A. The NSGA-II algorithm then adds an intra-set fitness in such a way that if $B \succ A$ then the elements of B will have a better fitness than the elements of A. In this way a total order between individuals is defined inside the population. Let A and B be two subsets of $\mathcal{P}$ from the previously defined family. Then for all $a \in A$ and $b \in B$ we have that:

$$a \succ b \Leftrightarrow A \succ B \text{ or } A = B \wedge a \succ_A b$$

where $\succ_A$ is the intra-set ordering function. The fitness is then defined using the ranking given by this total order.

For further details about the intra-set ordering function and the accurate definition of the NSGA and NSGA-II algorithm the reader is referred to [28].

5.4 Multi-optimization and Genetic Programming

In optimization problems the search bias usually involves only one fitness function. This is equivalent to the use of only one criterion to estimate the quality of a solution. On the other hand, in many situations a good solution is achieved by a compromise between multiple criteria [27]. The multi-optimization technique is applied to a problem that usually has only one optimization criterium by adding new GP-specific criteria. In this way the hope is that it is possible to counteract typical negative aspects of the GP evolution such as premature convergence, bloat and overfitting.

The choice of the fitness functions for multi-optimization is motivated by two objectives: (1)to find the optimal solution for the considered (single fitness) problem. This condition dictates the use of the original fitness function as one of the functions employed in the multi-optimization; (2) the necessity to enhance the generalization ability of the individuals in the population (that is their ability to perform well over data that have not been used during the training phase).

The choice for the first point is fixed: the original fitness function that, in the considered test case, is the *root mean square error* between target and obtained values, must be choosen. The choice for the second point is more problematic and it must be guided by previous empirical observations of possible relationships between observable properties of the evolution process (i.e. properties regarding the tree representing individuals or some kind of measure that encompasses the entire population) and the generalization ability of solutions.

In this work two different functions have been chosen. The first fitness function is the number of the nodes of the tree. Obviously this function should be minimized in order to counteract bloat and to reward small-sized trees. The choice of this function is interesting because it immediately demands for a comparison with DynOpEq. Both techniques, in fact, counteract bloat but the way they do it is different. While in DynOpEq code size reduction is achieved implicitly (following the target length distribution), in multi-optimization this is done explicitly.

The second function used is the variance of the error between the target and the obtained value. Intuitively the minimization of this value reduces the difference between the shape of the expected (unknown) function and the function described by the given individual. This kind of optimization is supposed to be related to overfitting as already discussed for instance in [72].

Because multi-optimization can be defined with an arbitrarily large number of functions, a natural extension of the two-function optimization is the use of three functions: the original fitness function and the other two auxiliary functions previously discussed. In this work, the goal is also to test if the use of more than one auxiliary function can result in a better generalization ability.

5.5 Experimental Setting

Here we describe how the data for the test problem was collected and prepared, and what parameters and tools were used in our experiments. The techniques tested were the standard GP algorithm, DynOpEq, and multi-objective optimization (NSGA-II) with different optimization criteria. It is important to underline that the considered multi-objective optimization technique differs from the standard NSGA II algorithm. In particular standard tournament selection has been used, considering only the error between targets and outputs as a criterion for tournament selection. Hence crowding distance has not been considered during the selection process. This means that other criteria are only used to select the best individuals at the end of the generation.

We have obtained a set of molecular structures and the corresponding LD50 values using the same data as in [117] and a public database of food and drug Administration (FDA) approved drugs and drug-like compounds [114]. Data have been gathered in a matrix composed by 234 rows (instances) and 627 columns (features). Each row is a vector of molecular descriptors values identifying a drug; each column represents a molecular descriptor, except the last one, that contains the known values of LD50. Training and test sets have been obtained by randomly splitting the dataset: at each GP run, 70% of the molecules have been randomly selected with uniform probability and inserted into the training set, while the remaining 30% form the test set (that is not used during the evolutionary phase).

A total of 30 runs were performed with each technique. All the runs used populations of 100 individuals allowed to evolve for 100 generations. Tree initialization was performed with the Ramped Half-and-Half method [82] with a maximum initial depth of 6. The function set contained the four binary operators $+$, $-$, $*$, and $/$ protected as in [82]. The terminal set contained all 234 variables and no random constants. Because the cardinalities of the function and terminal sets were so different, we have explicitly imposed functions and terminals to have the same probability of being chosen when a random node is needed. Fitness was calculated as follows: (1) for standard GP and DynOpEq fitness is the root mean squared error between outputs and targets. (2) For the first version of the multi optimization algorithm (from now on called MO_{nodes}) the two fitness functions are the standard GP fitness and the number of nodes. (3) For the second version of the multi optimization algorithm (from now on called MO_{var}) the two fitness functions are the standard GP fitness and variance of the errors between targets and outputs. (4) For the third version of the multi optimization algorithm (from now on called $MO_{var+nodes}$) the fitness functions used are the standard GP fitness and both the number of nodes and the variance of the errors between targets and outputs.

The reproduction (replication) rate was 0.1, meaning that each selected parent has a 10% chance of being copied to the next generation instead of being engaged in breeding. Standard tree mutation and standard crossover (with uniform selection of crossover and mutation points) were used with probabilities of 0.1 and 0.9, respectively. The new random branch created for mutation has maximum depth 6. Selection for survival was elitist. Regarding the parameters specific to each technique, standard GP used a fixed maximum depth of 17, DynOpEq used a bin width equal to 1.

The median was preferred over the mean in all the evolution plots shown in the next section because median is more robust to outliers.

5.6 Experimental results

In this section experimental results are presented, with a particular emphasis on the comparison between standard GP and the other presented techniques. Results obtained over the training data are firstly presented, and then the results obtained over test data are outlined.

Results on training data

The expectation is that, on the training data, standard GP achieves better results than the other studied methods, in particular the multi-optimization systems. Even though this expectation seems contradictory in respect to the scope of this work, it is not. In fact, all the presented techniques (except for standard GP) are guided by an auxiliary objective (implicit or explicit) that can sometimes act against the reduction of the error over the training data. In fact, the interest on the performance over the training data is quite limited while the main goal is in achieving good performances over test data. In Figure 5.1 each technique is represented by a box and pair of whiskers. Each box has lines at the lower quartile, median, and upper quartile values, and the whiskers mark the furthest value within 1.5 of the quartile ranges. Outliers are represented by $+$ symbols. The plot refers to the best fitness achieved on the final generation, measured on the training set. Results seem to confirm what is expected: standard GP is the best technique over the training data. DynOpEQ provides similar results, while multi-optimization algorithm's performances are poorer on training data regardless of the fitness function that guides the evolution. Fitness differences between standard GP and DynOpEQ (regarded as one group) and the three multi-optimization methods (as another group) are statistically significant. All the details of the analysis of statistical significance that have been performed on the experimental results are described in Section 5.6. Figures 5.2 to 5.6 report the best training fitness per generations. Reported data are the medians over 30 runs of every studied method. Standard GP and DynOpEq reach a better fitness values over training data, compared to the multi-optimization algorithms that have a worse fitness over those data.

Results on test data

In the test data, the expectation is to find differences in the fitness values between the multi-optimization systems and Standard GP and DynOpEQ Because the multi-optimization learning is guided by other goals other than the minimization of the training error, it is expected to be less greedy and allow a better exploration (as opposed to exploitation) of the search space.

Predicted results are confirmed by the plot in Figure 5.7, where MO_{size} reaches the best fitness value. Performances of MO_{Var} and $MO_{size+var}$ are also good and quite similar to the MO_{size} technique. Standard GP and DynOpEq perform worse than the multi-objective models on the test data, as expected, and, especially for standard GP, the presence of several outliers has been obeserved (not shown in Figure 5.7 because their fitness values are much larger than 3000, the upper limit of the used scale).

A further evidence of the generalisation ability provided by multi-optimization, is the fitness difference between training and test data (observable comparing Figure 5.1 and Figure 5.7). While this difference is notable for standard GP and DynOpEq, in the case of multi-optimization techniques training and test fitness are quite similar. In Figures 5.8 to 5.12 test fitness per generations is reported. In these plots the difference between standard GP and the other techniques has been shown and the three variants of multi-optimization GP have been compared. It is possible to see that multi-optimization that uses both individual size and error variance as auxiliary function does not perform better than multi-optimization with a single auxiliary function. This fact may seem counterintuitive. But in order to justify it, it is necessary to consider the effect of the simultaneous presence of two auxiliary functions: the NSGA-II algorithm could have some problems in finding optimal solutions in only 100 generations when the number of criteria increases. A hypothesis is that the increase of auxiliary functions can slow down the convergence of the algorithm towards better solutions. Experimental results confirmed the expectation about the generalization ability of the studied models: in particular it is possible to argue that the evolution process guided by only the error minimization leads to a solution that, at least in the considered problem, overfits training data. On the other hand, multi-optimization guided the evolution process towards the generation of individuals that perform less well over training data but do not overfit them (or at least overfit less than standard GP and DynOpEq). Finally it can be pointed out that DynOpEq is the technique that produces smallest individuals in terms of number of nodes (not shown) but this fact does not guarantee the absence of overfitting. This is a further confirmation of the fact that bloat and overfitting are two separate phenomena and that their mutual relationship is hard to understand [95, 110].

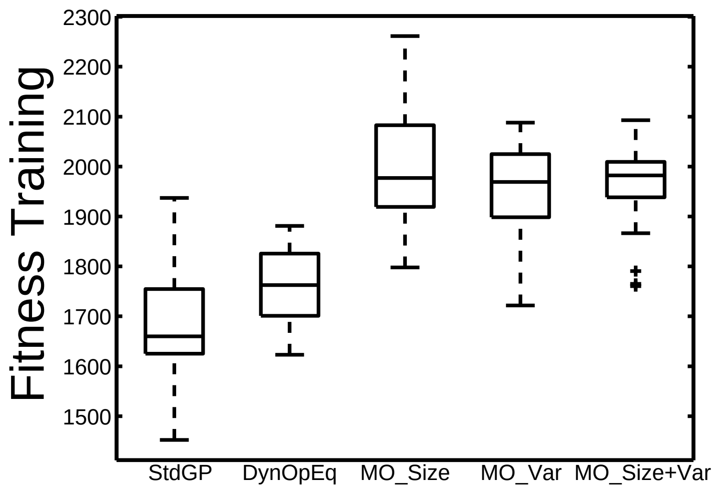


Figure 5.1: Boxplot of best training fitness.

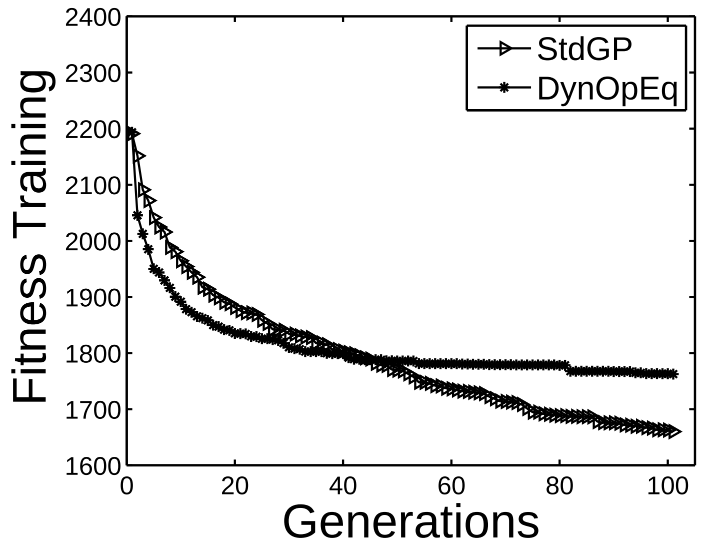


Figure 5.2: Comparison of the performances of Operator Equalisation and standard GP over the training data.

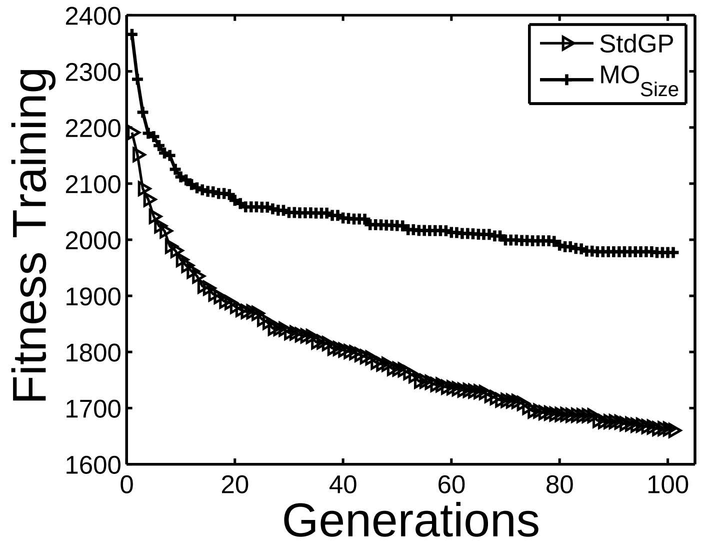


Figure 5.3: Comparison of the performances of MO_{Size} and standard GP over the training data.

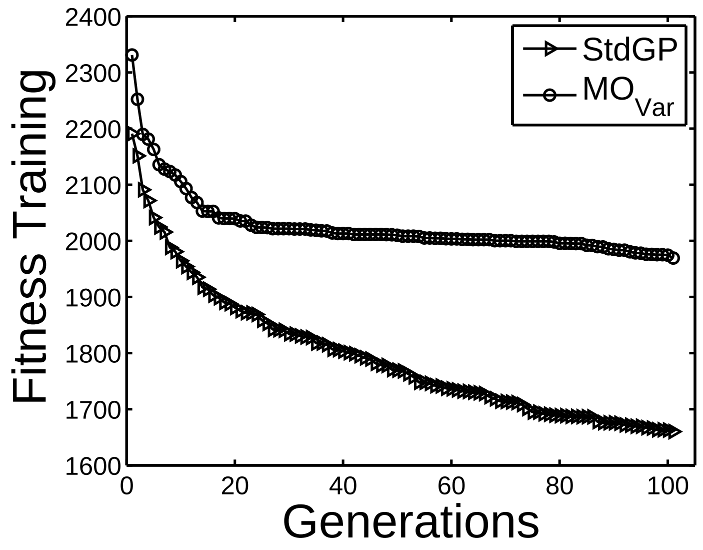


Figure 5.4: Comparison of the performances of MO_{Var} and standard GP over the training data.

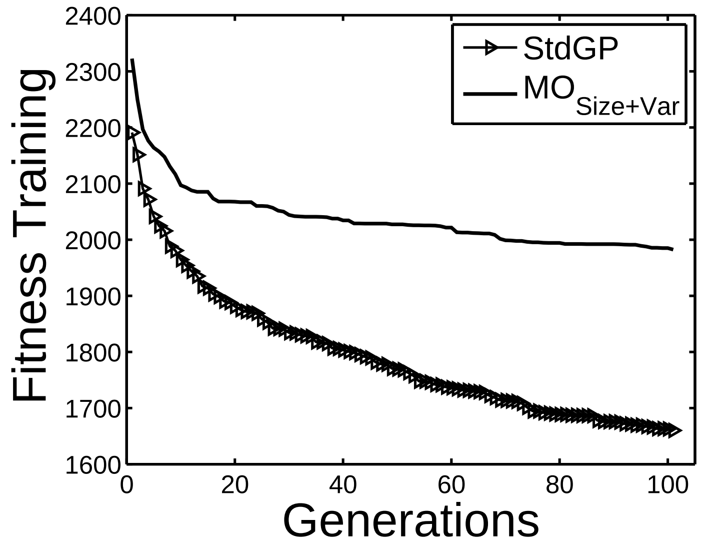


Figure 5.5: Comparison of the performances of $MO_{Size+Var}$ and standard GP over the training data.

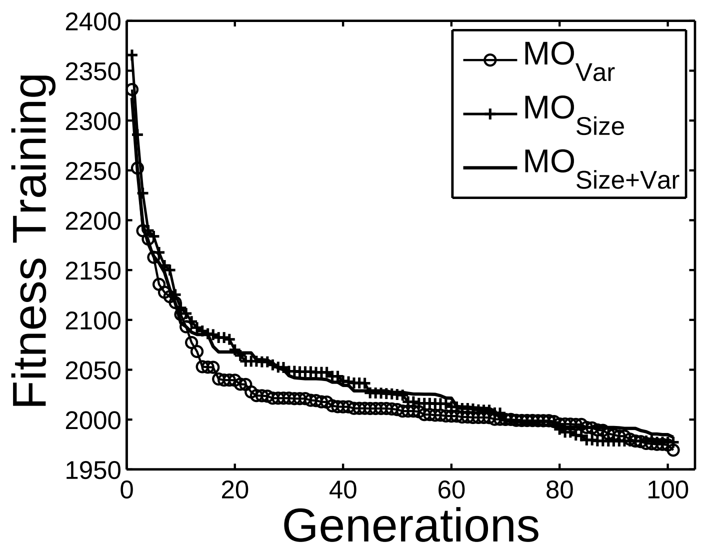


Figure 5.6: Comparison of the performances of all the multi-optimization technques over the training data.

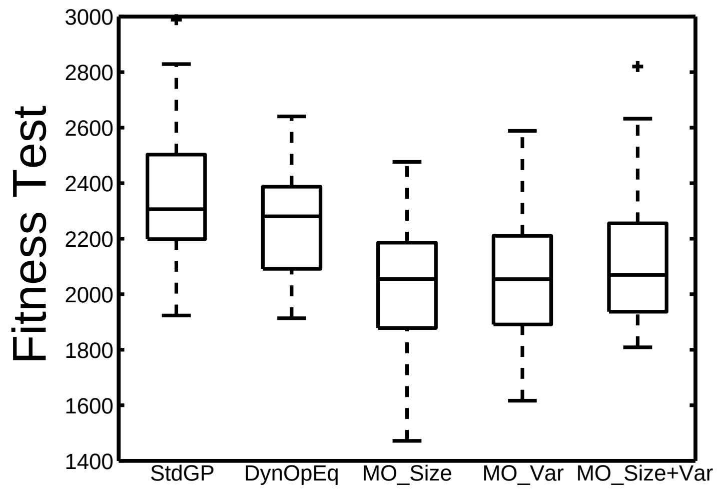


Figure 5.7: Boxplot of best test fitness.

Statistical validation

To confirm the qualitative considerations reported in the previous sections, a Kruskal-Wallis test of statistical significance has been performed . The aim of the test is to verify

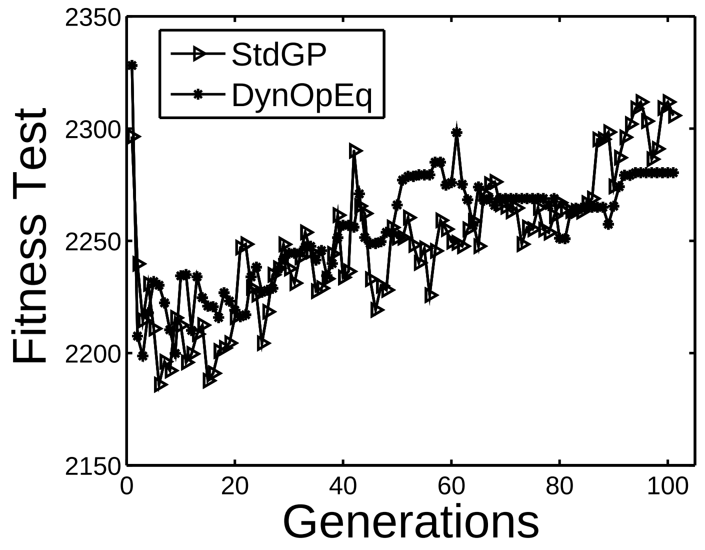


Figure 5.8: Comparison of the performances of Operator Equalisation and standard GP over the test data.

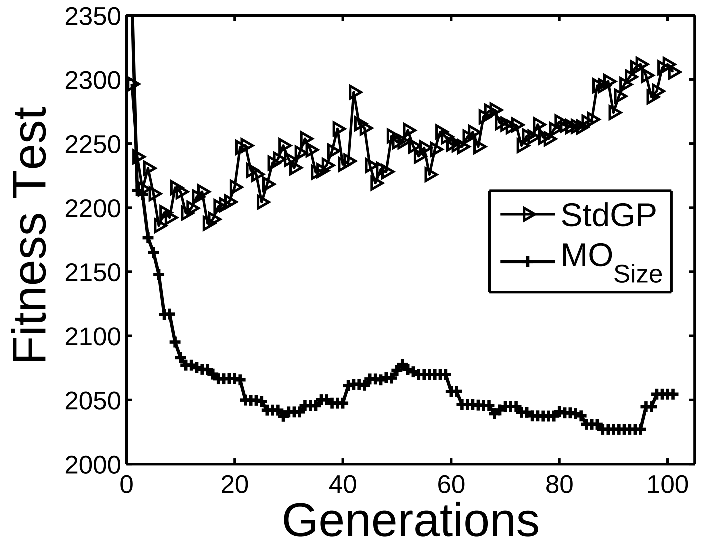


Figure 5.9: Comparison of the performances of MO_{Size} and standard GP over the test data.

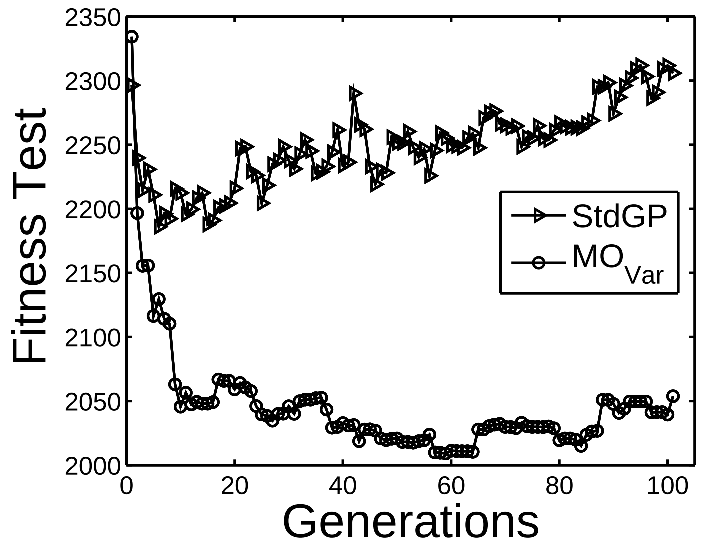


Figure 5.10: Comparison of the performances of MO_{Var} and standard GP over the test data.

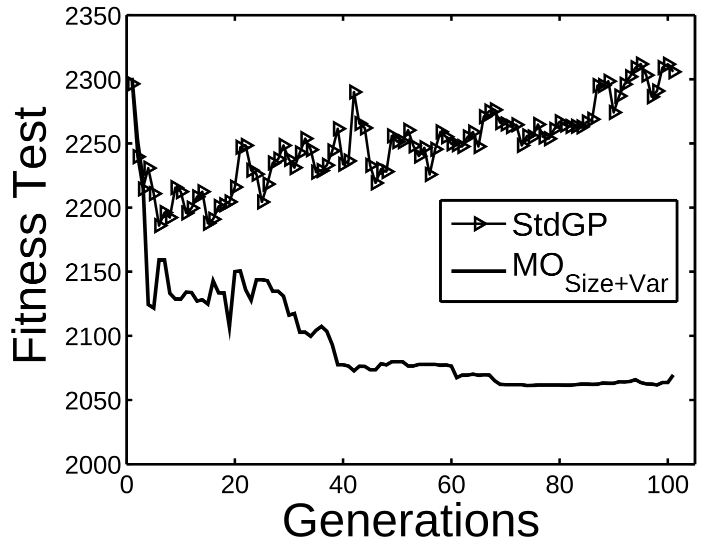


Figure 5.11: Comparison of the performances of $MO_{Size+Var}$ and standard GP over the test data.

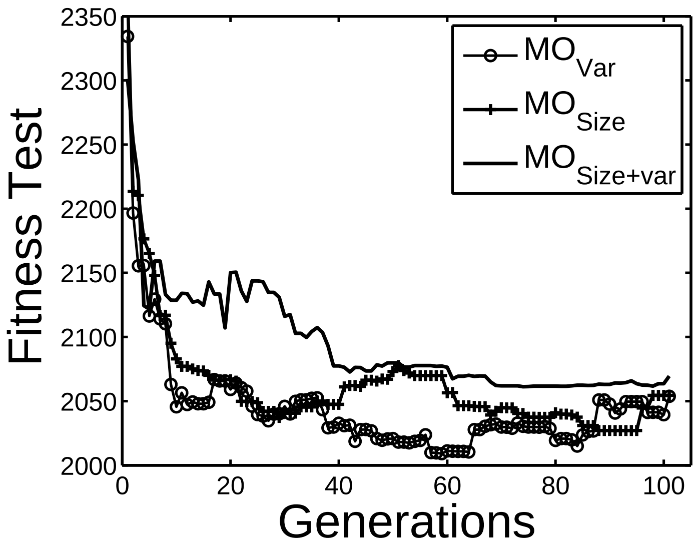


Figure 5.12: Comparison of the performances of all the multi-optimization technques over the test data.

whether the results obtained by the considered techniques (after 100 generations) are statistically different or not.

The results of the Kruskal-Wallis test performed over the *training* data are the following:

- The median of the fitness of standard GP after 100 generations compared to DynOpEq is statistically different with a confidence of 95%.
- The median of the fitness of standard GP after 100 generations compared to MO_{Size} is statistically different with a confidence of 95%.
- The median of the fitness of standard GP after 100 generations compared to MO_{Var} is statistically different with a confidence of 95%.
- The median of the fitness of standard GP after 100 generations compared to

$MO_{Size+Var}$ is statistically different with a confidence of 95%.

The results of the Kruskal-Wallis test performed over the *test* data are the following:

- When the median of the fitness of standard GP after 100 generations is compared to DynOpEq, the null hypothesis cannot be refuted with a significance level of 0.05.
- The median of the fitness of standard GP after 100 generations compared to MO_{Size} is statistically different with a confidence of 95%.
- The median of the fitness of standard GP after 100 generations compared to MO_{Var} is statistically different with a confidence of 95%.
- The median of the fitness of standard GP after 100 generations compared to $MO_{Size+Var}$ is statistically different with a confidence of 95%.

These results confirm the previous intuition: multi-optimization improves the ability of GP to generalize over unseen data.

5.7 Conclusions

Previous work has clearly shown that GP systems that produce smaller individuals do not necessarily produce solutions that generalize better. In particular in [95, 110] it has been shown that Dynamic Operator Equalisation (DynOpEq) is completely bloat free, but it produces solutions that, even though rather smaller than the ones produced by standard GP, are not able to generalize better than them.

In this chapter we have investigated a multi-objective GP system in which size minimization is used as an auxiliary optimization criterion (further than the usual root mean squared error). Rather surprisingly, results clearly show that that this system produces solutions with better generalization ability than standard GP and DynOpEq. Even though this is in agreement with several previously published results (see for instance [39, 105, 111]), it seems to contradict the fact that small size does not imply better generalization.

We hypothesize that the reason why multi-objective GP generalizes better then standard GP and DynOpEq does not reside in the fact that size minimization has been used as an objective. As a confirmation, we have also used a multi-objective GP system with a different auxiliary objective (variance of the errors) and we have found that its results on test data are comparable with the ones of multi-objective GP using size (the differences between these two systems are not statistically significant).

Thus, we hypothesize that better generalization ability is, so to say, an intrinsic characteristic of multi-optimization systems, which do not depend on the fact that size of solutions is explicitly minimized. But why does this happen? This is a question that definitely deserves further investigation.

CHAPTER 6

Generalization

Contents

6.1 Second chance GP

A method to increase the generalization ability of GP is proposed in this chapter. The idea consists in giving a second chance of mating to individuals belonging to "old" generations (hence the name of the method: "second chance GP"). Although original, the idea is inspired by well-known concepts such as short-term memory schemes, that have already been used in evolutionary computation so far. The issue of generalization has received a growing attention in the last few years (see [61] for a survey). For instance, in [37], a new GP system called Compiling GP System was introduced and its generalization ability was compared with that of other Machine Learning paradigms. In the same year, in [7], Banzhaf and coworkers showed the positive effect of an extensive use of the mutation operator on generalization in GP using sparse data sets. In [26] Da Costa and Landry have proposed a new GP model called Relaxed GP, showing its generalization ability and in [39] Gagné and coworkers have investigated two methods to improve generalization in GP: the selection of individuals using a three data sets methodology, and the application of parsimony pressure to reduce the size of the solutions. In [2] a theoretical analysis of GP from the perspective of statistical learning theory was proposed and the advantage of a parsimonious fitness using Vapnik-Chervonenkis theory was demonstrated. In [111], the authors proposed a multi-objective system where a measure called *order of nonlinearity* (that calculates the degree of the Chebyshev polynomial approximation of a function) is used together with other criteria and they demonstrated that this system is able to counteract overfitting. In [107] indicators of overfitting and complexity have been introduced and their mutual relationships have been investigated. There are many example of overfitting on real-life data (see, for example [4, 102, 107]).// The goal of this work is to define a simple-to-implement method to reduce overfitting. The idea is to re-use genetic material from older generations. Even though similar to the concept of "short-term" memory, which is typical of Tabu Search [41], the proposed method is new. Individuals belonging to "old" generations are allowed to participate again to the selection process at given time intervals, thus giving them a second chance to take part in mating. For this reason, the proposed method is called "second chance GP". In some senses, the method simulates the idea that partners do not have to have all the same age, but partners of different ages can mate and produce offspring. Furthermore, differently from the majority of the previously published related contributions, the proposed approach is focused specifically on generalization, with the idea that "old" individuals may be less specialized than "young" ones on training data, and thus potentially more general.

6.2 Method

The main idea of the proposed "second chance GP" method is to insert genetic material from older populations into the current population, replacing the worst individuals in the current population. To accomplish this goal, every k generations (where $k \in \mathbb{N}$ and $k > 1$), the worst $p_r\%$ individuals in the population (where p_r is the *replacement pressure*) are replaced. The individuals that replace them are extracted from the population of k generations before the current one (for this reason, k is called *refresh rate*), and they are chosen from that population using exactly the same selection method used by the standard algorithm. The name "second chance" is inspired by the fact that, in this way, an individual can participate in at least two selection phases, increasing its probability of being selected. The pseudo code of the "second chance GP" method is given in Algorithm 1.

Motivations for introducing this method are the following. First of all it is possible hypothesize that, generation by generation, the GP individuals become more and more specialized. Even if this behavior can be a good one, the GP individuals could also overfit training data. The insertion of earlier - and probably less specialized - individuals with a good fitness can allow the population diversity to increase, reducing at the same time the risk of overfitting training data. The second motivation is that the processes of selection, crossover and mutation of GP can discard (the former one) or

Algorithm 1 The *second chance GP* algorithm.

```
begin
   initialize_GP();
   number_of_generations $:= 0$;
   saved_population $:= \emptyset$;
   while $\neg$termination_criteria() do
         execute_gp_population();
         number_of_generations := number_of_generations $+ 1$;
         if mod(number_of_generations, $k$) $= 0$
           then
                tmp_population := current_population;
                merge(current_population, saved_population);
                saved_population := tmp_population;
         fi
   od

where

proc merge(P$_1$, P$_2$) $\equiv$
   for $i := 0$ to Population_Size $* (p_r/100)$ do
       remove_worst(P$_1$);
   od
   for $i := 0$ to Population_Size $* (p_r/100)$ do
       x = selection(P$_2$);
       add_individual(P$_1$, x);
   od
end
```

disrupt (the latter ones) good genetic material. Allowing a reinsertion, the probability of good genetic material to be propagated increases. The third motivation is that, also on training data themselves, the evolutionary process may lead the population to convergence towards local optima. The insertion of earlier individuals with a good fitness, while allowing the population diversity to increase, should also reduce the risk of premature convergence and stagnation in local optima.

The “second chance GP” method is characterized by the presence of two parameters: the refresh rate k and the replacement pressure p_r. Thus, their effect has to be investigated. An hypothesis may be that when the parameter k is low, the effect is to increase the probability to insert into the current population individuals with a good fitness that can positively affect the evolution process. On the other hand, a low k value may result in inserting genetic material that does not help in better exploring the search space, since it may be not significantly different with respect to the existing one. An high k value can help in solving the latter issue, but can generate a new problem: the individuals of old populations can have a much worse fitness than the current ones and their reinsertion may produce no effects on the search process. The second parameter p_r can also influence the behavior of the search process. In particular, a low p_r value may result in inserting not enough individuals, with a negligible effect on the evolutionary process. A high p_r value can solve the latter problem but, if too high, can disrupt what has been learned so far.

6.3 Experiments

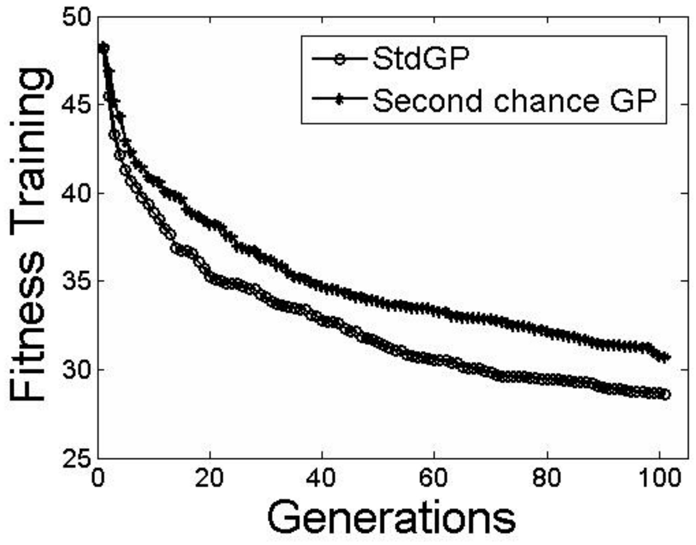
Experimental setting. A total of 30 runs were performed both with standard GP and second chance GP (in the latter case, 30 independent runs have been executed for each considered combination of the *refresh rate* and *replacement pressure* parameters). All the runs used populations of 200 individuals allowed to evolve for 100 generations. Tree initialization was performed with the Ramped Half-and-Half method [57] with a maximum initial depth of 6 and a maximum depth of 17. The function set contained the four binary operators $+$, $-$, $*$, and $/$, protected as in [57]. The terminal set contained 241 floating point variables for the %F dataset, 626 floating point variables for the LD50 dataset and 626 floating point variables for the %PPB dataset. In all these test problems no random constants were added to the terminal set. Because the cardinalities of the function and terminal sets were so different, a balanced choice between functions and terminals has been imposed when selecting a random node. Fitness was calculated as the root mean squared error between outputs and targets. Selection for reproduction used tournaments of size 4. The reproduction (replication) rate was 0.95, meaning that each selected parent has a 5% chance of being copied to the next generation instead of being engaged in breeding. Standard tree mutation and standard crossover (with uniform selection of crossover and mutation points) were used with probabilities of 0.05 and 0.95, respectively. The new random branch created for mutation has maximum depth equal to 6. Selection for survival used elitism (i.e. unchanged copy of the best individual in the next population). Regarding the parameters introduced by the proposed method, *refresh rates* with values equal to 5, 10, 20 and *selection pressures* with values equal to 25, 50, 75 have been considered. All the combinations of these values have been used in the experimental phase.

Additional experiments were performed with more “aggressive” parameters. In particular with tournament size 10, crossover rate 0.9 and mutation rate 0.1. The results are not reported but the behaviour of the second chance method was not different from the one presented here.

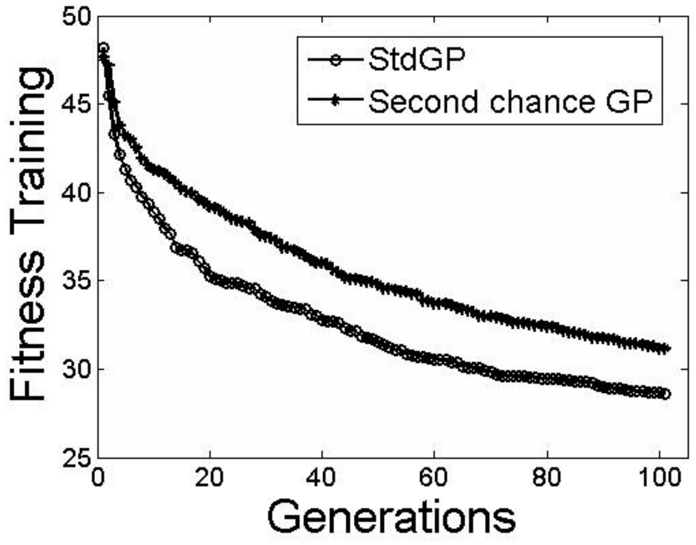
Experimental results. For all the considered test problems, performances obtained over the training and the test sets by standard GP and second chance GP for every combination of the considered values for the *refresh rate* and the *replacement* *pressure* parameters have been analyzed. In the first part of the experimental phase a *replacement pressure* equal to 50% (hence the half of the population that includes the worst individuals is involved in the replacement process) has been considered and the role of the *refresh rate* parameter has been analyzed. The results of this experimental phase are reported in Figures 6.1, 6.2 and 6.3.

From now on, the term “fitness on the training set” (or more simply “training fitness” or “fitness training”) refers to the root mean squared error between targets and outputs on training data and used by the selection algorithm. On the other hand, the term “fitness on the test set” (or more simply “test fitness” or “fitness test”) indicates the root mean squared error of the individual with the best training fitness in the population, calculated between targets and outputs on the test data.

Figure 6.1 reports the median of the best training fitness and the median of the test fitness for standard GP and second chance GP for the %F dataset for different values of the *refresh rate*.

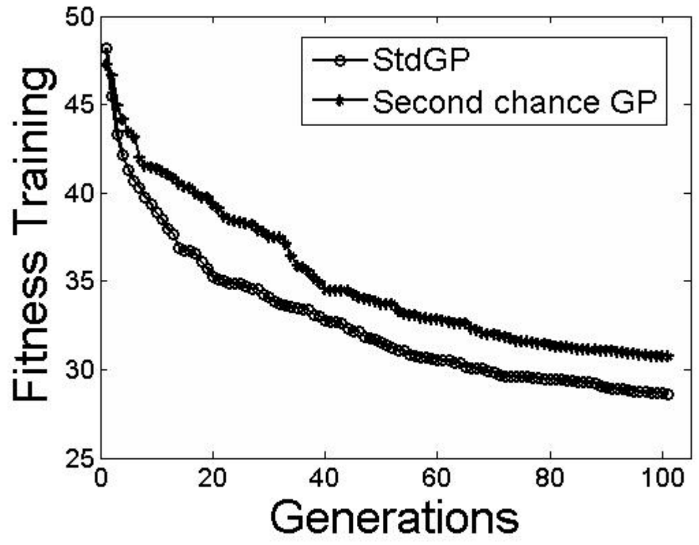


(a)

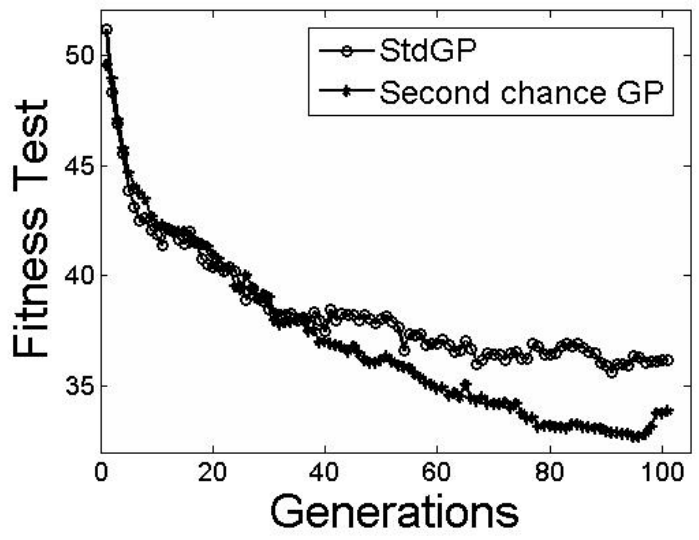


(b)

Figure 6.1: Test Function: *bioavailability (%F)*. Plots (a) and (d): *Refresh rate = 5*. Plots (b) and (e): *Refresh rate = 10*. Plots (c) and (f): *Refresh rate = 20*. Plots (a), (b) and (c): median of the best training fitness over the performed 30 runs against generations. Plots (d), (e) and (f): median of the test fitness over the performed 30 runs against generations.

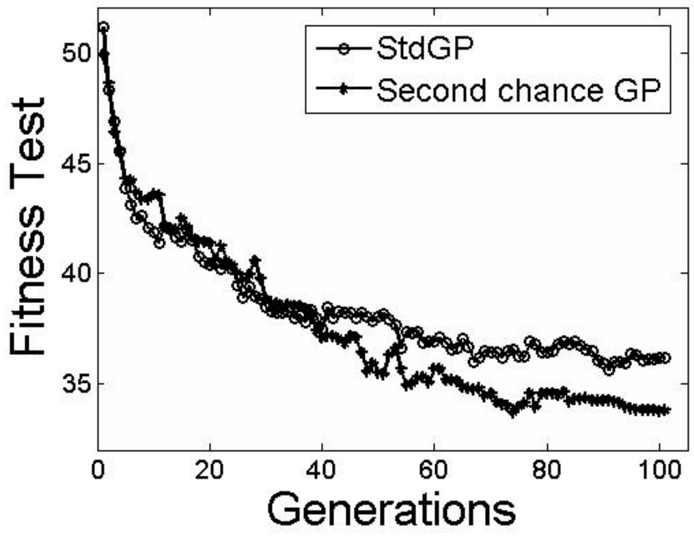


(c)

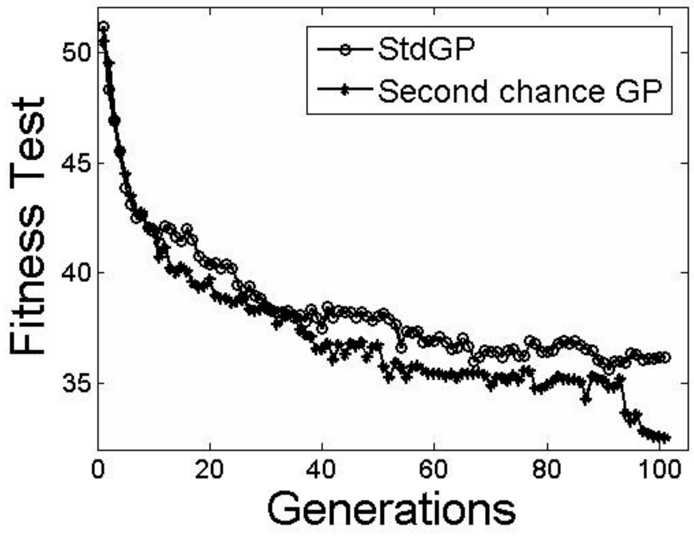


(d)

Figure 6.1: Test Function: *bioavailability (%F)*. Plots (a) and (d): *Refresh rate = 5*. Plots (b) and (e): *Refresh rate = 10*. Plots (c) and (f): *Refresh rate = 20*. Plots (a), (b) and (c): median of the best training fitness over the performed 30 runs against generations. Plots (d), (e) and (f): median of the test fitness over the performed 30 runs against generations.



(e)

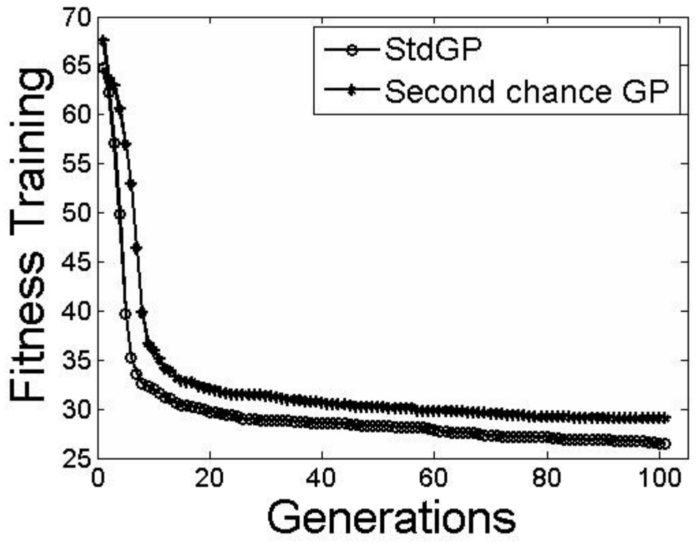


(f)

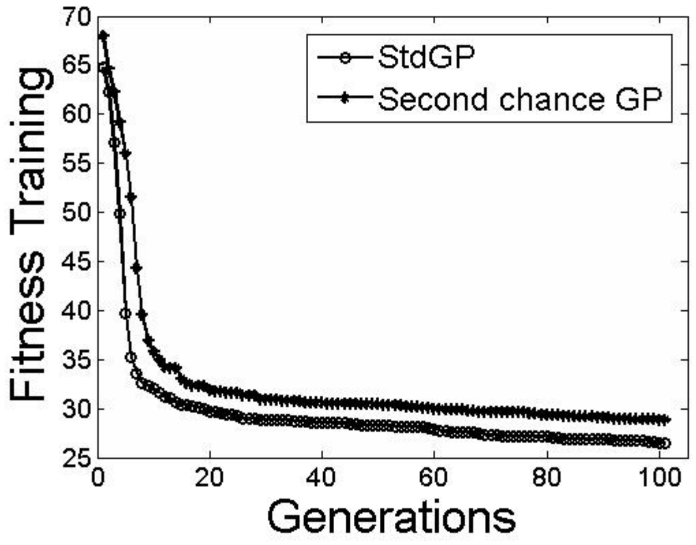
Figure 6.1: Test Function: *bioavailability (%F)*. Plots (a) and (d): *Refresh rate = 5*. Plots (b) and (e): *Refresh rate = 10*. Plots (c) and (f): *Refresh rate = 20*. Plots (a), (b) and (c): median of the best training fitness over the performed 30 runs against generations. Plots (d), (e) and (f): median of the test fitness over the performed 30 runs against generations.

In Figure 6.1, as in the subsequent ones, plots (a), (b) and (c) report the results on the training set, while plots (d), (e) and (f) report the results on the test set. Plots (a) and (d) (respectively (b) and (e) and (c) and (f)) report the results for the *refresh rate* equal to 5 (respectively 10 and 20). Standard GP performs better than second chance GP over training instances for all the considered *refresh rates*. Regarding the fitness on the test set and considering *refresh rates* equal to 5 and 10, second chance GP produces better performances than standard GP. Furthermore, considering test fitness from generation 60 to 100, the two techniques produce statistically different fitness values. In these experiments, as in all the ones presented later, the statistical significance of the results has been evaluated with the Kruskal-Wallis method [24], considering a confidence of 99%. Regarding the *refresh rate* equal to 20, there is not a significant difference in terms of test fitness between standard GP and second chance GP. These results confirm the initial expectation: a low *refresh rate* can help in maintaining a high diversity inside the population and reducing the risk of overfitting. Replacing bad individuals with individuals that are selected from too old generations does not produce the same effect because the evolution has already reached a more advanced stage. In such a situation, older individuals, that probably have a much worse fitness than the current best individuals, are penalized by tournament selection. In this situation, old individuals weakly participate in the mating process.

The second test problem that has been considered is %PPB. As already pointed out, for instance, in [107], in this problem standard GP has no overfitting, so it is possible to expect that the performances of standard GP and second chance GP to be similar. This fact is confirmed by Figure 6.2, where median values of the best training fitness and of the test fitness are reported.

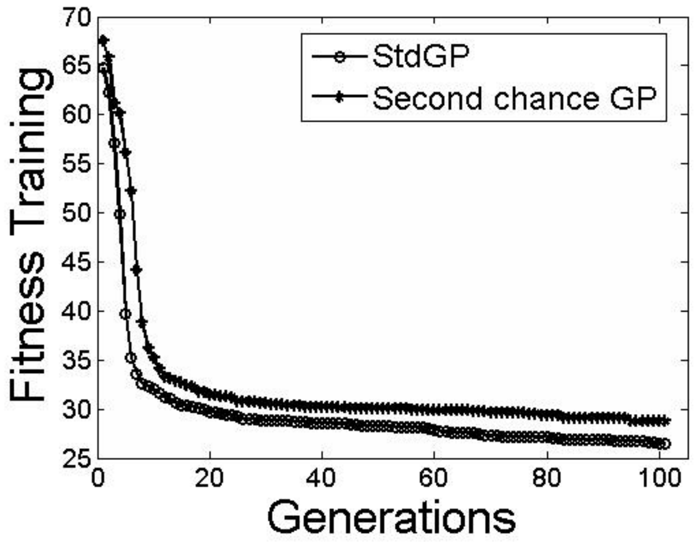


(a)

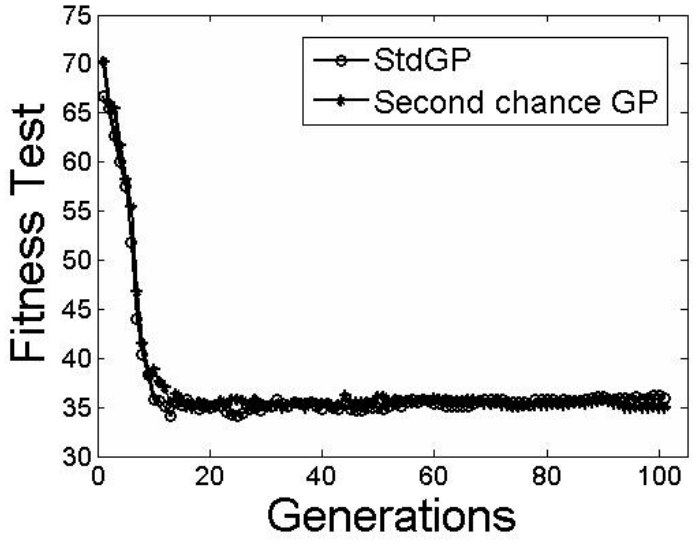


(b)

Figure 6.2: Test Function: *%PPB*. Plots (a) and (d): *Refresh rate = 5*. Plots (b) and (e): *Refresh rate = 10*. Plots (c) and (f): *Refresh rate = 20*. Plots (a), (b) and (c): median of the best training fitness over the performed 30 runs against generations. Plots (d), (e) and (f): median of the test fitness over the performed 30 runs against generations.

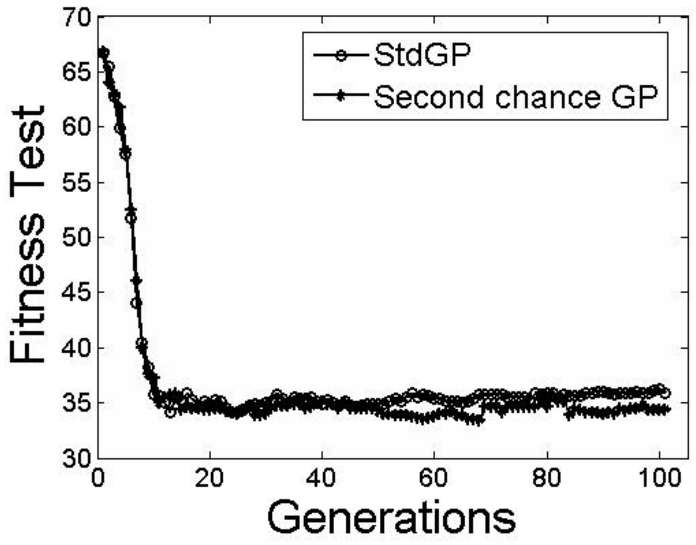


(c)

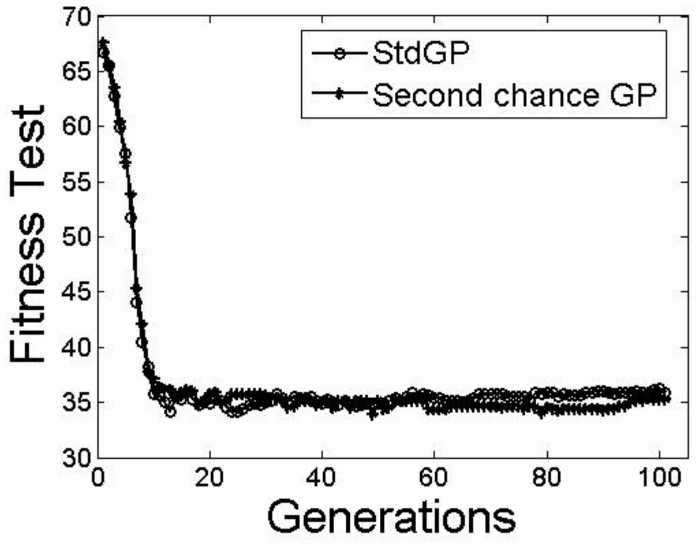


(d)

Figure 6.2: Test Function: *%PPB*. Plots (a) and (d): *Refresh rate = 5*. Plots (b) and (e): *Refresh rate = 10*. Plots (c) and (f): *Refresh rate = 20*. Plots (a), (b) and (c): median of the best training fitness over the performed 30 runs against generations. Plots (d), (e) and (f): median of the test fitness over the performed 30 runs against generations.



(e)

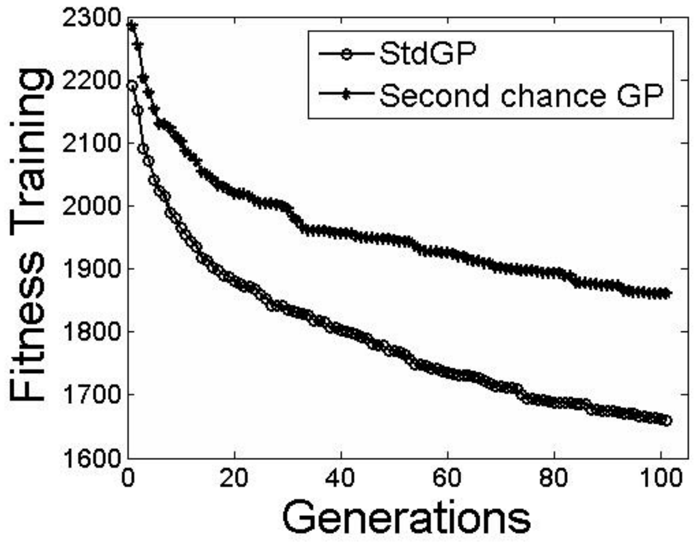


(f)

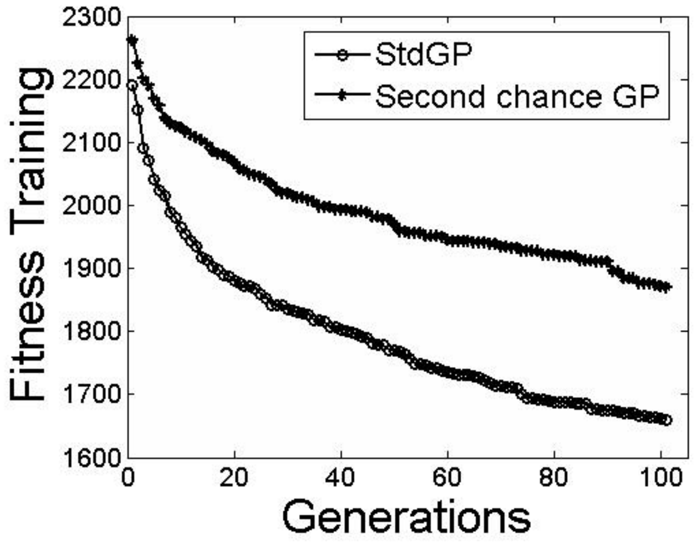
Figure 6.2: Test Function: *%PPB*. Plots (a) and (d): *Refresh rate = 5*. Plots (b) and (e): *Refresh rate = 10*. Plots (c) and (f): *Refresh rate = 20*. Plots (a), (b) and (c): median of the best training fitness over the performed 30 runs against generations. Plots (d), (e) and (f): median of the test fitness over the performed 30 runs against generations.

Indeed, for both training and test sets, the considered techniques produce similar results. In particular, it is possible to informally observe that the curves representing test fitness almost completely "overlap" during all the studied generations.

The last considered test problem, LD50, is very challenging for the proposed method. In fact, as already pointed out in [107], standard GP overfits training data since the earliest generations. Figure 6.3 reports the median values of the best training fitness and of the test fitness for standard GP and second chance GP for the LD50 dataset for different values of the *refresh rate*.

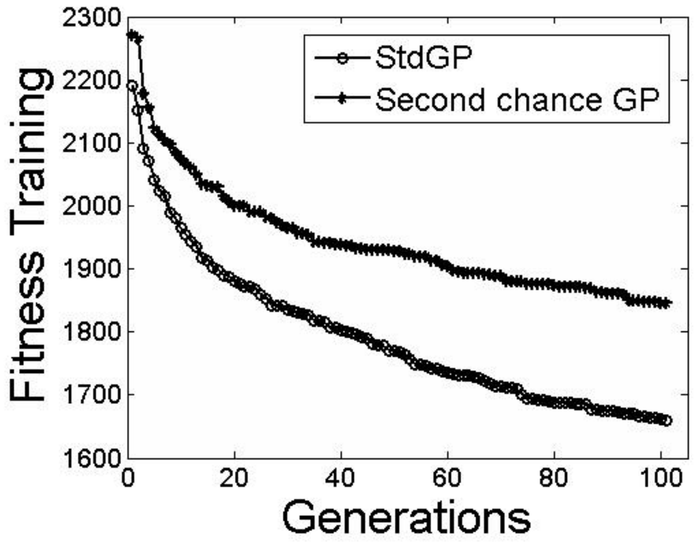


(a)

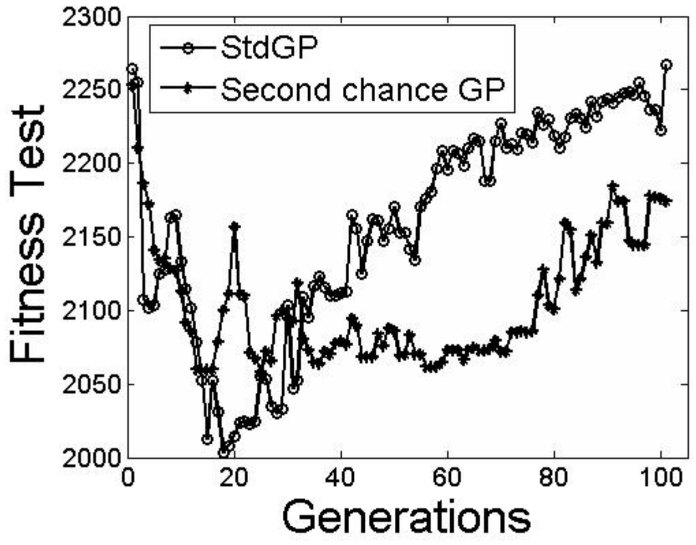


(b)

Figure 6.3: Test Function: *LD50*. Plots (a) and (d): *Refresh rate = 5*. Plots (b) and (e): *Refresh rate = 10*. Plots (c) and (f): *Refresh rate = 20*. Plots (a), (b) and (c): median of the best training fitness over the performed 30 runs against generations. Plots (d), (e) and (f): median of the test fitness over the performed 30 runs against generations.

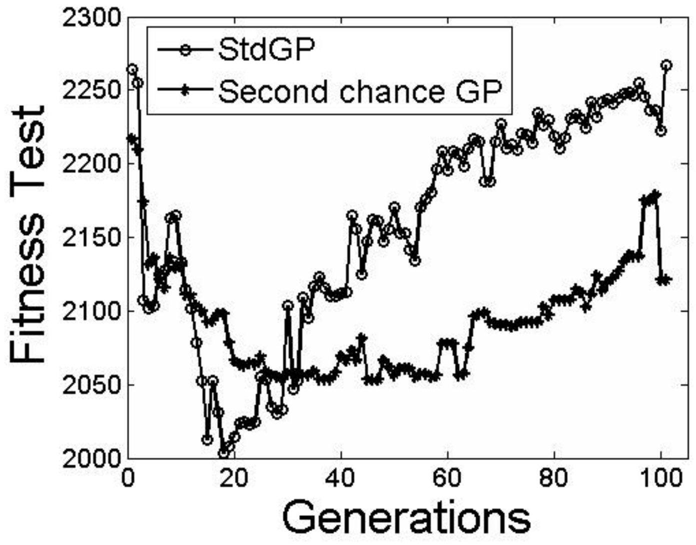


(c)

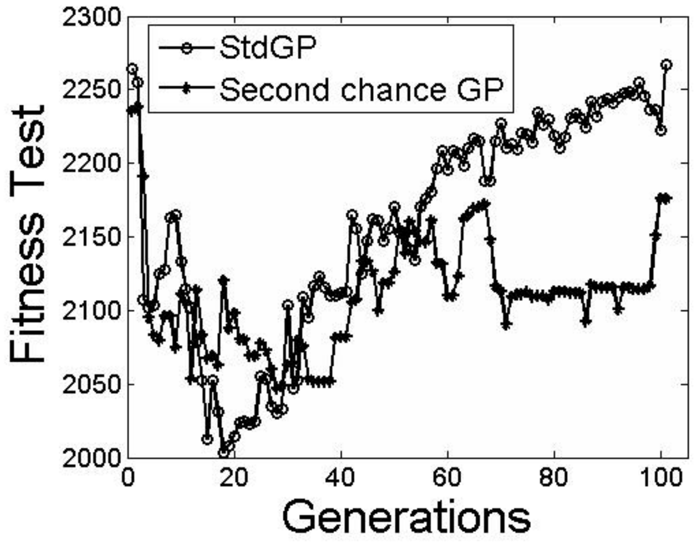


(d)

Figure 6.3: Test Function: *LD50*. Plots (a) and (d): *Refresh rate = 5*. Plots (b) and (e): *Refresh rate = 10*. Plots (c) and (f): *Refresh rate = 20*. Plots (a), (b) and (c): median of the best training fitness over the performed 30 runs against generations. Plots (d), (e) and (f): median of the test fitness over the performed 30 runs against generations.



(e)



(f)

Figure 6.3: Test Function: *LD50*. Plots (a) and (d): *Refresh rate = 5*. Plots (b) and (e): *Refresh rate = 10*. Plots (c) and (f): *Refresh rate = 20*. Plots (a), (b) and (c): median of the best training fitness over the performed 30 runs against generations. Plots (d), (e) and (f): median of the test fitness over the performed 30 runs against generations.

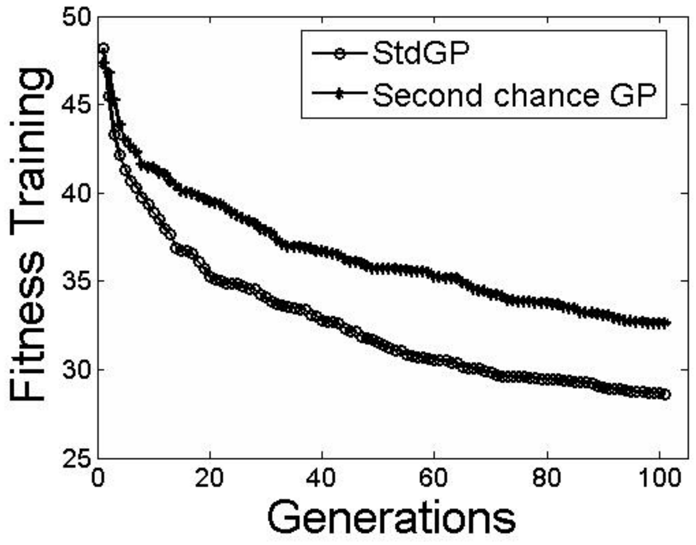
The focus is first on training fitness. The performances of second chance GP on the training set are poor compared to standard GP. Considering that the interest is in achieving better results over test instances, and keeping in mind that overfitting is a serious issue on this problem for standard GP [107], the results obtained on the training set could even be positively interpreted: having a worse fitness with the proposed method may indicate that the learning process did not specialize too much on the training data. This hypothesis is confirmed by the results obtained on the test set: even more markedly than for the %F problem, second chance GP outperforms standard GP on the test set. Furthermore, once again the best results have been obtained with the smallest *refresh rate* values. For the *refresh rate* equal to 5, 10 and 20, the Kruskal-Wallis method indicates that the results of second chance GP are statistically different from the ones obtained by standard GP (see Table 6.1). By observing the curves of the test fitness, it is possible to see that for standard GP the test fitness tends to deteriorate since the very beginning of the run (the trend of the curves of the error of standard GP on the test set is prevalently increasing), while it is not the case for second chance GP, where the increasing of error on the test data is much slower. Thus, the results obtained with the LD50 problem confirm and strengthen the ones obtained with %F: a low *refresh rate* can help in maintaining diversity inside the population and reducing the risk of overfitting. Replacing individuals after too many generations does not produce the same effect because evolution has already reached an advanced stage, so individuals coming from older generations are weakly involved in the evolutionary process.

In the second part of the experimental phase the *refresh rate* has been set to 5 and three different *replacement pressure* values have been considered: 25%, 50% and 75%. The results of this experimental phase are reported in Figures 6.4, 6.5 and 6.6.

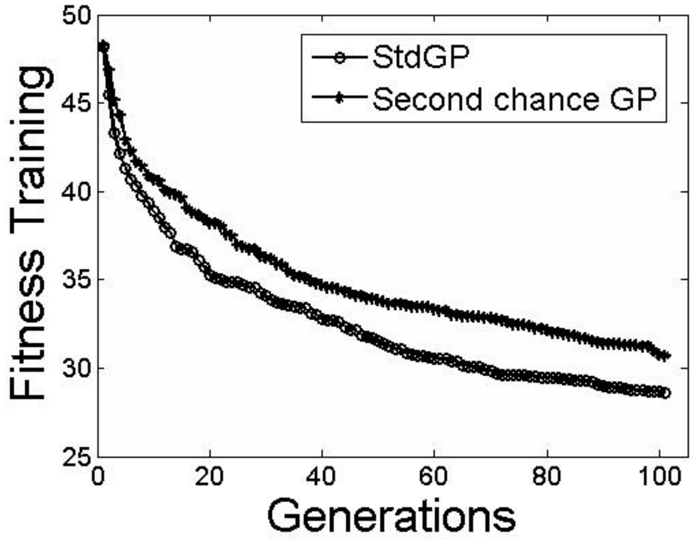
Refresh rate	**5**	**10**	**20**
%F	$4.986 \cdot 10^{-6}$	$3.788 \cdot 10^{-5}$	0.0005
%PPB	0.5947	$< 10^{-9}$	$< 10^{-9}$
LD50	$3.608 \cdot 10^{-8}$	$< 10^{-9}$	$3.149 \cdot 10^{-7}$

Table 6.1: The p-values for the Kruskal-Wallis test for the different problems when the *replacement pressure* is fixed to 50%.

Figure 6.4 reports the median values of the best training fitness and of the test fitness for standard GP and second chance GP for the %F dataset for different values of the *replacement pressure*.

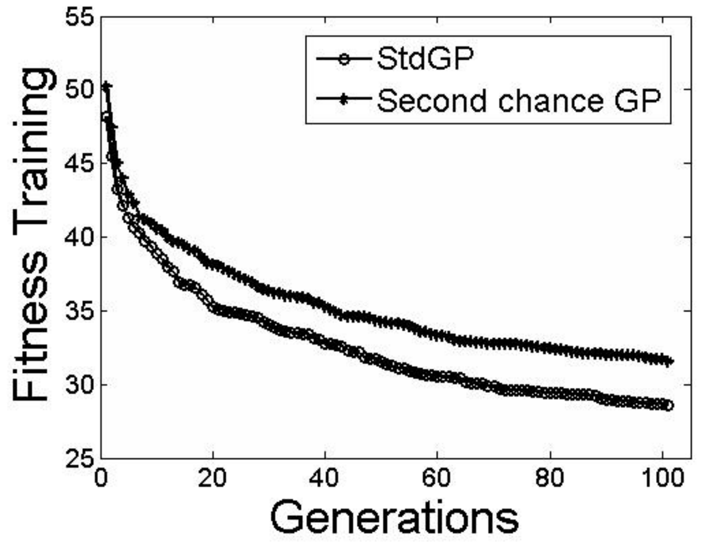


(a)

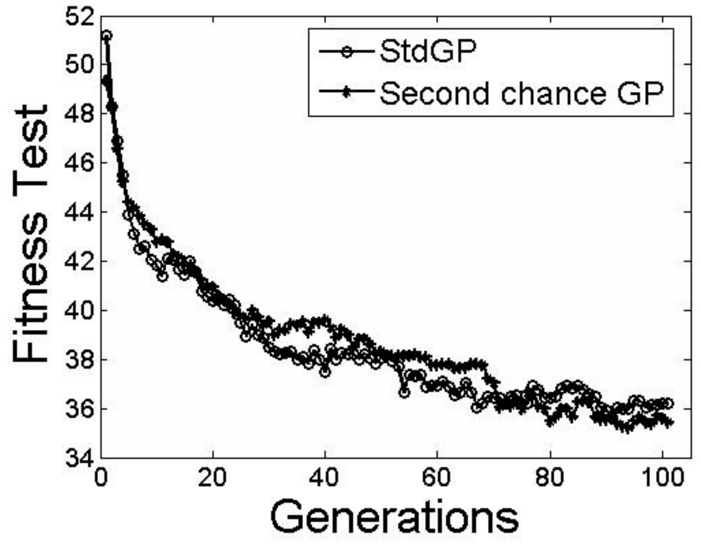


(b)

Figure 6.4: Test Function: *bioavailability (%F)*. Plots (a) and (d): *Replacement pressure = 25%*. Plots (b) and (e): *Replacement pressure = 50%*. Plots (c) and (f): *Replacement pressure = 75%*. Plots (a), (b) and (c): median of the best training fitness over the performed 30 runs against generations. Plots (d), (e) and (f): median of the test fitness over the performed 30 runs against generations.

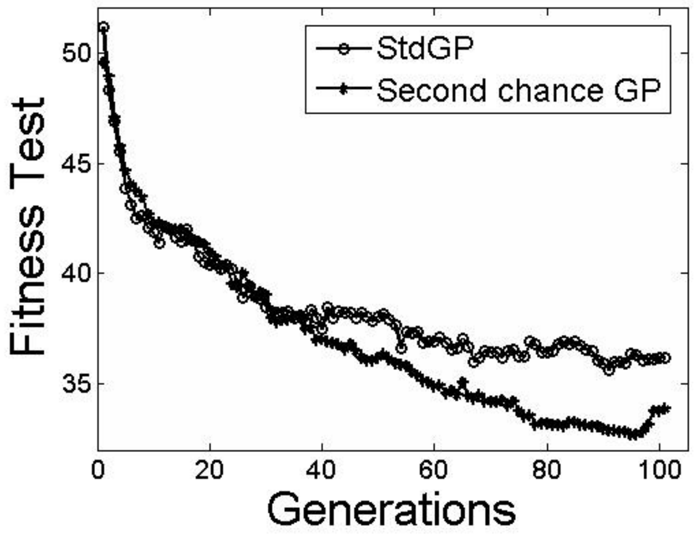


(c)

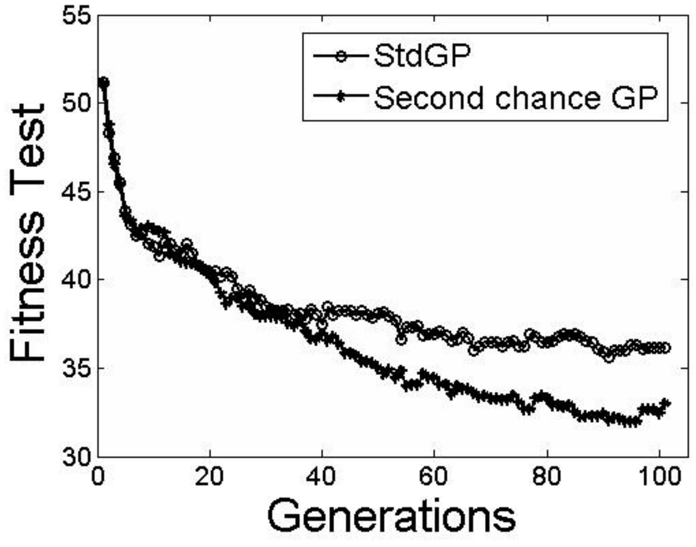


(d)

Figure 6.4: Test Function: *bioavailability (%F)*. Plots (a) and (d): *Replacement pressure = 25%*. Plots (b) and (e): *Replacement pressure = 50%*. Plots (c) and (f): *Replacement pressure = 75%*. Plots (a), (b) and (c): median of the best training fitness over the performed 30 runs against generations. Plots (d), (e) and (f): median of the test fitness over the performed 30 runs against generations.



(e)



(f)

Figure 6.4: Test Function: *bioavailability (%F)*. Plots (a) and (d): *Replacement pressure = 25%*. Plots (b) and (e): *Replacement pressure = 50%*. Plots (c) and (f): *Replacement pressure = 75%*. Plots (a), (b) and (c): median of the best training fitness over the performed 30 runs against generations. Plots (d), (e) and (f): median of the test fitness over the performed 30 runs against generations.

Standard GP performs better than second chance GP over training instances for all the considered *replacement pressures*. Regarding the test fitness, second chance GP produces better results. In particular the best results are obtained with high values of the *replacement pressure* parameter. The Kruskal-Wallis method indicates that the differences between the results on the test set returned by standard GP and second chance GP are statistically significant for *replacement pressures* of 50% and 75% (see Table 6.2). These results were expected: a high *replacement pressure* means that a large number of individuals in the current population is replaced by older and less specialized individuals.

Figure 6.5 reports the median values of the best training fitness and of the test fitness for the %PPB dataset.

In this case, as expected, there is not a significant difference between the results obtained by standard GP and the ones of second chance GP except for *replacement pressure* 75% (see Table 6.2).

Figure 6.6 reports the results for the LD50 dataset.

These results confirm, in an even more visible way, those previously obtained with the %F dataset: while standard GP performs better over training instances, second chance GP outperforms standard GP on the test set. This is particularly true for high values of the *replacement pressure* parameter. The Kruskal-Wallis test indicates that the differences between the results returned by standard GP and the ones returned by second chance GP on the test set are all statistically significant for the LD50 dataset and for *replacement pressures* of 25%, 50% and 75% (see Table 6.2).

To summarize, presented experimental results show that small *refresh rates* and large *replacement pressures* allow to reduce the risk of overfitting in those studied

Replacement pressure	**25%**	**50%**	**75%**
%F	0.5274	$4.986 \cdot 10^{-6}$	$7.493 \cdot 10^{-7}$
%PPB	0.0125	0.5947	0.0006
LD50	0.0004	$3.608 \cdot 10^{-8}$	$< 10^{-9}$

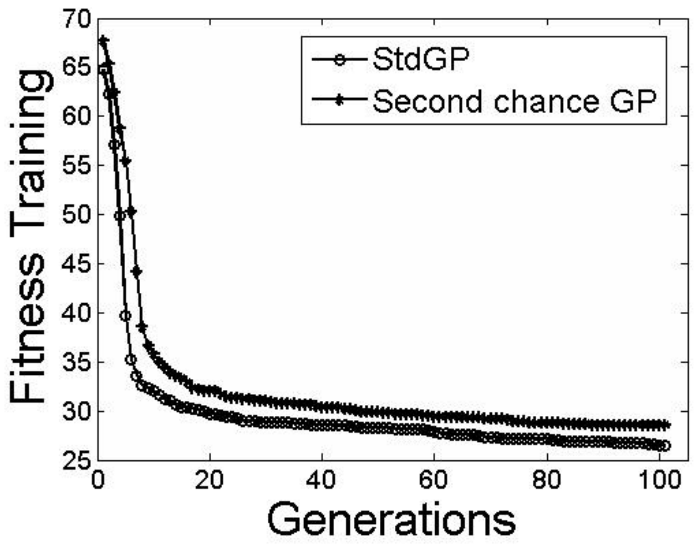
Table 6.2: The p-values for the Kruskal-Wallis test for the different problems when the *refresh rate* is fixed to 5.

problems where overfitting is an issue for standard GP. In the considered problem where standard GP has no overfitting, the proposed method produces results that are comparable to the ones returned by standard GP.

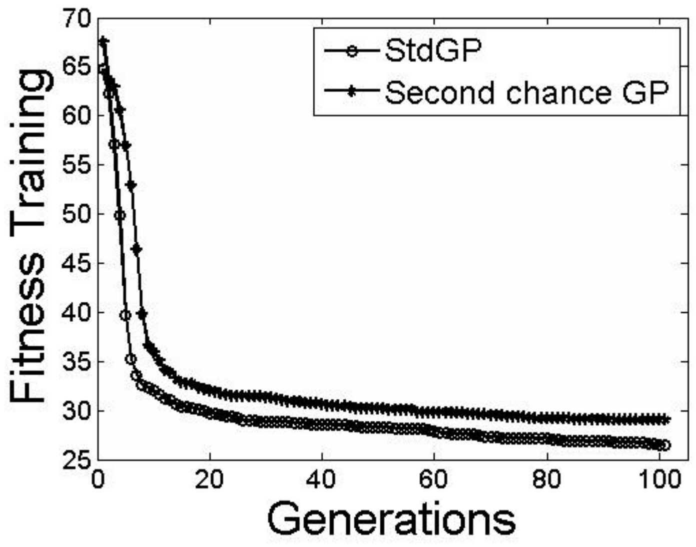
6.4 Conclusions and Future Works

A new GP method has been presented in this chapter. It is based on the idea of re-using "good" but "old" genetic material in the current population, giving them a second chance to survive an mate (hence the name of the proposed method: second chance GP). It is inspired by well-known concepts such as various forms of short-term memory scheme. The main motivation for introducing this method has been that we expect old individuals to be less specialized on training data than new ones, and thus we expect that re-using them can help reducing the risk of overfitting. This expectation has been confirmed by the results of a set of experiments that have been performed on three complex real-life regression problems from the field of drug discovery. In one of these three problems standard GP is seriously affected by overfitting, in the second one standard GP slightly overfits training data, while in the third one it does not have overfitting. Interestingly, second chance GP outperforms standard GP on test data for the first two problems, while its results are comparable with the ones of standard GP for the third one. These results seem to suggest the suitability of second chance GP. Furthermore, experiments have allowed us to investigate how two important parameters (*refresh rate* and *replacement pressure*) affect the performance of second chance GP. Experiments have clearly indicated that small *refresh rates* and large *replacement pressures* have allowed us to obtain the best results on test data for all the studied problems.

Future work includes the experimental validation of second chance GP on a wider set of test problems of different nature (for instance, could investigate the effectiveness of the method on problems that are not regressions). Furthermore, we plan to integrate the second chance GP algorithm with other methods that have been proposed so far to reduce the risk of overfitting in GP, like for instance the methods proposed in [86], in [111] or in [107].

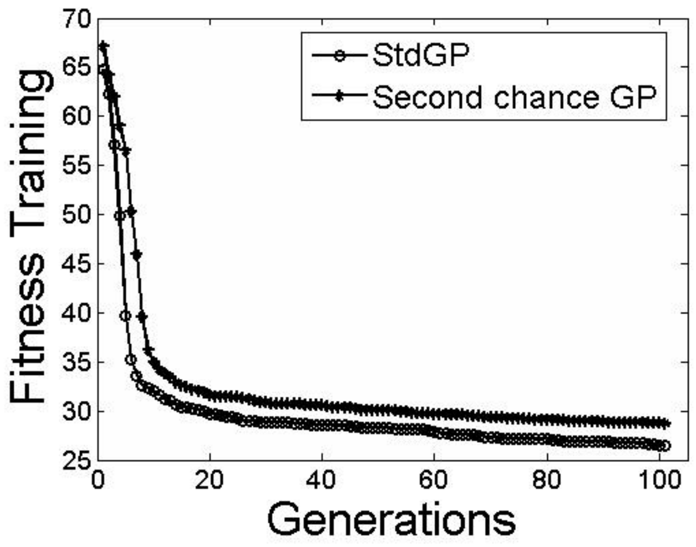


(a)

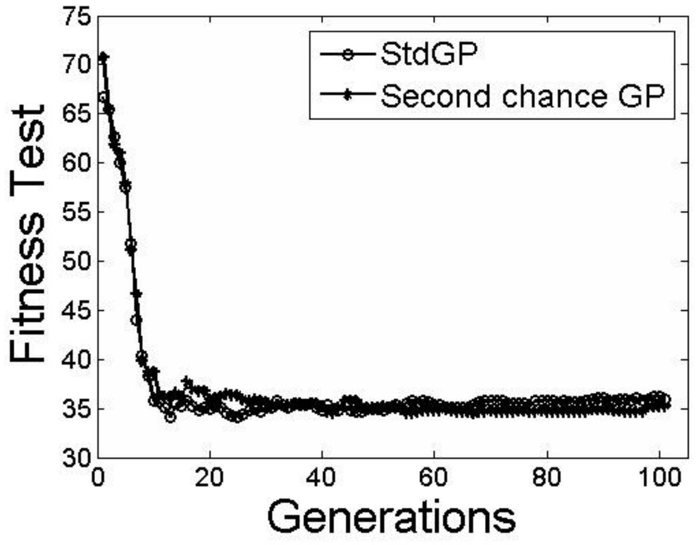


(b)

Figure 6.5: Test Function: *%PPB*. Plots (a) and (d): *Replacement pressure = 25%*. Plots (b) and (e): *Replacement pressure = 50%*. Plots (c) and (f): *Replacement pressure = 75%*. Plots (a), (b) and (c): median of the best training fitness over the performed 30 runs against generations. Plots (d), (e) and (f): median of the test fitness over the performed 30 runs against generations.

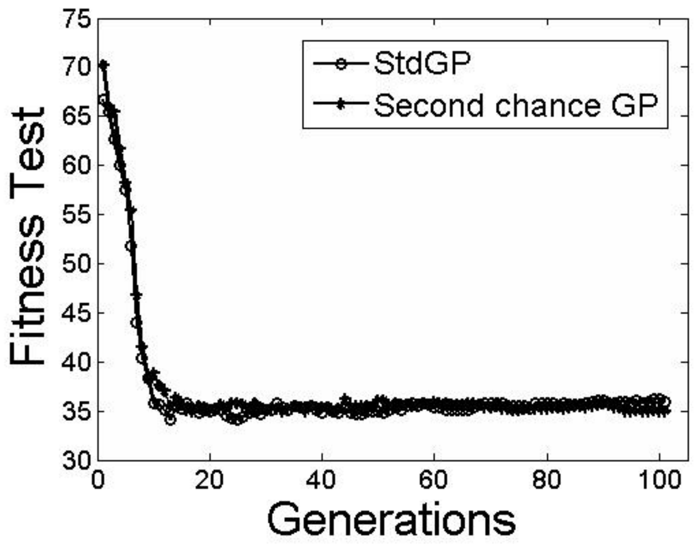


(c)

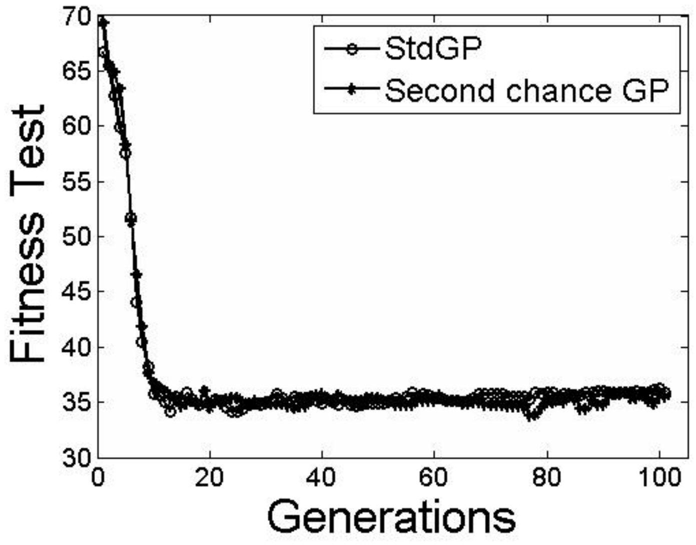


(d)

Figure 6.5: Test Function: *%PPB*. Plots (a) and (d): *Replacement pressure = 25%*. Plots (b) and (e): *Replacement pressure = 50%*. Plots (c) and (f): *Replacement pressure = 75%*. Plots (a), (b) and (c): median of the best training fitness over the performed 30 runs against generations. Plots (d), (e) and (f): median of the test fitness over the performed 30 runs against generations.

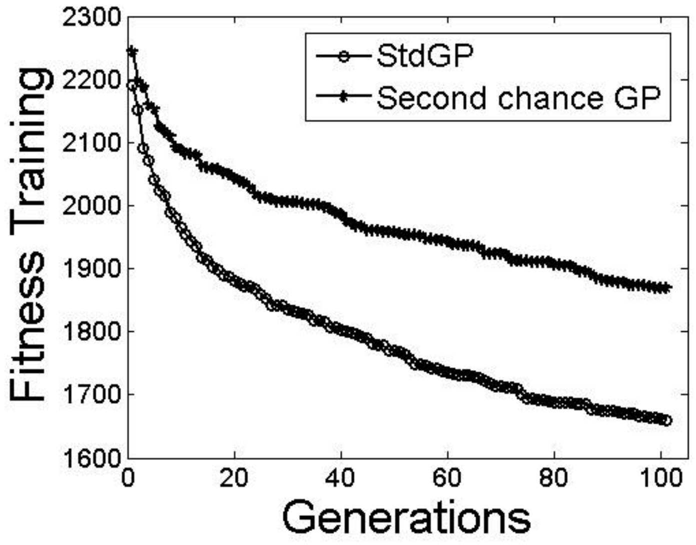


(e)

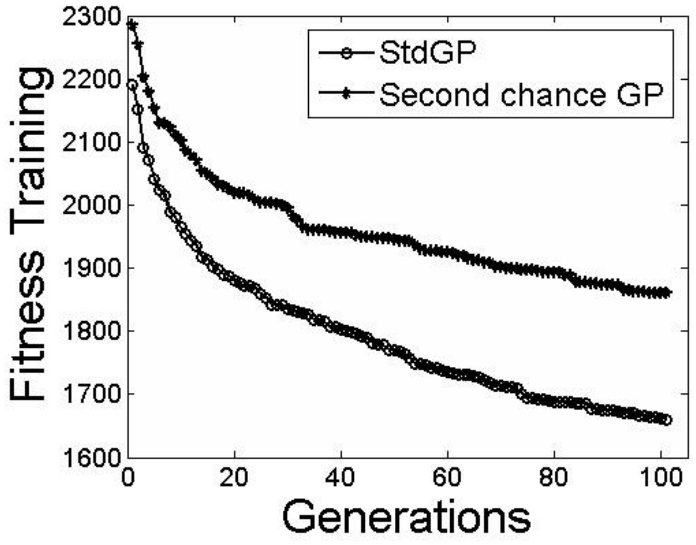


(f)

Figure 6.5: Test Function: *%PPB*. Plots (a) and (d): *Replacement pressure = 25%*. Plots (b) and (e): *Replacement pressure = 50%*. Plots (c) and (f): *Replacement pressure = 75%*. Plots (a), (b) and (c): median of the best training fitness over the performed 30 runs against generations. Plots (d), (e) and (f): median of the test fitness over the performed 30 runs against generations.

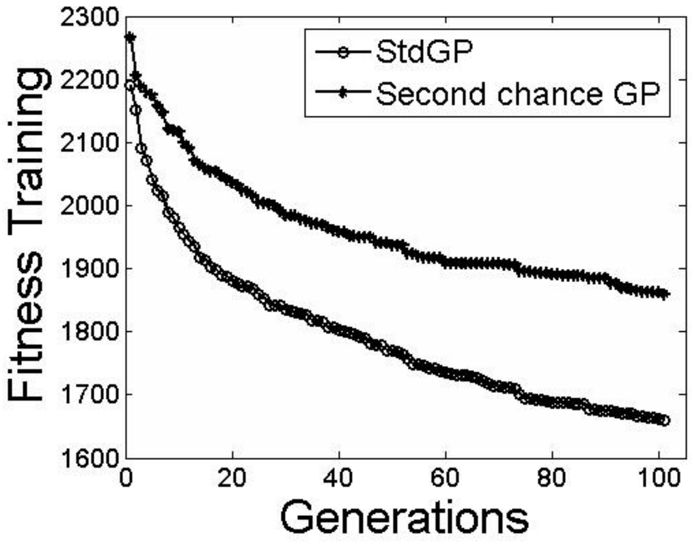


(a)

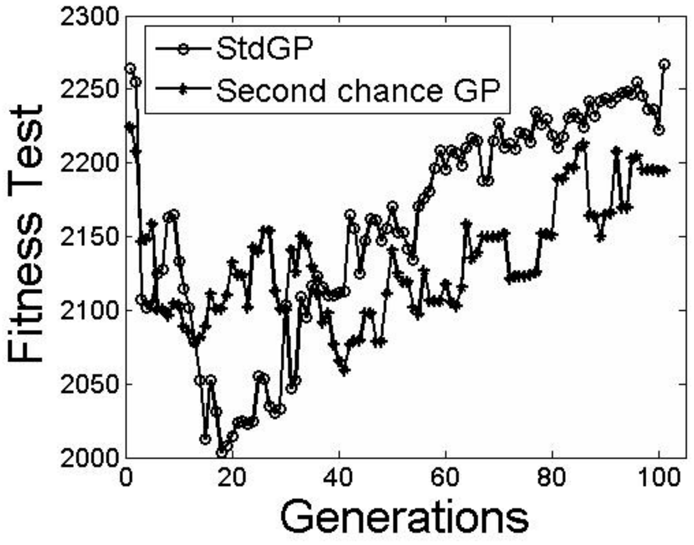


(b)

Figure 6.6: Test Function: *LD50*. Plots (a) and (d): *Replacement pressure = 25%*. Plots (b) and (e): *Replacement pressure = 50%*. Plots (c) and (f): *Replacement pressure = 75%*. Plots (a), (b) and (c): median of the best training fitness over the performed 30 runs against generations. Plots (d), (e) and (f): median of the test fitness over the performed 30 runs against generations.

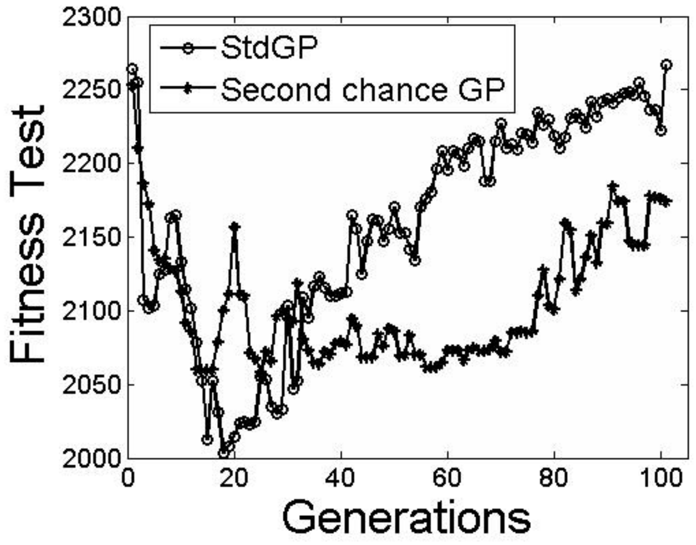


(c)

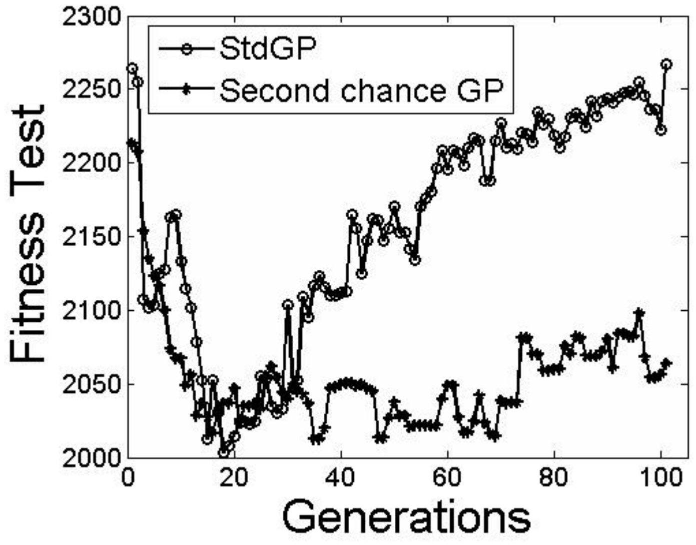


(d)

Figure 6.6: Test Function: *LD50*. Plots (a) and (d): *Replacement pressure = 25%*. Plots (b) and (e): *Replacement pressure = 50%*. Plots (c) and (f): *Replacement pressure = 75%*. Plots (a), (b) and (c): median of the best training fitness over the performed 30 runs against generations. Plots (d), (e) and (f): median of the test fitness over the performed 30 runs against generations.



(e)



(f)

Figure 6.6: Test Function: *LD50*. Plots (a) and (d): *Replacement pressure = 25%*. Plots (b) and (e): *Replacement pressure = 50%*. Plots (c) and (f): *Replacement pressure = 75%*. Plots (a), (b) and (c): median of the best training fitness over the performed 30 runs against generations. Plots (d), (e) and (f): median of the test fitness over the performed 30 runs against generations.

CHAPTER 7

Semantic

Contents

7.1 Introduction

GP has been widely investigated in the last decade. The analysis of a *GP* system is mainly based on the study of genotypic properties. While the study of genotype can be useful to capture particular phenomena related to the evolutionary process, it is not able to describe the entire dynamic of th process.

Thus incorporating semantic awareness in the GP process could improve performance, extending the applicability of GP to problems that are difficult with purely syntactic approaches. For this reason the study of a *GP* system has been extended to phenotypic aspects. This type of analysis does not consider the structural characteristics of a tree, but other properties related to the fitness. So a phenotypic study is strongly based on the fitness of the individuals involved in the evolutionary process.

In this chapter a semantic niching method to increase the performance of GP is proposed. The idea consists in building, maintaining and updating generation by generation a semantical distribution. Although original, the idea is inspired by the well-known control bloat method proposed in [92]. The similarities between the two methods are limited to acceptance criteria of new individuals produced by the genetic operators. Beyond this, the basic idea that characterizes the two methods is different: while the work in [92] builds and uses a distribution based on the syntax of the individuals, the work proposed here focuses on semantics. Five continuous and five non-continuous functions have been used to experimentally validate the approach.

7.2 State of the art

A key factor in the success or otherwise of a GP population in evolving towards a solution is the extent of diversity amongst its members. Diversity may be viewed in genotypic (structural) or in phenotypic (behavioural) terms, but the latter has received less attention. Overviews of diversity measures can be found in [78] and [18], while Burke *et al.* in [19] give a more extensive analysis of these measures and of how they relate to fitness.

Behavioural or phenotypic diversity metrics are based on the functionality of individuals, i.e. the execution of program trees rather than their appearance. Usually, behavioural diversity is viewed in terms of the spread of fitness values obtained on evaluating each member of the population [88]. One way of measuring such a diversity is by considering the fitness distribution as an indicator of entropy, or disorder, in the population [116] [87].

Other approaches consider sets or lists of fitness values and use them in combination with genotypic measures [29]. For certain types of problem it may be possible to achieve the effect of behavioural diversity without invoking the fitness function, via the use of semantic sampling schemes [65]. Semantic analysis of programs is also used in the diversity enhancing techniques described by Beadle and Johnson [11].

Gustafson in [45] developed two edit distances to sample semantic diversity in GP and conducted an analysis comparing behavioural diversity measures with changes in fitness. One of the limitations of the edit distance method is that it is defined on representations that differ substantially from the standard one.

Semantic analysis methods are starting to appear in combination with crossover.

McPhee *et al.* [71] used truth tables to analyze behavioural changes in crossover for boolean problems. They consider the semantics of two components in each tree: semantics of subtrees and semantics of context (the remainder of an individual after removing a subtree). They experimentally measured the variation of these semantic components throughout the GP evolutionary process. They paid special attention to fixed-semantic subtrees: subtrees where the semantics of the tree does not change when this subtree is replaced by another subtree. They showed that there may be many such fixed semantic subtrees when the tree size increases during evolution; thus it becomes very difficult to change the semantics of trees with crossover and mutation once the trees have become large.

While it is possible to represent behaviour using truth tables, a more efficient technique is that of using reduced ordered binary decision diagrams (ROBDDs) [16] to create reduced canonical representations to measure behavioural difference.

In [10] semantic is used to define an algorithm called Semantically Driven Crossover (SDC). The SDC algorithm has been developed based on analysis of the behavioural changes caused by crossover. The key feature of this method is the use of a canonical representation of members of the population (reduced ordered binary decision diagrams-ROBDDs) to check for semantic equivalence without having to access the fitness function. Two trees are semantically equivalent if and only if they reduce to the same ROBDD. This is used to determine which participating individuals are copied to the next generation. If the offspring are semantically equivalent to their parents, the children are discarded and the crossover is restarted. This process is repeated until semantically new children are found. The authors argue that this results in increased semantic diversity in the evolving population, and a consequent improvement in the

GP performance.

In [104] one method to incorporate semantic information into GP crossover operators for real-valued symbolic regression problems is proposed. In particular, authors aim to incorporate semantics into the design of new crossover operators so as to maintain greater semantic diversity, and provide higher locality (continuity - small changes in genotype corresponding to small changes in phenotype) than standard crossover.

Recently, Krawiec and Lichocki proposed a way to measure the semantics of an individual based on fitness cases [59]. In this work, the semantics of an individual is defined as a vector in which each element is the output of the individual at the corresponding input fitness case. This semantics is used to guide crossover in a method known as Approximating Geometric Crossover (AGC). In AGC, a number of children is generated at each crossover, the children most similar to their parents (in terms of semantics) being added to the next generation.

In [70] a new mechanism for studying the impact of subtree crossover in terms of semantic building blocks is proposed. This approach allows to completely and compactly describe the semantic action of crossover, and provide insight into what does (or does not) make crossover effective. Results make it clear that a very high proportion of crossover events (typically over 75% in the presented experiments) are guaranteed to perform no immediately useful search in the semantic space.

In [103] authors investigate the role of syntactic locality and semantic locality of crossover in GP. The results show that improving syntactic locality reduces code bloat, and that leads to a slight improvement of the ability to generalise. By comparison, improving semantic locality significantly enhances GP performance, reduces code bloat and substantially improves the ability of GP to generalise. Results confirm the more important role of semantic locality for crossover in GP.

In [12] semantic is used to test the effects of behavioural control at the point of the mutation operator. Using semantic analysis, authors present a technique known as semantically driven mutation (SDM) which can explicitly detect and apply behavioural changes caused by the syntactic changes in programs that result from the mutation operation. The SDM algorithm works to improve performance by not allowing mutated programs to be produced when they are behaviourally equivalent to the original program. The aim of this is to avoid returning to sections of the search space that have effectively already been traversed. As in [10] the key feature of the semantically driven operator is the ability to canonically represent candidate programs such that it is possible to compare for the equivalence of behaviours.

In [11] authors presented a semantic analysis of programs initialisation in GP. In particular authors defined different initialization methods that combine both semantic and syntactic. The analysis of program initialisation in GP has shown that the initialisation method chosen can have a dramatic impact on the performance of GP runs. However, it appears that this impact is problem specific, and it is not possible to conclude that one of the algorithms proposed is best for every problem. In the work there is a clear evidence that both the distribution of programs in the search space and the shape of the tree can have dramatic effects on the performance of GP. Both of these variables are strongly dependent on the problem being tackled, which therefore makes the challenge of constructing an initialisation algorithm a highly complex one.

In [49] a method to improve Phenotypic Diversity in initial GP populations is proposed. This approach to semantic diversity differs from others in that it does not involve structural considerations, fitness values or semantic analysis of programs. Instead, it focuses on the observed behaviour of individuals when they are executed. Hence phenotypic diversity is measured in terms of the run-time behaviour of programs. Author describes how this is applicable to a range of problem domains and shows how the promotion of such diversity in initial GP populations can have a substantial impact on solution-finding performance.

In [50] author extends the work proposed in [49]. In fact, although work in [49] has shown that improving behavioural diversity in initial GP populations can have a marked beneficial effect on performance, further analysis reveals that lack of behavioural diversity is a problem throughout whole runs. To address this, author proposed a method to enhance phenotypic diversity via modifications to the crossover operator, and showed that this can lead to additional performance improvements.

In [47] authors adopt statistical measurements from RNA systems to quantify robustness and evolvability at both genotypic and phenotypic levels. Using an ensemble of random walks, they demonstrate that the benefit of neutrality crucially depends upon its phenotypic distribution.

7.3 The Semantic Niching Method

The main idea of the proposed "Semantic niching" method is to use a semantic distribution to guide the evolutionary process. This distribution is used to direct the algorithm toward solutions that are semantically close to the best solution found so far by GP. Before explaining how the method works it is necessary to introduce some preliminary concepts.

The method is based on the concept of *prototype*. A prototype is defined as a vector of length equal to the number of fitness cases. Fixed an individual *i*, the vector contains the output of *i* calculated on every fitness case. So a specific prototype contains the outputs related to a single individual. This individual is the representative of the prototype.

As explained below, individuals with different fitness could belong to the same prototype.

The prototype is also associated with a value that represents the average training fitness of the elements belonging to that prototype. A prototype is then characterized by a second value that represents its capacity, that is the number of individuals that can be stored in that prototype.

When the evolutionary process starts, a new prototype for every individual in the initial population is created. Because a prototype is created for every individual, it is required that no identical individuals can be generated during the initialization process. If an individual *b* is equal to a previously generated individual *a* then *b* is discarded. At the end of the initialization process every prototype has a capacity equal to 1 and contains one element. Before starting a new generation the capacity of each prototype is updated. The capacity of a prototype is proportional to the average training fitness of its individuals. Hence, considering a maximization problem, the higher the average training fitness of a prototype, the higher is its capacity. The idea is to give "good" prototypes the opportunity to attract more individuals than "bad" prototypes. The objective is to force the search process to move towards area of the search space where good individuals lie. Once the capacities are updated, a new generation starts. Start a new generation means to apply genetic operators to the selected individuals to create new individuals. An acceptance criterion controls every newly created individual according to the algorithm 2. Before checking whether or not to accept the new individual it is necessary to determine which is the prototype to which the individual belongs. To do that it is necessary to perform the following operations:

- build the prototype p_{new} of the newly created individual;
- given the set $P = \{p_1, p_2, \ldots, p_k\}$ of already defined prototypes, $\forall p_i \in P$ calculate the average Manhattan distance $d(p_{new}, p_i)$;
- considering the set of distances $D = \{d(p_{new}, p_1), d(p_{new}, p_2), \ldots, d(p_{new}, p_k)\}$ let $\widehat{d}$ the minimal distance in D and $\widehat{p}$ the prototype for which $d(p_{new}, \widehat{p}) = \widehat{d}$;

Considering the prototypes p_{new} and p_i the average distance $d(p_{new}, p_i)$ is calculated as follows:

$$d(p_{new}, p_i) = \frac{1}{TC} \sum_{j=1}^{TC} \frac{\|p_{new}[j] - p_i[j]\|}{\|p_{new}[j] + p_i[j]\|}$$

where TC is the number of fitness cases.

The process of determining which is the prototype to which the new individual belongs can produce two different outcomes:

1. the new individual belongs to prototype $\widehat{p}$ if $\widehat{d}$ is within an acceptance threshold set *a priori*.
2. if $\widehat{d}$ is greater than the acceptance threshold the new individual does not belong to any of the existing prototypes.

These different possibilities are handled by the acceptance criterion procedure. The pseudo-code of the acceptance procedure and the pseudo-code of the generation of a new population is given in Algorithm 2.

The acceptance criterion decides whether or not to accept a newly created individual. The procedure includes several cases; first of all the procedure controls whether or not the new individual belongs to an existing prototype. If the individual does not belong to an existing prototype, a new prototype that stores only the new individual is created if and only if the new individual is the best individual in the population. This idea is inspired by Operator Equalization, a bloat control method proposed in [92]. This is the only case for which it is needed to create a new prototype. If the newly created individual belongs to an already defined prototype there are three different cases handled by the acceptance procedure:

- if the new individual belongs to an existing prototype that is not full, the new individual is accepted and stored in the prototype;
- if the new individual belongs to an existing prototype that is full, the new individual is accepted and stored in the prototype if and only if its fitness is the best with respect to the other individuals belonging to the prototype;
- in all the other cases the new individual is rejected.

It is important to point out that at the end of a generation, before updating the prototypes' capacities, it is necessary to reallocate the individual inside the prototypes if a new prototype has been created. In fact, an individual i belonging to the prototype p_a before the creation of the prototype p_{new}, may change its prototype if the prototype built considering the outputs of i is much closer to p_{new} than p_a. This procedure is not required by Operator equalization, but the computational time required by this procedure is negligible.

The semantic method is characterized by the presence of an acceptance threshold AT: this threshold influences the number of prototypes created during the evolution and also the distribution of the individuals inside the prototypes. A high value of AT permits to concentrate in a small number of prototypes a large number of individuals. On the other hand a small AT value permits a more fine grained distribution of the individuals in a large number of prototypes.

7.4 Test Problems

In this section we report the problems used in the experimental phase. 10 regression problems have been chosen, 5 are continuous functions while the other 5 are non-continuous functions. The non-continuous functions used are reported in figure 7.1 while figure 7.2 reports the continuous test functions. Ranges are denoted using (start : step : stop). Random ranges are denoted using rand(min, max), which defines uniform random sampling in the range. We point out that the considered test functions consists of one, two or three variables. In the second phase of the experimental phase, we also considered real life problems characterized by a high number (greater than 100) of features. Results obtained with these functions are similar to the ones obtained by the test functions showed in figure 7.1 and figure 7.2, hence we do not report them in the following sections. Differently from the previous chapters, we do not consider real life problems during the experimental phase. The main motivation relies in the characteristics of the considered test problems: we needed continuous and non-continuous functions where the target function is known. In fact, in a preliminary analysis, we were interested not only in the fitness of the individuals, but also in the shape of the best individuals with respect to the target function.

7.5 Experimental Settings

In this section experimental settings are reported. The proposed method is compared with standard GP and with the method proposed in [40]. The method in [40] works by separating the training set into two sets, the fitness evaluation set (67% of the training data) and the validation set (33% of the training data), and making use of the lexicographic parsimony pressure [67]. The fitness evaluation set is used to compute the error rate that guides the evolution while the validation set is used only to select the best-of-run individual. The fitness measure consists in minimizing the error rate on the fitness evaluation set. At each generation, a two-objective sort is conducted in order to extract a set of non-dominated individuals (the Pareto front), with regards to the lowest fitness evaluation set error rate and the smallest individuals. These non-dominated individuals are then evaluated on the validation set, with the best-of-run individual selected as the one of these with the smallest error rate on the validation set, ties being solved by choosing the smallest individual. From now on we refer to this method with *parsimony pressure*.

For every test problem, 30 runs of the considered techniques have been performed. All the runs used populations of 100 individuals and the evolution stopped after 40000 fitness evaluations. Tree initialization was performed with the Ramped Half-and-Half method [82] with a maximum initial depth of 6. The function set contained the four binary operators $+$, $-$, $*$, and $/$ protected as in [82]. The terminal set contained all 1, 2 or 3 variables depending on the test functions and no random constants. Tournament

Function	Notation	Training Data	Test Data
f_{NC1}	$f(x) = \begin{cases} e^{x+5} & \text{if } x < 0 \\ 1 - log(x^2 + x + 1) & \text{if } x \geq 0 \end{cases}$	100 points, x=rand(-2.0,2.0)	x = (-2.0 : 0.05 : 2.0)
f_{NC2}	$f(x) = \begin{cases} x + 6 & \text{if } x < 0 \\ \frac{3}{x^2+1} & \text{if } x \geq 0 \end{cases}$	100 points, x=rand(-2.0,2.0)	x = (-2.0 : 0.05 : 2.0)
f_{NC3}	$f(x) = \begin{cases} x + \sin(x-1) & \text{if } x < 0 \\ 6\sin(x)\cos(x) & \text{if } 0 \leq x < 1 \\ \sqrt{x} & \text{if } x \geq 1 \end{cases}$	100 points, x=rand(-2.0,2.0)	x = (-2.0 : 0.05 : 2.0)
f_{NC4}	$f(x) = \begin{cases} \frac{30x}{(x-2)x} & \text{if } x \leq -1 \\ \frac{x^4-x^3+x^2}{(x+2)} & \text{if } -1 < x < 0 \\ \frac{x}{5} & \text{if } 0 \leq x < 1 \\ x^3 e^x \cos x & \text{if } x \geq 1 \end{cases}$	100 points, x=rand(-2.0,2.0)	x = (-2.0 : 0.05 : 2.0)
f_{NC5}	$f(x) = \begin{cases} \frac{8}{(2+x^2+2x^4)x} & \text{if } x < -2 \\ \frac{x^3}{5} & \text{if } -2 \leq x \leq 0 \\ \frac{x}{x^2} & \text{if } x > 0 \end{cases}$	100 points, x=rand(-4.0,2.0)	x = (-4.0 : 0.05 : 2.0)

Figure 7.1: Non-continuous test functions

Function	Notation	Training Data	Test Data
f_{C1}	$f(x_1,x_2) = \frac{e^{-(x_1-2)^2}}{1.2+(x_2-2.5)^2}$	100 points, x_1,x_2=rand(0.3,4.0)	x_1,x_2=rand(0.3 : 0.1 : 4.0)
f_{C2}	$f(x_1,x_2,x_3) = \frac{30x_1x_3}{(x_1-10)x_2^2}$	1000 points, x_1,x_3=rand(-1.0,1.0), x_2=rand(1.0, 2.0)	x_1,x_3 = (-1.0 : 0.1 : 1.0), x_2 = (1.0 : 0.1 : 2.0)
f_{C3}	$f(x_1,x_2) = c\sin(x)\cos(x)$	30 points, x_1,x_2=rand(0.1,5.9)	x_1, x_2 = (0.1 : 0.1 : 5.9)
f_{C4}	$f(x_1,x_2,x_3) = 30\frac{(x_1-1)(x_3-1)}{x_2^2(x_1-10)}$	300 points, x_1,x_3=rand(0.05,2.0) x_2=rand(1.0,2.0)	x_1, x_3 = (0.05 : 0.05 : 2.0), x_2 = (1.0 : 0.05 : 2.0)
f_{C5}	$f(x_1,x_2) = \frac{(x_1-3)^4+(x_2-3)^3-(x_2-3)}{(x_2-2)^4+10}$	50 points, x_1,x_2=rand(0.05,6.05)	x_1,x_2 = (0.05 : 0.01 : 6.05)

Figure 7.2: Continuous test functions

selection has been used with tournament size equal to 4. The reproduction (replication) rate was 0.1, meaning that each selected parent has a 10% chance of being copied to the next generation instead of being engaged in breeding. Standard tree mutation and standard crossover (with uniform selection of crossover and mutation points) were used with probabilities of 0.1 and 0.9, respectively. The new random branch created for mutation has maximum depth 6. Selection for survival was elitist and only the best individual is inserted in the new population. No maximum tree depth was imposed.

The fitness of the individuals is the Canberra distance [62] normalized between 0 and 1, where 0 is the optimal fitness.

The median was preferred over the mean in all the evolution plots shown in the next section because median is more robust to outliers.

7.6 Experimental Results

In this section the obtained experimental results are reported. In particular section 7.6.1 outlines results obtained with the continuous test functions while section 7.6.2 presents results obtained with the non-continuous test problems. The experimental phase has been performed to check whether or not the proposed semantic niching method is able to produce better results with respect to the other considered techniques. In particular we are interested in checking the generalization ability of the proposed techniques and its ability to increase population diversity. Another aspect we are interested in is the size of the individuals produced by our method.

To check the generalization ability, the fitness on unseen data reported in the column labelled by "Test Data" in Figures 7.1 and 7.2 is considered; to measure the degree of diversity of the individuals in the population we use entropy, according to [19]. The size of a specific individual is defined as the number of its nodes.

7.6.1 Continuous Functions

In this section, results related to the continuous test functions are presented. The entropy of the population and the test fitness of the best element are reported, for all the considered test functions, from Figure 7.3 to Figure 7.7. We start to analyze the entropy of the populations. The behaviour of the entropy of the population is similar for all the continuous test functions. Standard GP produces the lowest entropy value, that means that it has a low degree of population diversity. On the other hand the parsimony pressure technique shows the highest entropy value meaning that the population maintain a high degree of diversity. The interesting point is that the entropy values for standard GP and parsimony pressure are quite stable during the evolutionary process.

The behaviour showed by our method is different. In the first evaluations we have a high entropy value, so a high degree of population diversity. During the evolutionary process the entropy value decreases and reaches the minimum value in the last evaluations. This behaviour is desirable and expected if we consider the role of the semantic bins. In fact, in the first evaluations, all the prototypes contain more or less the same number of individuals. Hence, in the earliest evaluations, a large number of different semantics are taken in account. Evaluation by evaluation, individuals tend to concentrate in a few prototypes that represent the semantics of individuals with good fitness.

So entropy seems to confirm the capability of our method to converge towards good individuals.

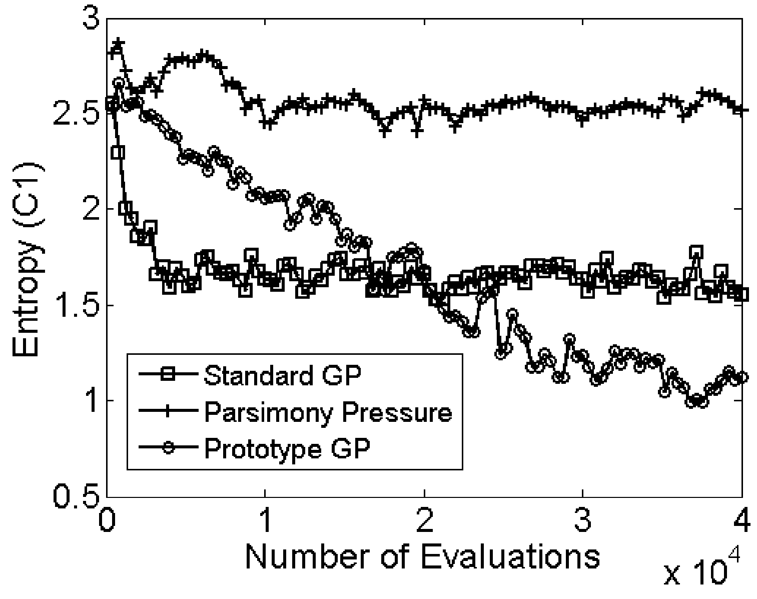
Now we analyze the behaviour of test fitness. In function C_1 standard GP is the best performer but the behaviour of all the considered techniques is quite similar. In function C_2 parsimony pressure and *Prototype GP* are the best techniques, while standard GP performs poorly. In functions C_3 and C_5 our method produces the best results while standard GP produces the worst results. In function C_4 our method and the parsimony pressure technique produce similar results, while standard GP is the worst technique.

A statistical validation of the results has been performed. In particular, the statistical significance of the results has been evaluated with the Mann-Whitney test [24], considering a confidence of 95% and a pairwise Bonferroni correction for the value of α. According to the Mann-Whitney test the *Prototype GP* method produces fitness values that are statistically different from the other considered techniques, except for the test function C_1 and for the test function C_2. In particular for C_1 standard GP and our method are not statistically different, while the parsimony pressure and our method are not statistically different when C_2 is considered.

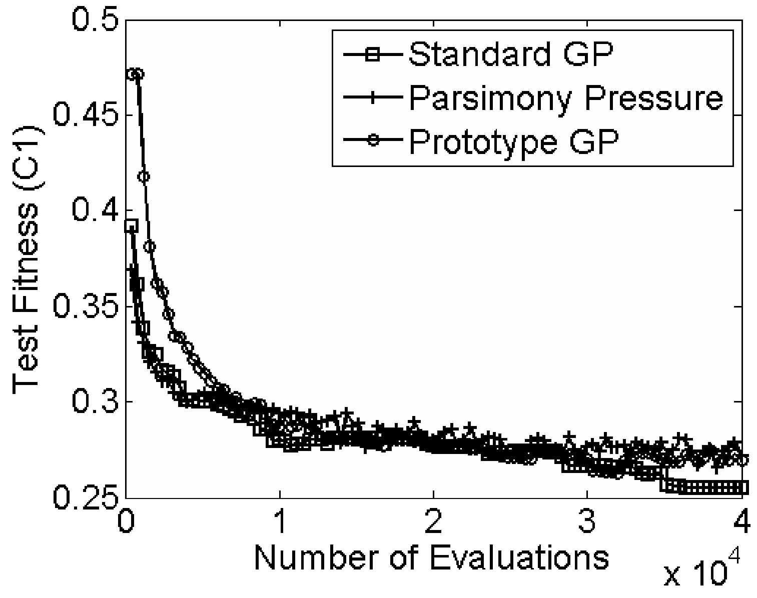
Regarding the entropy, according to the Mann-Whitney test the *Prototype GP* method produce values that are statistically different from the other considered techniques for all the considered continuous test functions.

7.6.2 Non-Continuous Functions

In this section results related to the non-continuous test functions are discussed. The entropy of the population and the test fitness of the best element are reported, for all the considered test functions, from Figure 7.8 to Figure 7.12. Regarding the entropy values, the same considerations made for the continuous functions remain valid. In particular the entropy values produced by standard GP and parsimony pressure are

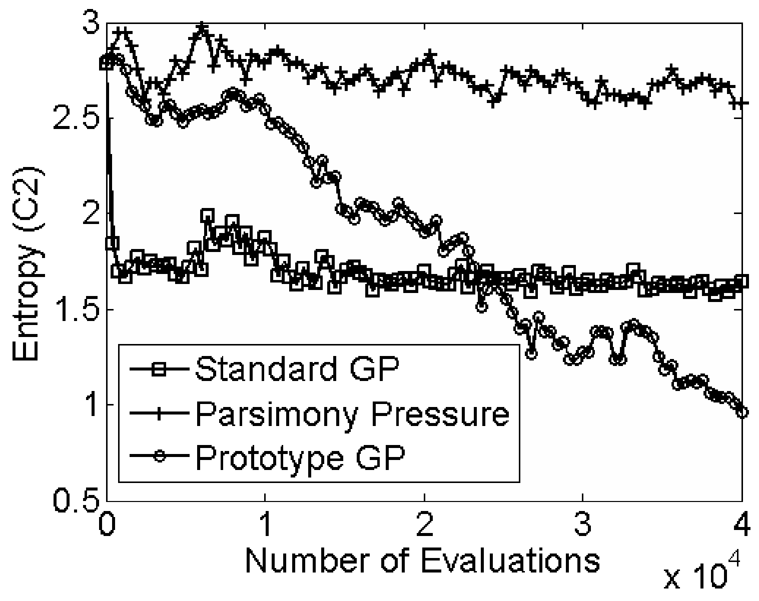


(a)

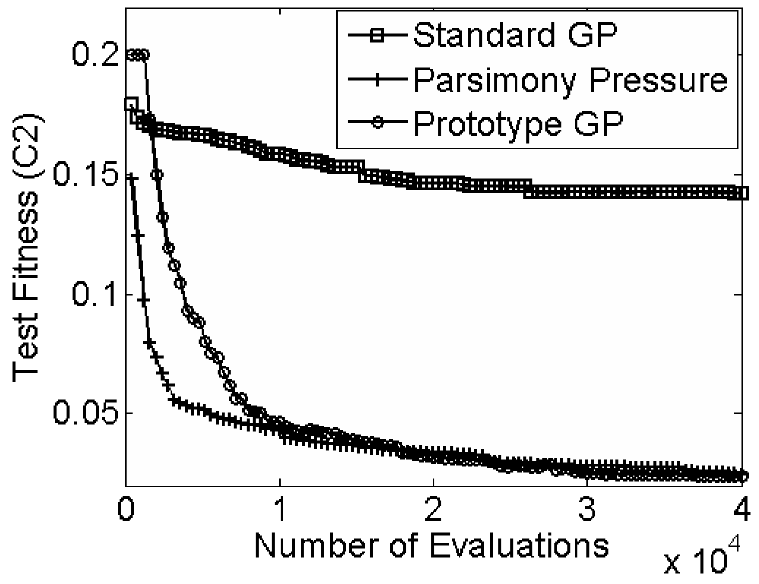


(b)

Figure 7.3: Figure (a) reports entropy of the populations with respect to the number of evaluations for test function C_1. The median over 30 runs has been reported. Figure (b) reports test fitness of the best individuals with respect to the number of evaluations for test function C_1. The median over 30 runs has been reported.

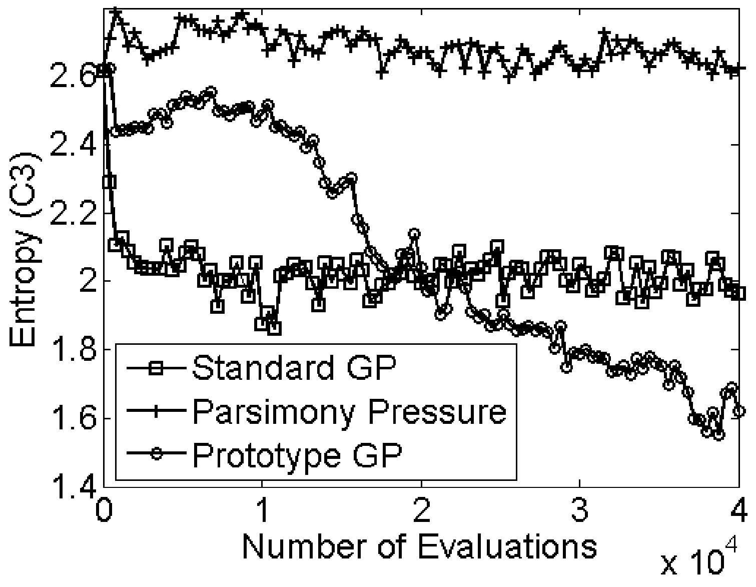


(a)

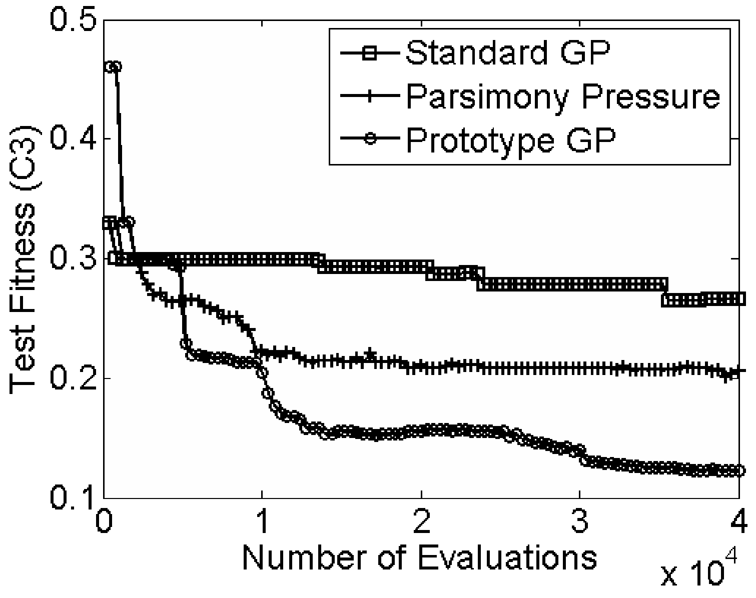


(b)

Figure 7.4: Figure (a) reports entropy of the populations with respect to the number of evaluations for test function C_2. The median over 30 runs has been reported. Figure (b) reports test fitness of the best individuals with respect to the number of evaluations for test function C_2. The median over 30 runs has been reported.

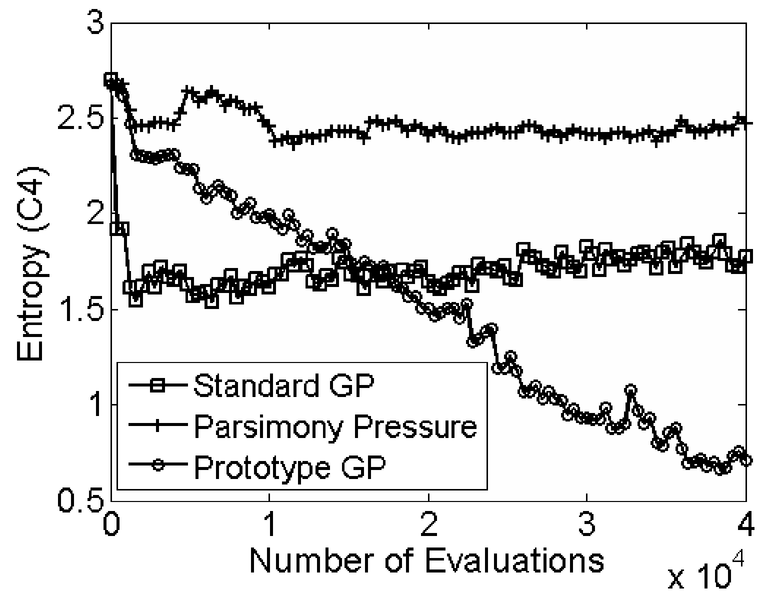


(a)

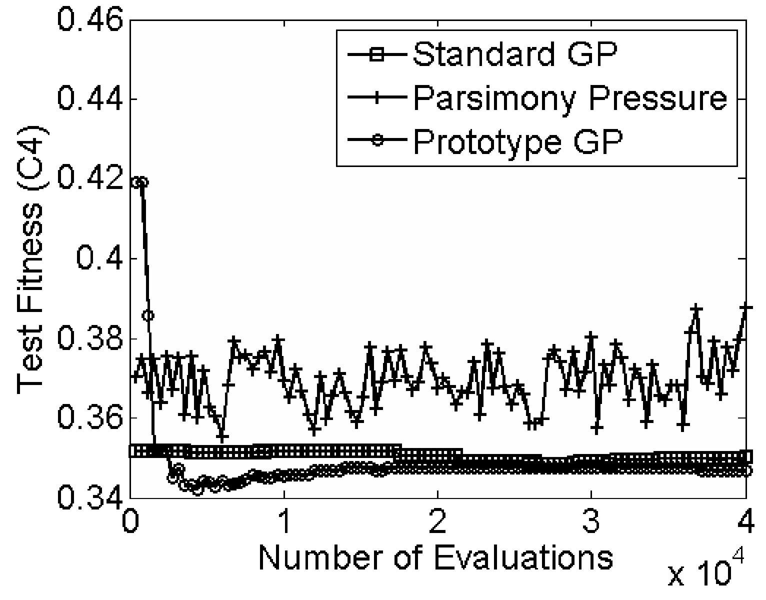


(b)

Figure 7.5: Figure (a) reports entropy of the populations with respect to the number of evaluations for test function C_3. The median over 30 runs has been reported. Figure (b) reports test fitness of the best individuals with respect to the number of evaluations for test function C_3. The median over 30 runs has been reported.

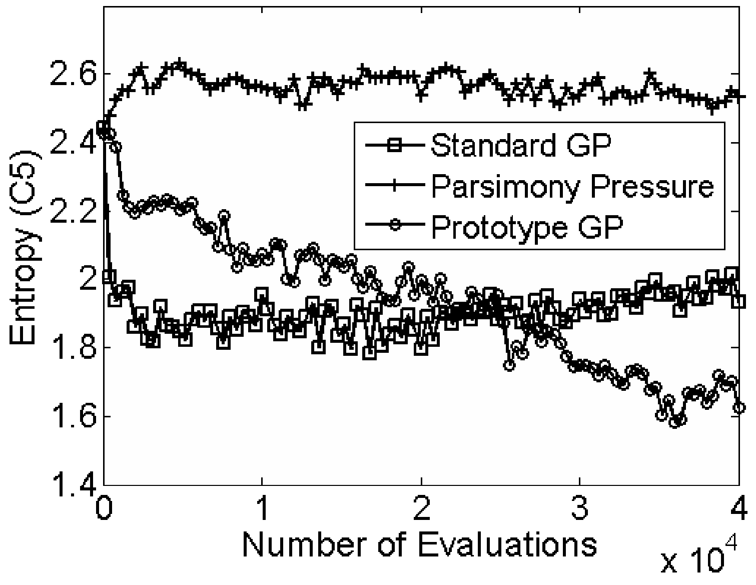


(a)

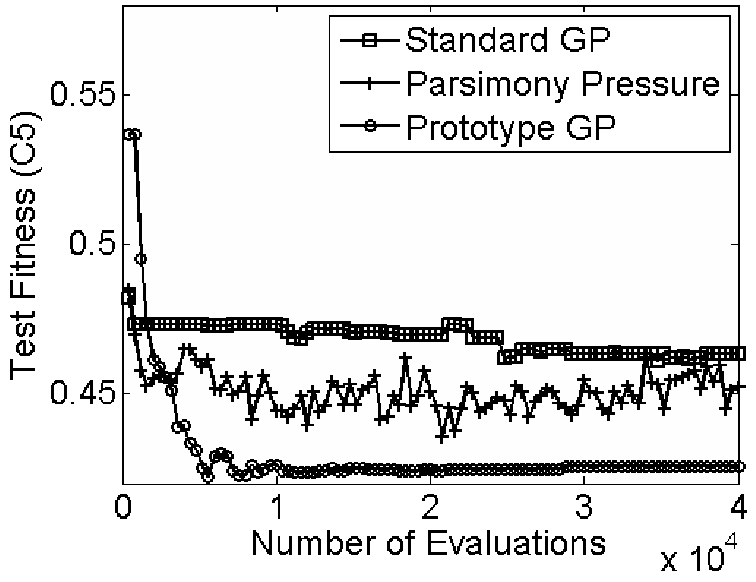


(b)

Figure 7.6: Figure (a) reports entropy of the populations with respect to the number of evaluations for test function C_4. The median over 30 runs has been reported. Figure (b) reports test fitness of the best individuals with respect to the number of evaluations for test function C_4. The median over 30 runs has been reported.



(a)



(b)

Figure 7.7: Figure (a) reports entropy of the populations with respect to the number of evaluations for test function C_5. The median over 30 runs has been reported. Figure (b) reports test fitness of the best individuals with respect to the number of evaluations for test function C_5. The median over 30 runs has been reported.

quite stable, regardless the state of the evolutionary process. Considering the *Prototype GP* method, it is possible to see that the entropy values decrease while the number of evaluations increases.

Regarding the test fitness, the best performances are obtained by the *Prototype GP* method on all the considered test functions.

As well as for continuous functions, a statistical validation of the results has been performed. According to the Mann-Whitney test, the results produced by the *Prototype GP* method are statistically different from the ones produced by the other considered techniques for all the considered non-continuous test functions.

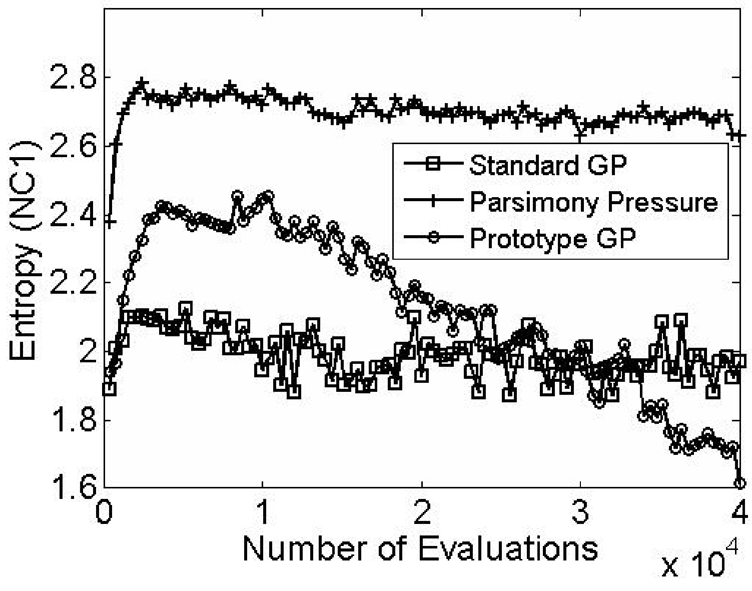
7.7 Distribution of the fitness values

The objective of this section is to analyze how individuals that take part in the evolutionary process are distributed within the search space for the different studied techniques. In particular, the main aim is to determine whether or not there are significant differences, between the considered techniques, regarding the exploration of the search space.

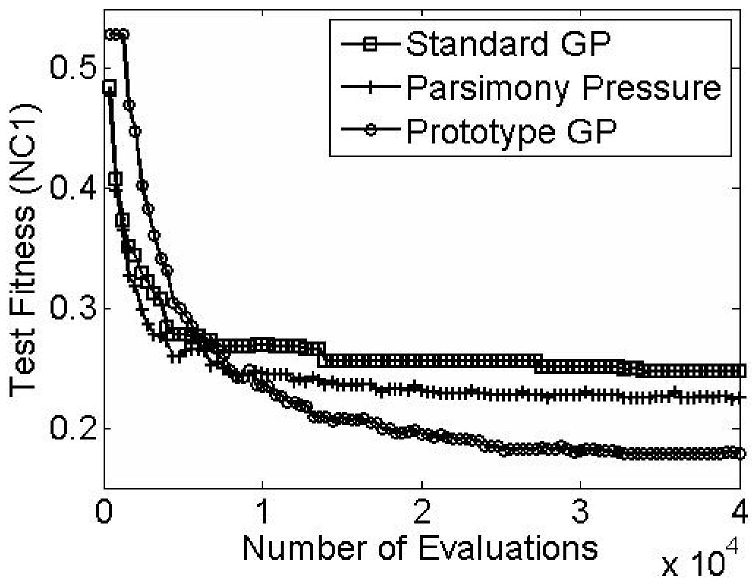
To do that, all the 30 runs have been considered and the training fitness of all the individuals produced by the evolutionary process has been analyzed. This analysis involved all the test functions, but we reported only the results for one continuous function and for one non-continuous function. For the other considered test functions results are similar.

Results are presented in Figure 7.13 and in Figure 7.14.

On the horizontal axis, numbers denote fitness intervals. Hence 1 covers individ-

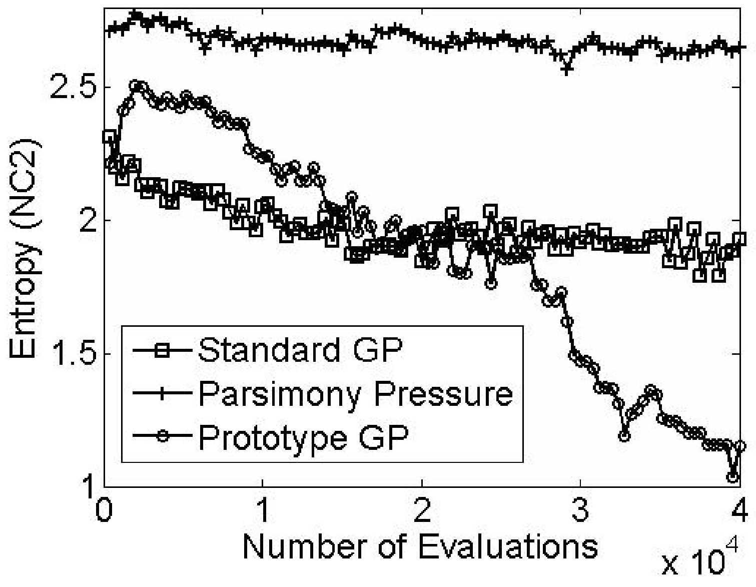


(a)

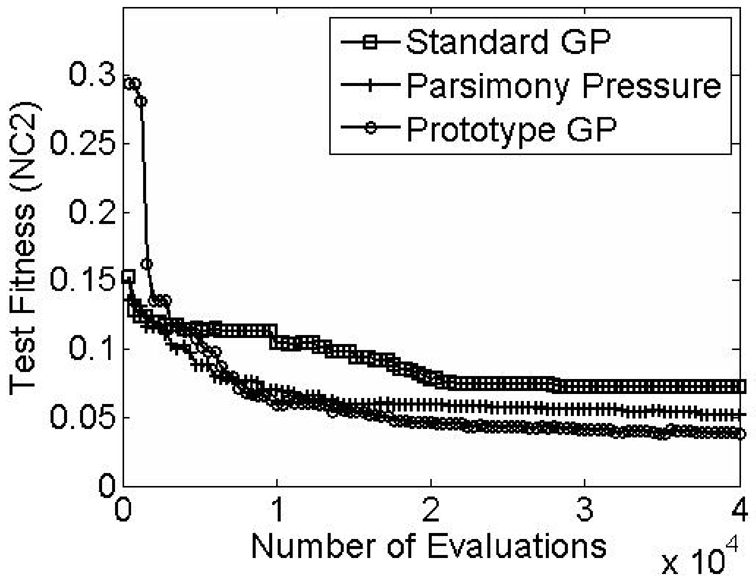


(b)

Figure 7.8: Figure (a) reports entropy of the populations with respect to the number of evaluations for test function NC_1. The median over 30 runs has been reported. Figure (b) reports test fitness of the best individuals with respect to the number of evaluations for test function NC_1. The median over 30 runs has been reported.

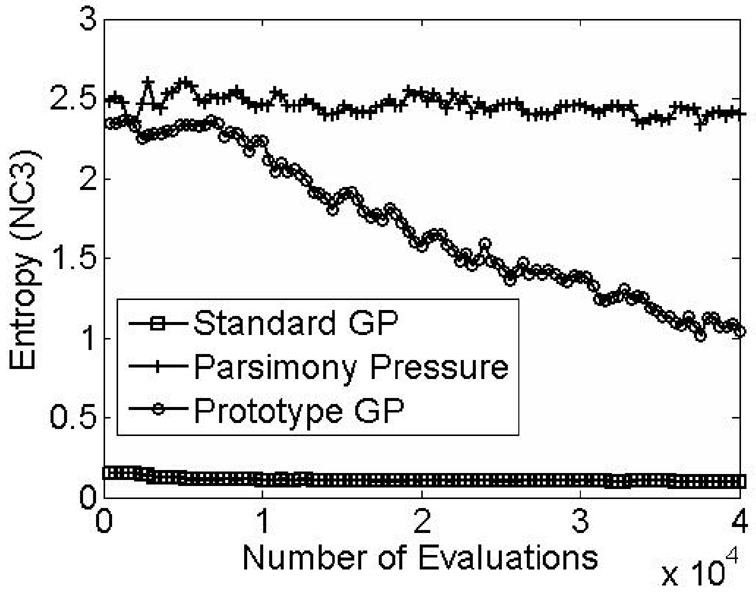


(a)

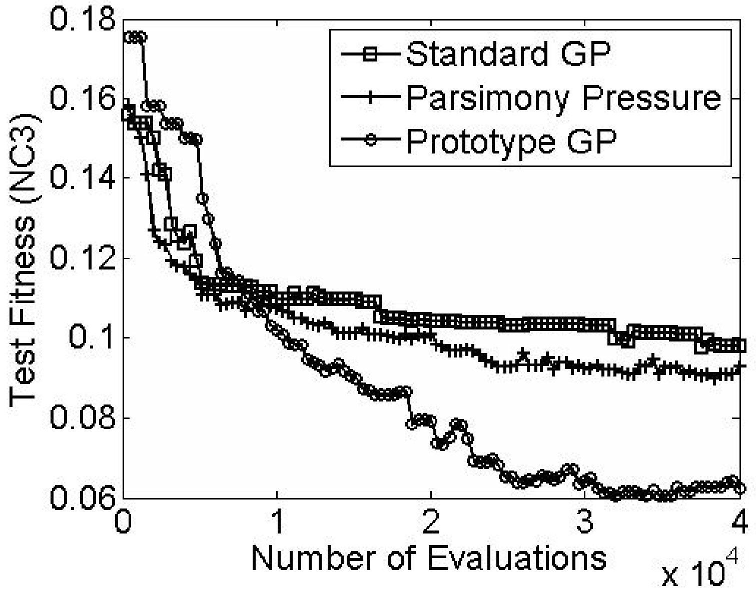


(b)

Figure 7.9: Figure (a) reports entropy of the populations with respect to the number of evaluations for test function NC_2. The median over 30 runs has been reported. Figure (b) reports test fitness of the best individuals with respect to the number of evaluations for test function NC_2. The median over 30 runs has been reported.

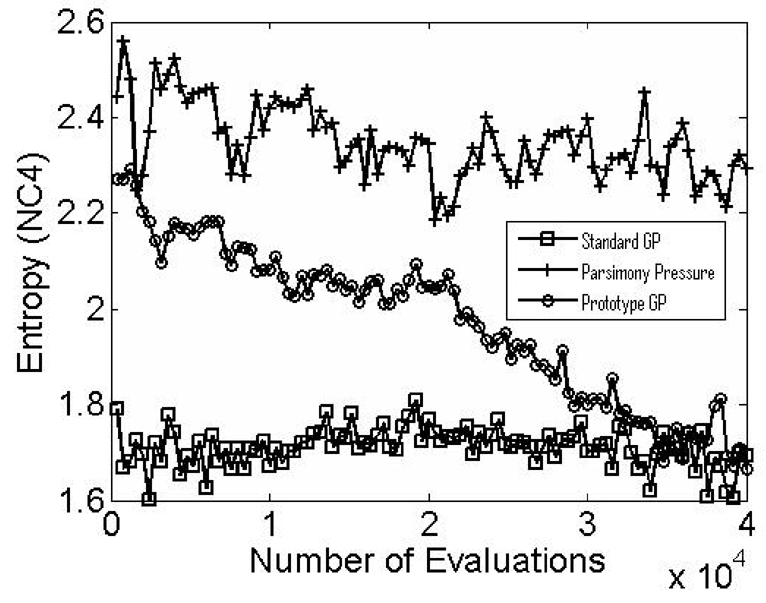


(a)

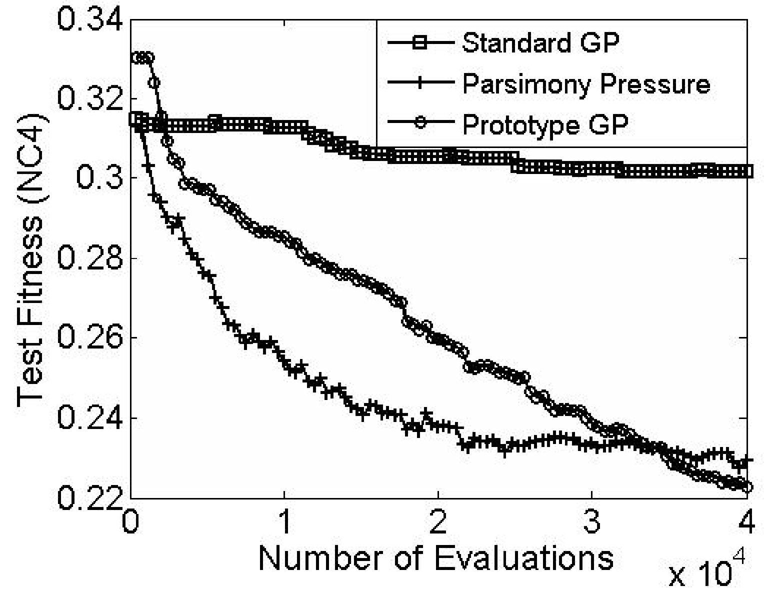


(b)

Figure 7.10: Figure (a) reports entropy of the populations with respect to the number of evaluations for test function NC_3. The median over 30 runs has been reported. Figure (b) reports test fitness of the best individuals with respect to the number of evaluations for test function NC_3. The median over 30 runs has been reported.

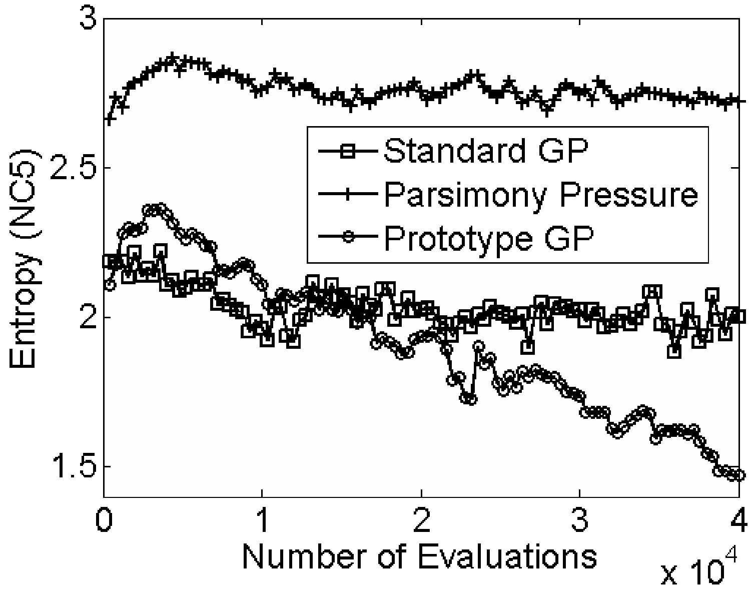


(a)

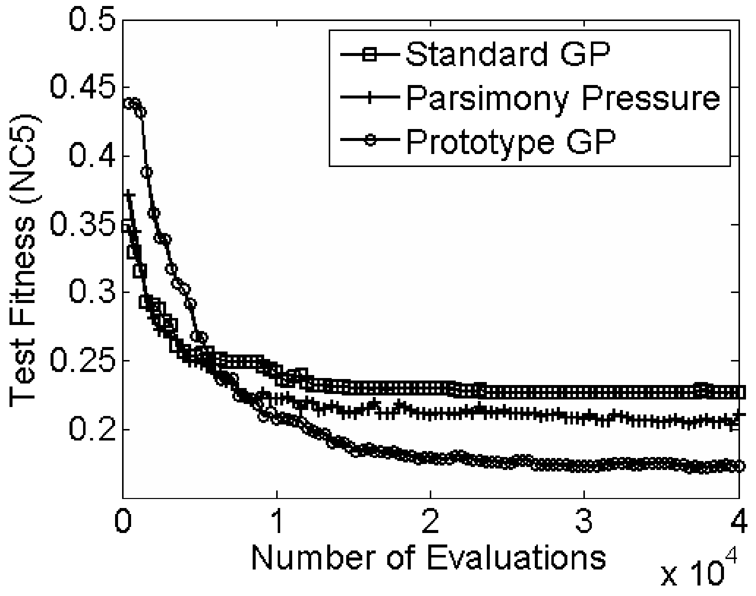


(b)

Figure 7.11: Figure (a) reports entropy of the populations with respect to the number of evaluations for test function NC_4. The median over 30 runs has been reported. Figure (b) reports test fitness of the best individuals with respect to the number of evaluations for test function NC_4. The median over 30 runs has been reported.

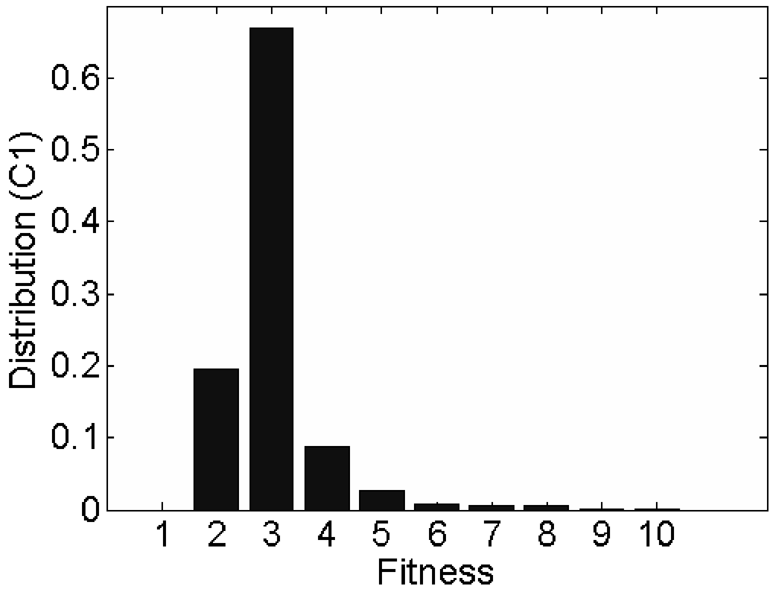


(a)

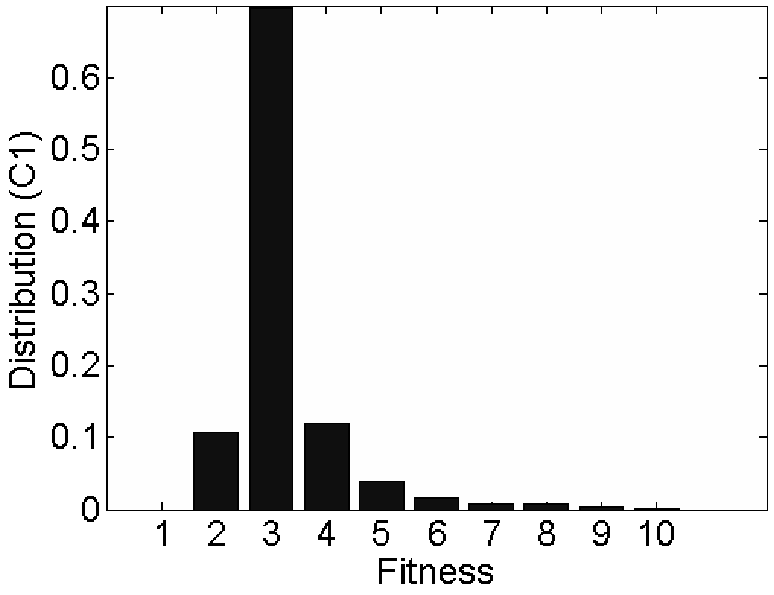


(b)

Figure 7.12: Figure (a) reports entropy of the populations with respect to the number of evaluations for test function NC_5. The median over 30 runs has been reported. Figure (b) reports test fitness of the best individuals with respect to the number of evaluations for test function NC_5. The median over 30 runs has been reported.

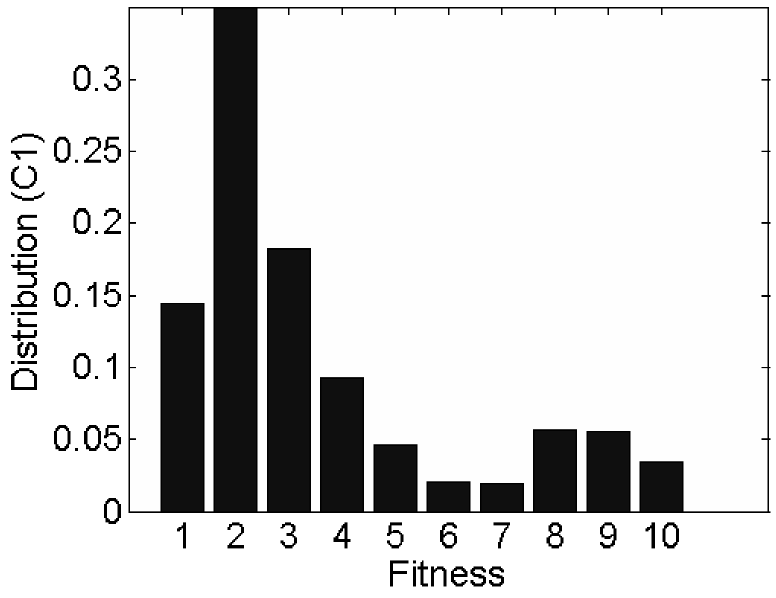


(a)



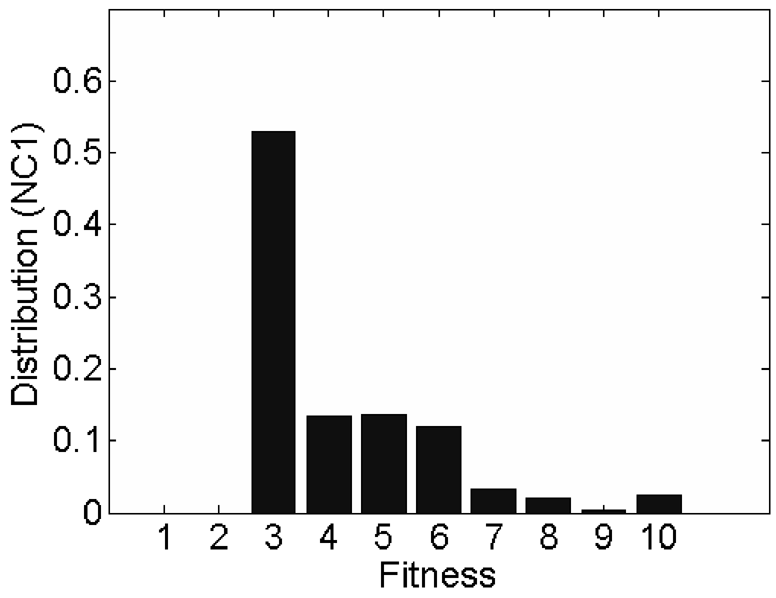
(b)

Figure 7.13: Figures (a),(b) report respectively the probability distribution for *Standard GP* and *Parsimony Pressure*. Test problem: C_1 function.

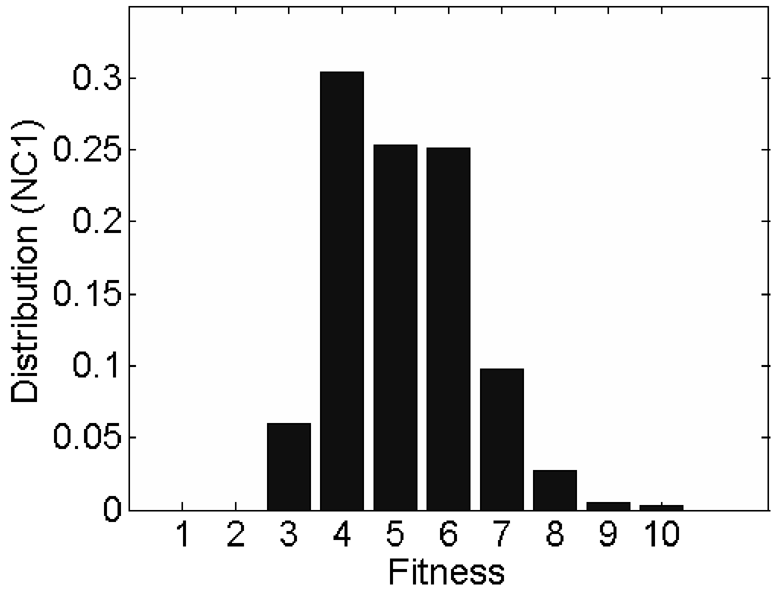


(c)

Figure 7.13: Figure (c) reports the probability distribution for *Prototype GP*. Test problem: C_1 function.

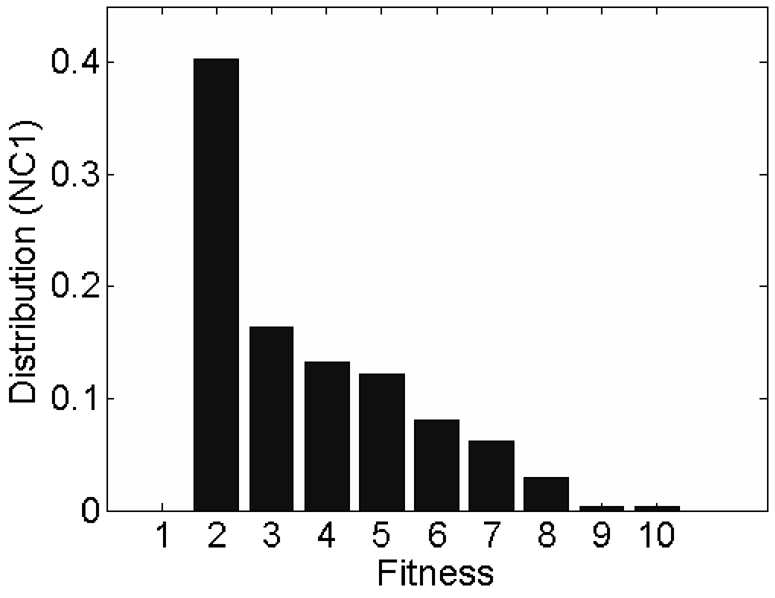


(a)



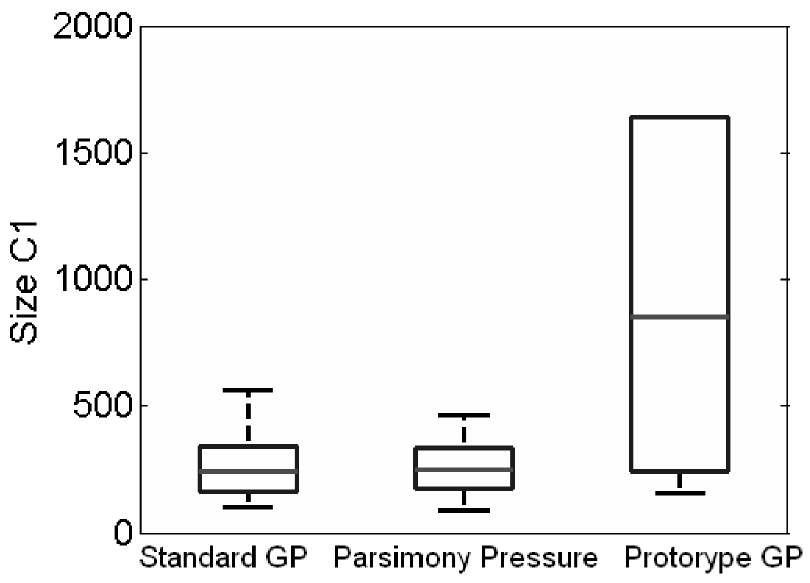
(b)

Figure 7.14: Figures (a),(b) report respectively the probability distribution for *Standard GP* and *Parsimony Pressure*. Test problem: NC_1 function.

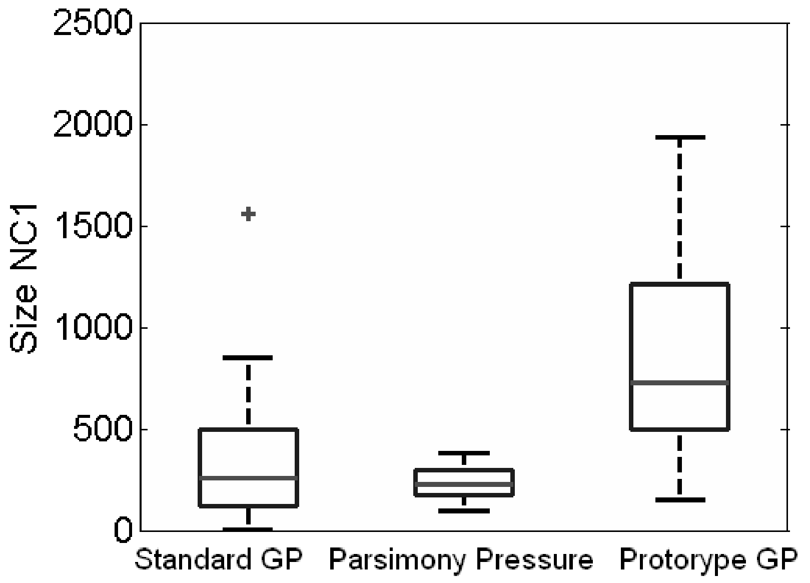


(c)

Figure 7.14: Figure (c) reports the probability distribution for *Prototype GP*. Test problem: NC_1 function.



(a)



(b)

Figure 7.15: Figure (a) and (b) report size of best individuals over 30 runs for C_1 and NC_1 respectively.

uals with fitness in the range [0;0.1], 2 covers individuals with fitness in the range [0.1;0.2] and so on.

As it is possible to see, in both the considered problems the *Prototype GP* method shows a larger number of good individuals compared to the other studied techniques. In particular, for the C_1 function, while standard GP and parsimony pressure do not have any individual with training fitness in the range [0:0.1], the *Prototype GP* method produces a 15% of individuals in this range. Considering also the range [0.1;0.2] it is possible to note that our method produces a percentage of individuals in this range that is remarkably greater than the one produced by the other considered techniques.

Considering Figure 7.14, with the NC_1 function we have the same behaviour. In particular the *Prototype GP* method produces a 40% of individuals with fitness in the range [0.1;0.2] while the other considered techniques produce no individuals with fitness in this range.

This analysis seems to confirm the ability of our method to explore areas of the search space where good individuals lie.

7.8 Problems

In this section we outline two drawbacks of our method. The first one is related to the number of individuals that are rejected after the evaluation. Obviously this fact leads to an increase of the computational effort required to run the evolutionary process. In order to take account of this problem and perform a fair comparison with the other considered techniques, in the experimental phase presented in the previous sections, the results are reported against the total number of evaluated individuals, instead of the number of generations, thus taking into account also the individuals that have been evaluated and then rejected by *Prototype GP*.

The second aspect that we want to discuss is related to the size of the individuals produced by our method. Figure 7.15(a) and Figure 7.15(b) report the size of the best individuals produced by the considered techniques for two test functions. For the other considered test functions results are similar.

Results show that our technique produces very large trees in respect to the ones produced by the other considered ones. While this fact does not generate bloat because the growth in size of the individuals is also accompanied by a fitness improvement [92] it would be preferable to have small size trees. One method to reduce this problem could be the simplification of the expressions represented by the individuals.

We think that possible explanations of this phenomenon lie in the way in which the proposed method works. In particular having a prototype P with capacity n, the algorithm will look for n individuals with the semantic described by the prototype P. The problem is that a particular semantic could be represented only by few trees of a certain depth. Hence the algorithm must look for big trees in order to fill the prototype P.

7.9 Conclusions and Future Works

A semantic niching method to increase the performance of GP has been proposed. The idea consists in building, maintaining and updating generation by generation a semantical distribution. Experimental results show that the proposed method performs better or in a comparable way to standard GP and to the Parsimony Pressure method. In particular, the proposed method outperforms the other studied ones on all the considered non-continuous test functions. More than the absolute performance, it is of interest the effect of the proposed method on population's semantic diversity. In fact, the *Prototype GP* method maintains a high degree of population diversity in the first evaluations of the learning process while it reduces the population diversity in the following evaluations. This is a good indicator of the fact that the proposed method is able to converge on solutions that are semantically better. In particular, the *Prototype GP* method seems to be able to cover areas of the search space characterized by the presence of solutions with good fitness.

Nevertheless, the proposed method has also some problems. In particular, it tends to create large individuals. This does not correspond to the definition of bloat [92], since individuals generated by *Prototype GP* have a fitness that is better than the fitness of individuals created by standard GP or parsimony pressure. However, one of the future works concerns the simplification of the solutions produced by the *Prototype GP* method.

Another future work is the application of our method with a fitness sharing mechanism. Fitness sharing modifies the search landscape by reducing the payoff in densely populated regions. Hence, we can use fitness sharing to slow down the convergence of the individuals towards a few prototypes. In fact, while the convergence is desirable, the premature convergence must be avoided.

Algorithm 2 Generation of a new population.

```
begin
  n_accepted := $0$;
  while n_accepted $<$ pop_size do
        select parents from population;
        apply genetic operator;
        for (every child i) do
           k := calculate prototype(i);
           if (acceptance_criterion(i,k) $==$ $true$)
             then
                  insert i in the new population;
                  n_accepted = n_accepted + 1;
           fi
        od
  od

  proc acceptance_criterion(individual i, prototype k)
     if (k is an existing prototype)
       then
            if (k is full )
              then
                   if (i is the best of the prototype k)
                     then
                          accept := true;
                     else
                          accept := false;
                   fi
              else
                   accept := true;
            fi
       else if (i is the best of the run)
              then
                   create a new prototype k that contains only individual i
                   accept := true;
              else
                   accept := false;
            fi
     fi
  end
```

CHAPTER 8

Conclusions

Contents

In this chapter the main contributions of this thesis are summarized.

8.1 Contributions Overview

The main goal of this thesis was the definition of methods for a better understanding of the dynamics of GP. While GP has been widely used in the last two decades, a lot of open problems deserve investigation. Hence, the objective of this thesis was to to provide techniques that may help in facing these open problems. Because the techniques that have been proposed should be used in a wide range of domains, an important part of this thesis has been dedicated to the experimental validation of the proposed techniques. In particular, the claim is that all the proposed measures and methods could be used not only in toy examples, but also in real life applications. In this light, the experimental validation has been performed, when possible, by considering different real life applications characterized by a large dimension of the space of the features.

A particular attention has been dedicated to the set up of the experiments, considering different values of the parameters and repeating the experiments many times. Results produced in this thesis have been analyzed from different points of view, and a statistical validation has been performed to guarantee the significance of the results. This latter phase was necessary given the stochastic nature of the GP process.

The open problems that have been considered in this thesis are:

- The lack of measures to quantify phenomena that characterize the GP process.
- The defect of techniques that could improve the generalization ability of the solutions produced by GP.
- The lack of a method to completely exploit the semantic of the individuals.

All these problems are particularly interesting because they limit the ability of GP to provide optimal solutions.

The following sections summarize the solutions for these problems presented in this thesis.

8.2 GP measures

The analysis of the dynamics of the evolutionary process is fundamental for a better understanding of the phenomena that influence such process. Some of these phenomena are common to many machine learning techniques, while others are strictly related to the use of GP. Three important phenomena are:

- bloat;
- overfitting;
- solutions' complexity.

Bloat is one of the most important problem that affects GP. According to [9] bloat (or code bloat) is defined as code growth without a significant improvement in terms of fitness. In recent years several theories have been developed trying to explain the origin of this phenomenon and several methods have been defined to reduce this phenomenon. Some of these theories and methods have been discussed in chapter 3. Bloat is a very important issue because large trees are created and the whole process of fitness evaluation requires more time. In addition, the larger size of the trees is not followed by an improvement in fitness. While methods to counteract bloat have been defined (see chapter 3), there is not a measure to quantify the amount of bloat in a GP run. In section 4.1 a measure to quantify bloat has been proposed. According to the proposed measure, in case of minimization problems (i.e. problems were a small fitness value is better than a large one) the "amount of bloat" at generation g is defined as:

$$bloat(g) = \frac{(\overline{\delta}(g) - \overline{\delta}(0))/\overline{\delta}(0)}{(\overline{f}(0) - \overline{f}(g))/\overline{f}(0)} \tag{8.1}$$

where $\overline{\delta}(g)$ is the average program length in the population at generation g and $\overline{f}(g)$ is the average fitness (calculated using training data) in the population at generation g.

Hence, $bloat(g)$ is an expression of the relationship between the average length growth and the average fitness improvement up to generation g compared to the respective values at generation zero. Further details are reported in section 4.1.

While bloat is a problem strictly related to the use of GP, overfitting [74] affects many machine learning techniques.

Overfitting occurs when a model begins to memorize the training data rather than learning to generalize from the model. According to [74] given a hypothesis space H, a hypothesis $h \in H$ is said to overfit the training data if there exists some alternative hypothesis $h' \in H$ such that h has smaller error than h' over the training examples, but h' has a smaller error than h over the entire distribution of instances.

A lot of studies have been performed trying to define method to counteract overfitting and to enhance the generalization ability of the solutions. Some of these methods are reported in chapter 3. As well as for the bloat, there are not measures that can quantify the amount of overfitting. In section 4.2 a measure to quantify overfitting has been proposed. To summarize, the ideas that inspired the proposed measure are the following:

- if, at a given generation `g`, test fitness is better than training fitness, then there is no overfitting (`overfit(g)=0`);
- if test fitness is better than the best test point, then there is no overfitting (`overfit(g)=0`);
- otherwise overfitting is quantified by the difference of the distance between train-

ing and test fitness at generation g and the distance between training and test fitness at the generation where the best test point has been found.

Another important study is related to the complexity of the solutions produced by GP. In fact it is often stated that simpler solutions will be more robust and will generalize better than complex ones, with the latter being more likely to overfit the training data [74], [107].

An immediate question concerns how to measure program complexity. In chapter 4 different measures to quantify the complexity of solutions have been proposed. The first measure is called Multi-Slope Complexity Measure (MSCM) and it is inspired by the mathematical concept of curvature [20]. The formal definition of this measure is reported in section 4.3. The intuition is the following: consider the three simple bidimensional functions in Figure 4.2. The objective is to define a measure to quantify the fact that the function represented in Figure 4.2(c) is more complex than the one represented in Figure 4.2(b), which is more complex than the one represented in Figure 4.2(a). In fact, all the three functions are polylines, but in Figure 4.2(a) all the segments forming the polyline have the same slope, while in Figure 4.2(b) the segments have different slopes, even though all the slopes have the same sign; finally, segments in Figure 4.2(c) have different slopes of different signs. In other words, what is wanted is to express the complexity of a function by counting the number of different slopes and by assigning a higher weight to inversions in the slope sign. It is worth pointing out that such a measure has formally absolutely no relationship with the curvature measure. It is only inspired by an informal intuition of the concept of curvature.

Considering bidimensional regression problems, one can imagine the graphical representation of a GP individual as a polyline, by plotting the points that have fitness cases on the abscissas and the corresponding values of the function coding the GP individual as ordinates and joining those points by segments. Thus, in case of bidimensional problems, all that must be done to calculate the measure is to sort all fitness cases (from the smaller value to the larger one) and to consider the values assumed by the GP individual on those fitness cases. Then the slope of each segment joining these points is calculated. Let $(s_1, s_2, s_3, s_4, s_5, ...)$ be those slopes. The measure is simply calculated as: s_1 - s_2 + s_2 - s_3 + s_3 - s_4 + s_4 - s_5 + ... In this way, if the slopes are all identical, the value of the measure is zero (minimal possible complexity) and if the signs of the slopes of all the consecutive segments change, the contribution of each segment is maximal (because all slopes' absolute values are summed). On the other hand, if two consecutive segments have different slopes with the same sign, their contribution is the subtraction of their respective absolute values (i.e. a contribution larger than zero, but smaller than in the case where the slope sign changes).

In case of multidimensional spaces of features, the projection of the function coding the GP individual on each single dimension is considered. The same calculus as above is executed for each dimension separately and then the average is calculated.

The second complexity measure has been defined in section 4.4 and it is called Graph Based Complexity (GBC). The name of this measure depends on the fact that it is possible to represent it in terms of counting operations on a graph as explained in section 4.4. This complexity measure is inspired by the idea that complex functions should have a larger curvature than simple ones. But, contrarily to the Multi-Slope Complexity Measure, in this case the idea of curvature is quantified using the following intuition: let g be a GP individual; the curvature of g can be expressed by counting the number of pairs of "*close*" training points $\mathbf{x}$ and $\mathbf{y}$ for which the corresponding values $g(\mathbf{x})$ and $g(\mathbf{y})$ are "far". The formal definition is reported in section 4.4. GBC has been introduced to overcome some of the limitations of the MSCM. In particular GBC is rotationally invariant, contrarily to what happens for the MSCM.

The definition of measures to quantify the amount of bloat, overfitting and the complexity of the solutions could be useful to investigate the relations between these phenomena. In this light an accurate experimental phase has been performed. In this experimental phase three different real life applications, have been considered and, as stated in the previous section, a particular attention has been dedicated to the analysis of the results. The objective of this experimental phase was twofold: first of all we were interested in assess the coherence of the proposed measures with respect to the underlying phenomena. The second objective is to find relations between these phenomena.

Experimental results show that:

- all the proposed measures seem to be able to capture the underlying phenomena;
- complexity measure (GBC) increases in case of overfitting;
- bloat and overfitting are independent phenomena: it is possible to have bloat without overfitting and overfitting without bloat;
- overfitting seems to be related to the complexity of the solutions rather than bloat.

8.3 Techniques to enhance GP's generalization ability

The study of the generalization ability of GP is an important problem. In many cases GP is able to find solutions that produce good performances on the training instances, but this performances are not replicated on unseen data. At the end of the learning process, the hope is to find a solution that could achieve on unseen data the same performances obtained on the training instances. Although generalisation of learned solutions is the primary interest of any machine learning technique [74], it was not seriously considered in the field of GP for a long time. Before Kushchu published his work on the generalisation ability of GP [61], there were rather few research dealing with the GP generalisation aspect.

In [8] authors studied the influence of using extensive mutation on the ability of a new GP system called Compiling GP (CGP) to generalise. Authors found that increasing the mutation rate can significantly improve the generalization capabilities of GP. The mechanism by which mutation affects the generalization capability of GP is not entirely clear. What is clear is that changing the balance between mutation and crossover affects the course of GP training substantially.

In [38] authors compared the ability of CGP to generalize with that of other machine learning paradigms. Results show that when CGP was trained on data sets that were not too sparse, CGP performed very well, equaling the generalization capability of other machine learning systems quickly and consistently. Moreover, when CGP was trained on very sparse data sets, CGP produced individuals that generalized almost as well other machine learning systems trained on much larger data sets.

In [69] authors experimented with Tarpeian Control on some symbolic regression problems and tested the side effects of this method on the generalisation ability of GP. The results are contrasted, showing that it can either increase or reduce the generalization power of GP hypotheses, depending on the problem at hand.

In [39] authors investigated two methods to improve generalisation in GP: the selection of the best of run individuals using a three datasets method (training, validation, and test sets), and the application of parsimony pressure in order to reduce the complexity of the solutions. Their experimental results indicate that using a validation set could slightly improve the stability of the best of run solutions on the test sets.

In [108] authors improved GP generalisation using a crossover based similarity measure. Their method is to keep a list of over-fitting individuals and to prevent any individual entering the next generation if it is similar (based on structural distance or a subtree crossover based similarity measure) to one individual in the list.

In [26] authors proposed a new GP system called Relaxed Genetic Programming (RGP). They show how a small degree of relaxation improves the generalization error of the best solutions.

In [25] authors showed the important role of generalisation on GP. They experimentally showed that a technique like Linear Scaling [56] may only be significantly better than standard GP on training data but not superior on testing data. They proposed an approach to improve GP generalisation by combining Linear Scaling and the No Same Mate strategy [44].

The use of semantic to improve generalization ability has recently been proposed . A literature review about semantic methods is proposed in section 7.2.

The method presented in this thesis consists in giving a second chance of mating to individuals belonging to "old" generations (hence the name of the method: "second chance GP").

To accomplish this goal, every k generations (where $k \in \mathbb{N}$ and $k > 1$), the worst $p_r\%$ individuals in the population (where p_r is the *replacement pressure*) are replaced. The individuals that replace them are extracted from the population of k generations before the current one (for this reason, k is called *refresh rate*), and they are chosen from that population using exactly the same selection method used by the standard algorithm.

In some senses, the method simulates the idea that partners do not have to have all the same age, but partners of different ages can mate and produce offspring. Furthermore, differently from the majority of the previously published related contributions, the proposed approach is focused specifically on generalization, with the idea that "old" individuals may be less specialized than "young" ones on training data, and thus potentially more general.

Although original, the idea is inspired by well-known concepts such as short-term memory schemes, that have already been used in evolutionary computation so far.

Motivations for introducing this method are the following. First of all it is possible hypothesize that, generation by generation, the GP individuals become more and more specialized. Even if this behavior can be a good one, the GP individuals could also overfit training data. The insertion of earlier (and probably less specialized) individuals with a good fitness can allow the population diversity to increase, reducing at the same time the risk of overfitting training data. The second motivation is that the processes of selection, crossover and mutation of GP can discard (the former one) or disrupt (the latter ones) good genetic material. Allowing a reinsertion, the probability of good genetic material to be propagated increases. The third motivation is that, also on training data themselves, the evolutionary process may lead the population to convergence towards local optima. The insertion of earlier individuals with a good fitness, while allowing the population diversity to increase, should also reduce the risk of premature convergence and stagnation in local optima.

The "second chance GP" method is characterized by the presence of two parameters: the refresh rate k and the replacement pressure p_r. Thus, their effect has been investigated.

Experimental results have shown that when the parameter k is low, the effect is to increase the probability to insert into the current population individuals with a good fitness that can positively affect the evolution process. On the other hand, a low k value may result in inserting genetic material that does not help in better exploring the search space, since it may be not significantly different with respect to the existing one. An high k value can help in solving the latter issue, but can generate a new problem: the individuals of old populations can have a much worse fitness than the current ones and their reinsertion may produce no effects on the search process. Regarding the second parameter p_r, experimental results have shown taht it can also influence the behavior of the search process. In particular, a low p_r value may result in inserting not enough individuals, with a negligible effect on the evolutionary process. A high p_r value can solve the latter problem but, if too high, can disrupt what has been learned so far.

Other methods to increase the generalization ability of GP involve the use of multi objective optimization. In optimization problems the search bias usually involves only one fitness function. This is equivalent to the use of only one criterion to estimate the quality of a solution. On the other hand, in many situations a good solution is achieved by a compromise between multiple criteria [27].

In this thesis an experimental study of the generalization ability of a set of multi-optimization GP frameworks, comparing them with standard GP and Operator Equalisation [93] is presented. The idea of optimizing more than one criterion on training data to produce more general solutions is not new: a related idea has been used in [39] in the domain of binary classification (where a two objective sort is performed in order to extract a set of non-dominated individuals). In [109] authors have motivated and empirically shown that GP using a Pareto multi-optimization on the training set has a remarkably higher generalization ability than canonic or standard GP (besides counteracting bloat in a more efficient way and maintaining a higher diversity inside the population). Furthermore, in [111], authors defined two measures of complexity for the GP individuals, they used them as further criteria (besides fitness) in a multi-objective system based on Pareto optimization, and they showed that this system is able to counteract bloat and overfitting. Compared to those contributions, in this thesis different objectives have been used and the NSGA-II [28] multi-objective evolutionary method has been used. The functions used in the multi-optimization algorithm are the fitness of the problem and one or more other functions that can intuitively be related to overfitting. These function are (1) the number of the nodes of the tree and (2) the variance of the error between the target and the obtained value. Obviously the former function should be minimized in order to counteract bloat and to reward small-sized trees. The choice of this function is interesting because it immediately demands for a comparison with the bloat control method proposed in [93]. Both techniques, in fact, counteract bloat but the way they do it is different. While in [93] code size reduction is achieved implicitly (following the target length distribution), in multi-optimization this is done explicitly. Regarding the latter function, the minimization of the variance of the error between the target and the obtained value reduces the difference between the shape of the expected (unknown) function and the function described by the given individual. This kind of optimization is supposed to be related to overfitting as already discussed for instance in [72].

Experimental results show that the multi objective optimization system produces solutions with better generalization ability than standard GP and Operator Equalisation. Even though this is in agreement with several previously published results (see for instance [39, 105, 111]), it seems to contradict the fact that small size does not imply better generalization.

We hypothesize that the reason why multi-objective GP generalizes better then standard GP and Operator Equalisation does not reside in the fact that size minimization has been used as an objective. As a confirmation, results on test data obtained considering the variance of the error as a second objective, are comparable with the ones of multi-objective GP using size (the differences between these two systems are not statistically significant).

Thus, we hypothesize that better generalization ability is, so to say, an intrinsic characteristic of multi-optimization systems, which do not depend on the fact that size of solutions is explicitly minimized.

8.4 GP based on semantic

The analysis of GP has been mainly conducted considering syntactical properties. Individuals, or programs, are usually presented in a language of syntactic formalism such as s-expression trees [57], grammars, or graphs [82]. The genetic operators in such GP systems are usually designed to ensure the syntactic closure property, i.e. to produce syntactically valid children from any syntactically valid parent(s). As observed in [77], using such purely syntactical genetic operators, GP evolutionary search is conducted on the syntactical space of programs with the only semantic guidance from the fitness of program measured by the difference of behavior of evolving programs and the target programs (usually on a finite input-output set called fitness cases). Although GP has been shown to be effective in evolving programs for solving different problems using such (finite) behavior-based semantic guidance and pure syntactical genetic operator, this practice is somewhat unusual from real programmers'perspective. Computer programs are not just constrained by syntax but also by semantics. As a normal practice, any change to a program should pay heavy attention to the change in semantics of the program and not just those changes that guarantee to maintain the program syntactical validity.

To amend this deficiency in GP resulting from the lack of semantic guidance on genetic operators, several works on semantic have been proposed in the last years. In [104] one method to incorporate semantic information into GP crossover operators for real-valued symbolic regression problems is proposed. In particular, authors aim to incorporate semantic into the design of new crossover operators so as to maintain greater semantic diversity, and provide higher locality than standard crossover.

In [10] Beadle and Johnson have proposed a semantic-based crossover operator for GP. They showed that their semantically driven crossover operator could help GP in achieving better results and less code bloat on some standard Boolean test problems.

In [77] authors extend the ideas from [10] to investigate the effect of some semantic based guidance on the crossover operator in GP on a family of real-valued symbolic regression problems. They also propose a new form of semantic-aware crossover for GP, which considers approximations of the semantics of the exchanged subtrees.

An extensive literature review regarding methods to include semantic information in the GP process is presented in chapter 7.

In a large number of works, the semantic information has been included by redefining the genetic operators. The idea that has been presented in this thesis is different, and does not require any modification to the standard GP operators. The method (introduced in chapter 7) is inspired by the well-known control bloat method proposed in [92]. The similarities between the two methods are limited to the acceptance criteria of new individuals produced by the genetic operators. Beyond this, the basic idea that characterizes the two methods is different: while the work in [92] builds and uses a distribution based on the syntax of the individuals, the work proposed in this thesis focuses on semantics. Hence, the main idea of the proposed "Semantic niching" method is to use a semantic distribution to guide the evolutionary process. This distribution is used to direct the algorithm toward solutions that are semantically close to the best solution found so far by GP. The main goal is to increase the generalization ability of GP. To do that the proposed algorithm has been design to maintain a high population diversity in the first generations while in the last generations should search the optimal solutions in a small area of the search space.

Regarding the experimental phase 10 regression problems have been chosen, 5 are continuous functions while the other 5 are non-continuous functions. The main objective of this experimental phase was to assess the performance of the proposed method with respect to standard GP and the Parsimony Pressure method [40]. In particular we want to check whether or not the proposed method could improve the generalization ability of GP. On the other hand, we want to control whether or not the proposed method has a significant impact on population diversity. Experimental settings and results have been outlined in sections 7.5 and 7.4. To summarize, experimental results show that:

- the proposed method performs better or in a comparable way to standard GP and to the Parsimony Pressure method on unseen data;
- the proposed method maintains a high degree of population diversity in the first evaluations of the learning process while it reduces the population diversity in the following evaluations. This is a good indicator of the fact that the proposed method is able to converge on solutions that are semantically better;
- the proposed method seems to be able to cover areas of the search space characterized by the presence of solutions with good fitness;
- the proposed method tends to create large individuals.

CHAPTER 9

Future works

Contents

This thesis contains some contributions to face some open problems in the field of GP. Nevertheless, all these problems are not completely fixed. In particular, finding a definitive solution to avoid overfitting still remains a very difficult task; in the same way, finding the relations between phenomena that characterize the execution of a GP process requires further studies. However, results presented in this thesis are encouraging: some aspects related to the behaviour of GP systems have been analyzed and some conclusions have been drawn. This analysis opens up further research directions. New research directions, opened by the work presented in this thesis, are discussed in this chapter. In particular, section 9.1 describes possible future works in the field of GP measures, while section 9.2 presents some idea to improve the generalization ability of GP systems.

9.1 GP Measures

Future activities regarding the definition of measures to capture phenomena that characterize a GP system, could be conducted in the following directions:

- improving the efficiency of the proposed measures;
- finding relations between bloat, overfitting and complexity;
- defining a GP system that integrates the proposed measures to allow GP to find better solutions.

Regarding the first point, it is possible to note that the definition of the measures proposed in 4 presents some problems:

- the bloat measure compares the fitness and the length of programs at a given generation with the fitness and the length of programs at generation zero. This is a good starting point, given that it is possible to assume that there is no bloat at generation zero. Nevertheless, the initialization algorithm (that defines the population at generation zero) has biases both on program length and on fitness. These biases clearly affect the measure and should be taken into account. Or otherwise, a measure that does not take generation zero as a reference deserves to be defined and investigated;
- the overfitting measure clearly depends on how training and test sets have been chosen. Given that at each GP run a different (random) partitioning of the dataset into training and test sets has been used, it may happen that exactly the same population has two different values of the overfitting measure in two different

runs. New versions of this measure deserve to be defined, for instance alternating training and test data in a crossvalidation-like way;

- using the same idea as in equation (4.1) for measuring the relationship between training and test fitness (i.e. comparing the values at the current generation with the corresponding ones at generation zero) is definitely unsuitable. In fact, at generation zero the population has been created by the (typically random) initialization algorithm and the solutions in the population have not undergone any evolution yet. For this reason, there is typically no reason why training fitness should be better than test fitness at generation zero. On the other hand, these two values are typically quite similar and being the test set often smaller than the training set, it is even more likely that the average training fitness is worse than the average test fitness at generation zero. For this reason, using a formula like equation (4.1) to quantify overfitting would produce a measure that is more affected by the values of training and test fitness at generation zero than by the course of the learning process. As a consequence of the fact that training and test fitness values are very similar to each other at generation zero (or test fitness is even better than training fitness), a measure like equation (4.1) would very often produce very high overfitting values, even when it is clearly not the case;

- the different functional complexity measures are formally weakly related with curvature. They are empirical indicator, that captures the idea of curvature only from an intuitive viewpoint. Compared to other (more formal) measures they have the advantage of being simple and computationally cheap, but the lack of formality of these measures may cause problems.

Working on these problems could lead to the definition of measures that are problem and data independent. Such measures could be used to control the phenomena that characterize a GP run, without any bias. This results in better and reliable measures. Starting from this point, it would be much easy to find the relations that link bloat, overfitting and functional complexity. In the work proposed in this thesis several conclusions about these phenomena can be drawn, but the lack of formalisms in defining some measures sometimes leads to contrasting results.

Defining better measures not only allows to determine more easily existing relations between the underlying phenomena, but would allow the use of these measures within a GP system, with the aim of controlling the learning process and taking the actions necessary to limit the effects of these phenomena.

These future research directions could have an important impact on the application of GP. It could be possible to stop the learning process when overfitting starts to appear or when solutions start to bloat. Hence, the possible scenarios resulting from this research are promising.

9.2 GP Generalization Ability

In this section possible research directions on improving the generalization ability of GP are outlined. This problem is interesting not only for GP, but also for many machine learning techniques. Nowadays, methods to increase the generalization ability of a GP system can be divided in two categories:

- methods that work considering structural properties;
- methods that work considering semantic properties;

In this thesis different methods to solve this problem have been studied. One method is based on semantic (chapter 7), one method used multi objective optimization combining semantic and structural properties (chapter 5) while one method (chapter 6) is inspired by the concept of "short-term" memory, which is typical of Tabu Search [41]. All these methods present strengths and drawbacks. The method proposed in chapter 7 has given better performances with respect to other GP systems, but it generates large trees. Moreover, this method is characterized by the presence of an acceptance threshold AT: this threshold influences the final performances of the system and must be carefully chosen.

The performances of the method proposed in chapter 6 are also based on the values of two distinct parameters that strongly influence the behaviour of the GP system.

Regarding the multi objective optimization framework proposed in chapter 5, a further analysis is needed. In particular, it is not clear how the different objectives interact during the evolutionary process.

While some conclusions can be drawn from the work presented in this thesis, defining a method to completely avoid overfitting is an open problem. The fact is that this problem depends on several factors that are independent from the learning process. For instance, it is clear that the particular training and test sets used could influence the generalization ability of the learned solution. Hence, different GP runs with different training and test sets may lead to solutions that present a different amount of overfitting.

In this light, the process that could lead to a system without overfitting should consider also these "external" factors. Conversely, considering only the learning process, it is clear that it is necessary to combine both structural and semantic information. As stated in chapter 7 semantic plays a central role in the evolutionary process, but semantic alone is not enough to guarantee a final solution with a good generalization ability. Thus, a learning technique should optimize more than one criterion; in this light an accurate analysis of multi optimization algorithms could lead to interesting scenarios.

Bibliography

[1] Lee Altenberg. The evolution of evolvability in genetic programming. In Kenneth E. Kinnear, Jr., editor, *Advances in Genetic Programming*, chapter 3, pages 47–74. MIT Press, 1994. 36

[2] Nur Merve Amil, Nicolas Bredeche, Christian Gagné, Sylvain Gelly, Marc Schoenauer, and Olivier Teytaud. A statistical learning perspective of genetic programming. In *Proceedings of the 12th European Conference on Genetic Programming*, EuroGP '09, pages 327–338, Berlin, Heidelberg, 2009. Springer-Verlag. 6, 104

[3] Peter J. Angeline. *Genetic programming and emergent intelligence*, pages 75–97. MIT Press, Cambridge, MA, USA, 1994. 36

[4] Francesco Archetti, Stefano Lanzeni, Enza Messina, and Leonardo Vanneschi. Genetic programming for human oral bioavailability of drugs. In Maarten Keijzer, Mike Cattolico, Dirk Arnold, Vladan Babovic, Christian Blum, Peter Bosman, Martin V. Butz, Carlos Coello Coello, Dipankar Dasgupta, Sevan G. Ficici, James Foster, Arturo Hernandez-Aguirre, Greg Hornby, Hod Lipson, Phil McMinn, Jason Moore, Guenther Raidl, Franz Rothlauf, Conor Ryan, and Dirk Thierens, editors, *GECCO 2006: Proc. of the 8th annual conference on Genetic and evolutionary computation*, volume 1, pages 255–262. ACM Press, 2006. 104

[5] T Bäck, D.B Fogel, and Z Michalewicz, editors. *Evolutionary Computation 1: Basic Algorithms and Operators*. Institute of Physics Publishing, Bristol, 2000. 15

[6] Thomas Bäck. *Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms*. Oxford University Press, Oxford, UK, 1996. 15

[7] W. Banzhaf, F. D. Francone, and P. Nordin. The effect of extensive use of the mutation operator on generalization in genetic programming using sparse data sets. In W. Ebeling *et al*, editor, *4th Int. Conf. on Parallel Problem Solving from Nature (PPSN96)*, pages 300–309. Springer, Berlin, 1996. 6, 104

[8] Wolfgang Banzhaf, Frank D. Francone, and Peter Nordin. The effect of extensive use of the mutation operator on generalization in genetic programming using sparse data sets. In *In Parallel Problem Solving from Nature IV, Proceedings of the International Conference on Evolutionary Computation, edited by*, pages 300–309. Springer Verlag, 1996. 179

[9] Wolfgang Banzhaf, Peter Nordin, Robert E. Keller, and Frank D. Francone. *Genetic Programming - An Introduction; On the Automatic Evolution of Computer Programs and its Applications*. Morgan Kaufmann, San Francisco, CA, USA, January 1998. 15, 35, 36, 42, 174

[10] Lawrence Beadle and Colin Johnson. Semantically driven crossover in genetic programming. In Jun Wang, editor, *Proceedings of the IEEE World Congress*

on Computational Intelligence, pages 111–116, Hong Kong, 1-6 2008. 11, 138, 140, 185

[11] Lawrence Beadle and Colin G. Johnson. Semantic analysis of program initialisation in genetic programming. *Genetic Programming and Evolvable Machines*, 10(3):307–337, September 2009. 10, 137, 140

[12] Lawrence Beadle and Colin G Johnson. Semantically driven mutation in genetic programming. In Andy Tyrrell, editor, *2009 IEEE Congress on Evolutionary Computation*, pages 1336–1342, Trondheim, Norway, 18-21 May 2009. IEEE Computational Intelligence Society, IEEE Press. 140

[13] Lee A. Becker and Mukund Seshadri. Comprehensibility and overfitting avoidance in genetic programming for technical trading rules. Technical report, Worcester Polytechnic Institute, May 2003. 5

[14] Tobias Blickle and Lothar Thiele. Genetic programming and redundancy. In J. Hopf, editor, *Genetic Algorithms within the Framework of Evolutionary Computation (Workshop at KI-94, Saarbrücken)*, pages 33–38, Im Stadtwald, Building 44, D-66123 Saarbrücken, Germany, 1994. Max-Planck-Institut für Informatik (MPI-I-94-241). 36

[15] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. K. Warmuth. Occam's razor. *Information Processing Letters*, 24(6):377–380, 1987. Cited By (since 1996): 154. 5

[16] Randal E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Trans. Comput.*, 35:677–691, August 1986. 10, 138

[17] Wray Buntine and Tim Niblett. A further comparison of splitting rules for decision-tree induction. *Machine Learning*, 8:75–85, 1992. 10.1007/BF00994006. 5

[18] Edmund Burke, Steven Gustafson, Graham Kendall, and Natalio Krasnogor. Advanced population diversity measures in genetic programming. In Juan J. Merelo-Guervos, Panagiotis Adamidis, Hans-Georg Beyer, Jose-Luis Fernandez-Villacanas, and Hans-Paul Schwefel, editors, *Parallel Problem Solving from Nature - PPSN VII*, number 2439 in Lecture Notes in Computer Science, LNCS, pages 341–350, Granada, Spain, 7-11 September 2002. Springer-Verlag. 9, 137

[19] Edmund K. Burke, Steven Gustafson, and Graham Kendall. Diversity in genetic programming: An analysis of measures and correlation with fitness. *IEEE Transactions on Evolutionary Computation*, 8(1):47–62, 2004. 9, 137, 150

[20] J. Casey. *Exploiting Curvature*. Wiesbaden, Germany: Vieweg, 1996. 7, 51, 176

[21] Mauro Castelli, Luca Manzoni, Sara Silva, and Leonardo Vanneschi. A quantitative study of learning and generalization in genetic programming. In Silva et al. [94], pages 25–36. 65

[22] M.J. Cavaretta and K. Chellapilla. Data mining using genetic programming: the implications of parsimony on generalization error. In *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*, volume 2, pages 3 vol. (xxxvii+2348), 1999. 40

[23] Peter Clark and Tim Niblett. The cn2 induction algorithm. *Mach. Learn.*, 3:261–283, March 1989. 5

[24] G.W. Corder and D.I. Foreman. *Nonparametric statistics for Non-Statisticians*. Wiley, New York, 2009. 113, 151

[25] Dan Costelloe and Conor Ryan. On improving generalisation in genetic programming. In Leonardo Vanneschi, Steven Gustafson, Alberto Moraglio, Ivanoe De Falco, and Marc Ebner, editors, *Genetic Programming*, volume 5481 of *Lecture Notes in Computer Science*, pages 61–72. Springer Berlin / Heidelberg, 2009. 180

[26] L. E. Da Costa and J.-A. Landry. Relaxed genetic programming. In M. Keijzer *et al.*, editor, *GECCO 2006: Proceedings of the 8th annual conference on Genetic and evolutionary computation*, volume 1, pages 937–938, Seattle, Washington, USA, 8-12 July 2006. ACM Press. 104, 180

[27] K. Deb. *Multiobjective Optimization Using Evolutionary Algorithms.* Wiley, Chichester, U.K.:, 2001. 8, 88, 182

[28] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and T. Meyarivan. A fast elitist multi-objective genetic algorithm: Nsga-ii. *IEEE Transactions on Evolutionary Computation*, 6:182–197, 2000. 82, 87, 88, 183

[29] Patrik D'haeseleer and Jason Bluming. *Effects of locality in individual and population evolution*, pages 177–198. MIT Press, Cambridge, MA, USA, 1994. 10, 137

[30] S. Dignum and R. Poli. Crossover, sampling, bloat and the harmful effects of size limits. In Michael O'Neill, Leonardo Vanneschi, Steven Gustafson, Anna Isabel Esparcia Alcazar, Ivanoe De Falco, Antonio Della Cioppa, and Ernesto Tarantino, editors, *Proceedings of the 11th European Conference on Genetic Programming, EuroGP 2008*, volume 4971 of *Lecture Notes in Computer Science*, pages 158–169. Springer, 2008. 83

[31] Stephen Dignum and Riccardo Poli. Generalisation of the limiting distribution of program sizes in tree-based genetic programming and analysis of its effects on bloat. In Dirk Thierens, Hans-Georg Beyer, Josh Bongard, Jurgen Branke, John Andrew Clark, Dave Cliff, Clare Bates Congdon, Kalyanmoy Deb, Benjamin Doerr, Tim Kovacs, Sanjeev Kumar, Julian F. Miller, Jason Moore, Frank Neumann, Martin Pelikan, Riccardo Poli, Kumara Sastry, Kenneth Owen Stanley, Thomas Stutzle, Richard A Watson, and Ingo Wegener, editors, *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, volume 2, pages 1588–1595, London, 7-11 July 2007. ACM Press. 38, 39

[32] Stephen Dignum and Riccardo Poli. Generalisation of the limiting distribution of program sizes in tree-based genetic programming and analysis of its effects on bloat. In Dirk Thierens, Hans-Georg Beyer, Josh Bongard, Jurgen Branke, John Andrew Clark, Dave Cliff, Clare Bates Congdon, Kalyanmoy Deb, Benjamin Doerr, Tim Kovacs, Sanjeev Kumar, Julian F. Miller, Jason Moore, Frank Neumann, Martin Pelikan, Riccardo Poli, Kumara Sastry, Kenneth Owen Stanley, Thomas Stutzle, Richard A Watson, and Ingo Wegener, editors, *GECCO*

'07: Proceedings of the 9th annual conference on Genetic and evolutionary computation, volume 2, pages 1588–1595, London, 7-11 July 2007. ACM Press. 83

[33] Stephen Dignum and Riccardo Poli. Operator equalisation and bloat free GP. In Michael O'Neill, Leonardo Vanneschi, Steven Gustafson, Anna Isabel Esparcia Alcazar, Ivanoe De Falco, Antonio Della Cioppa, and Ernesto Tarantino, editors, *Proceedings of the 11th European Conference on Genetic Programming, EuroGP 2008*, volume 4971 of *Lecture Notes in Computer Science*, pages 110–121, Naples, 26-28 March 2008. Springer. 41, 84

[34] Pedro Domingos. The role of occam's razor in knowledge discovery. *Data Min. Knowl. Discov.*, 3:409–425, December 1999. 5

[35] R. O. Duda and P. E. Hart. *Pattern Classification and Scene Analysis*. New York: Wiley, 1973. 72

[36] David B. Fogel. What is evolutionary computation? *IEEE Spectr.*, 37:26–32, February 2000. 2

[37] F. D. Francone, P. Nordin, and W. Banzhaf. Benchmarking the generalization capabilities of a compiling genetic programming system using sparse data sets. In J. R. Koza *et al.*, editor, *Genetic Programming: Proceedings of the first annual conference*, pages 72–80. MIT Press, Cambridge, 1996. 6, 103

[38] Frank D. Francone. Benchmarking the generalization capabilities of a compiling genetic programming system using sparse data sets. In *Genetic Programming 1996: Proceedings of the First Annual Conference*, pages 72–80. MIT Press, 1996. 179

[39] C. Gagné, M. Schoenauer, M. Parizeau, and M. Tomassini. Genetic programming, validation sets, and parsimony pressure. In P. Collet *et al.*, editor, *Genetic Programming, 9th European Conference, EuroGP2006*, Lecture Notes in Computer Science, LNCS 3905, pages 109–120. Springer, Berlin, Heidelberg, New York, 2006. 6, 82, 102, 104, 180, 183, 184

[40] Christian Gagné, Marc Schoenauer, Marc Parizeau, and Marco Tomassini. Genetic programming, validation sets, and parsimony pressure. In Pierre Collet, Marco Tomassini, Marc Ebner, Steven Gustafson, and Anikó Ekárt, editors, *Proceedings of the 9th European Conference on Genetic Programming*, volume 3905 of *Lecture Notes in Computer Science*, pages 109–120, Budapest, Hungary, 10 - 12 April 2006. Springer. 42, 146, 186

[41] Fred Glover and Manuel Laguna. *Tabu Search*. Kluwer Academic Publishers, Norwell, MA, USA, 1997. 7, 104, 192

[42] Faustino J. Gomez, Julian Togelius, and Juergen Schmidhuber. Measuring and optimizing behavioral complexity for evolutionary reinforcement learning. In *Proceedings of the 19th International Conference on Artificial Neural Networks: Part II*, ICANN '09, pages 765–774, Berlin, Heidelberg, 2009. Springer-Verlag. 6

[43] John J. Grefenstette and James E. Baker. How genetic algorithms work: a critical look at implicit parallelism. In *Proceedings of the third international conference on Genetic algorithms*, pages 20–27, San Francisco, CA, USA, 1989. Morgan Kaufmann Publishers Inc. 30

[44] S. Gustafson, E.K. Burke, and N. Krasnogor. On improving genetic programming for symbolic regression. In *Evolutionary Computation, 2005. The 2005 IEEE Congress on*, volume 1, pages 912–919 Vol.1, sept. 2005. 180

[45] Steven Gustafson. *An Analysis of Diversity in Genetic Programming*. PhD thesis, School of Computer Science and Information Technology, University of Nottingham, Nottingham, England, February 2004. 10, 137

[46] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The elements of statistical learning: data mining, inference and prediction*. Springer, 2 edition. 43

[47] Ting Hu, Joshua L. Payne, Wolfgang Banzhaf, and Jason H. Moore. Robustness, evolvability, and accessibility in linear genetic programming. In Silva et al. [94], pages 13–24. 141

[48] J. and Rissanen. Modeling by shortest data description. *Automatica*, 14(5):465 – 471, 1978. 5

[49] David Jackson. Phenotypic diversity in initial genetic programming populations. In Anna Isabel Esparcia-Alcazar, Aniko Ekart, Sara Silva, Stephen Dignum, and A. Sima Uyar, editors, *Proceedings of the 13th European Conference on Genetic Programming, EuroGP 2010*, volume 6021 of *LNCS*, pages 98–109, Istanbul, 7-9 April 2010. Springer. 140, 141

[50] David Jackson. Promoting phenotypic diversity in genetic programming. In Robert Schaefer, Carlos Cotta, Joanna Kolodziej, and Guenter Rudolph, editors, *PPSN 2010 11th International Conference on Parallel Problem Solving From*

Nature, volume 6239 of *Lecture Notes in Computer Science*, pages 472–481. Springer, Krakow, Poland, 11-15 September 2010. 141

[51] K. G. Janardan. Weighted lagrange distributions and their characterizations. *SIAM J. Appl. Math.*, 47:411–415, April 1987. 39

[52] Konanur G. Janardan and B. Raja Rao. Lagrange distributions of the second kind and weighted distributions. *SIAM Journal on Applied Mathematics*, 43(2):302–313, 1983. 39

[53] David D. Jensen and Paul R. Cohen. Multiple comparisons in induction algorithms. *Mach. Learn.*, 38:309–338, March 2000. 5

[54] T. Jones and S. Forrest. Fitness distance correlation as a measure of problem difficulty for genetic algorithms. pages 184–192. Morgan Kaufmann, 1995. 72

[55] Tatiana Kalganova and Julian Miller. Evolving more efficient digital circuits by allowing circuit layout evolution and multi-objective fitness. In *The First NASA/DoD Workshop on Evolvable Hardware, Pasadena, California, IEEE Computer Society*, pages 54–63. IEEE, 1999. 40

[56] Maarten Keijzer. Improving symbolic regression with interval arithmetic and linear scaling. In Conor Ryan, Terence Soule, Maarten Keijzer, Edward Tsang, Riccardo Poli, and Ernesto Costa, editors, *Genetic Programming*, volume 2610 of *Lecture Notes in Computer Science*, pages 275–299. Springer Berlin / Heidelberg, 2003. 180

[57] John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA, 1992. 23, 36, 39, 46, 47, 55, 69, 107, 108, 184

[58] J.R. Koza. A genetic approach to the truck backer upper problem and the inter-twined spiral problem. In *Neural Networks, 1992. IJCNN., International Joint Conference on*, volume 4, pages 310 –318 vol.4, jun 1992. 24

[59] Krzysztof Krawiec and Pawel Lichocki. Approximating geometric crossover in semantic space. In Guenther Raidl, Franz Rothlauf, Giovanni Squillero, Rolf Drechsler, Thomas Stuetzle, Mauro Birattari, Clare Bates Congdon, Martin Middendorf, Christian Blum, Carlos Cotta, Peter Bosman, Joern Grahl, Joshua Knowles, David Corne, Hans-Georg Beyer, Ken Stanley, Julian F. Miller, Jano van Hemert, Tom Lenaerts, Marc Ebner, Jaume Bacardit, Michael O'Neill, Massimiliano Di Penta, Benjamin Doerr, Thomas Jansen, Riccardo Poli, and Enrique Alba, editors, *GECCO '09: Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 987–994, Montreal, 8-12 July 2009. ACM. 139

[60] Gabriel Kronberger, Michael Kommenda, and Michael Affenzeller. Overfitting detection and adaptive covariant parsimony pressure for symbolic regression. In *Proceedings of the 13th annual conference companion on Genetic and evolutionary computation*, GECCO '11, pages 631–638, New York, NY, USA, 2011. ACM. 43

[61] I. Kushchu. An evaluation of evolutionary generalization in genetic programming. *Artificial Intelligence Review*, 18(1):3–14, 2002. 6, 103, 179

[62] G. N. Lance and W. T. Williams. Mixed-data classificatory programs i - agglomerative systems. *Australian Computer Journal*, 1(1):15–20, 1967. 149

[63] W. B. Langdon and R. Poli. Fitness causes bloat. Technical Report CSRP-97-09, University of Birmingham, School of Computer Science, Birmingham, B15 2TT, UK, 24 February 1997. 38

[64] W. B. Langdon and Riccardo Poli. *Foundations of Genetic Programming*. Springer-Verlag, 2002. 40

[65] Moshe Looks. On the behavioral diversity of random programs. In Dirk Thierens, Hans-Georg Beyer, Josh Bongard, Jurgen Branke, John Andrew Clark, Dave Cliff, Clare Bates Congdon, Kalyanmoy Deb, Benjamin Doerr, Tim Kovacs, Sanjeev Kumar, Julian F. Miller, Jason Moore, Frank Neumann, Martin Pelikan, Riccardo Poli, Kumara Sastry, Kenneth Owen Stanley, Thomas Stutzle, Richard A Watson, and Ingo Wegener, editors, *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, volume 2, pages 1636–1642, London, 7-11 July 2007. ACM Press. 10, 137

[66] Sean Luke. Modification point depth and genome growth in genetic programming. *Evolutionary Computation*, 11(1):67–106, Spring 2003. 38

[67] Sean Luke and Liviu Panait. Lexicographic parsimony pressure. In 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, editors, *GECCO 2002: Proceedings of the*

Genetic and Evolutionary Computation Conference, pages 829–836, New York, 9-13 July 2002. Morgan Kaufmann Publishers. 55, 146

[68] Sean Luke and Liviu Panait. A comparison of bloat control methods for genetic programming. *Evolutionary Computation*, 14(3):309–344, Fall 2006. 39

[69] Sébastien Mahler, Denis Robilliard, and Cyril Fonlupt. Tarpeian bloat control and generalization accuracy. In Maarten Keijzer, Andrea Tettamanzi, Pierre Collet, Jano van Hemert, and Marco Tomassini, editors, *Genetic Programming*, volume 3447 of *Lecture Notes in Computer Science*, pages 141–141. Springer Berlin / Heidelberg, 2005. 179

[70] Nicholas Freitag McPhee, Brian Ohs, and Tyler Hutchison. Semantic building blocks in genetic programming. Working Paper Series Volume 3 Number 2, University of Minnesota Morris, 600 East 4th Street, Morris, MN 56267, USA, 12 December 2007. 139

[71] Nicholas Freitag McPhee, Brian Ohs, and Tyler Hutchison. Semantic building blocks in genetic programming. In *Proceedings of the 11th European conference on Genetic programming*, EuroGP'08, pages 134–145, Berlin, Heidelberg, 2008. Springer-Verlag. 10, 138

[72] Ingo Mierswa. Controlling overfitting with multi-objective support vector machines. In *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 1830–1837, New York, NY, USA, 2007. ACM. 89, 184

[73] John Mingers. An empirical comparison of pruning methods for decision tree induction. *Machine Learning*, 4:227–243, 1989. 10.1023/A:1022604100933. 5

[74] Tom M. Mitchell. *Machine Learning*. McGraw-Hill, 1997. 4, 5, 42, 44, 47, 175, 176, 179

[75] J.-M. Morvan. *Generalized Curvatures*. Springer. Series: Geometry and Computing, Vol. 2, 2008. 51

[76] Annie Wu Naval, Annie S. Wu, Alan C. Schultz, and Arvin Agah. Evolving control for distributed micro air vehicles. In *In IEEE Conference on Computational Intelligence in Robotics and Automation*, pages 174–179, 1999. 40

[77] Quang Nguyen, Xuan Nguyen, and Michael O'Neill. Semantic aware crossover for genetic programming: The case for real-valued function regression. In Leonardo Vanneschi, Steven Gustafson, Alberto Moraglio, Ivanoe De Falco, and Marc Ebner, editors, *Genetic Programming*, volume 5481 of *Lecture Notes in Computer Science*, pages 292–302. Springer Berlin / Heidelberg, 2009. 185

[78] Thi Hien Nguyen and Xuan Hoai Nguyen. A brief overview of population diversity measures in genetic programming. In The Long Pham, Hai Khoi Le, and Xuan Hoai Nguyen, editors, *Proceedings of the Third Asian-Pacific workshop on Genetic Programming*, pages 128–139, Military Technical Academy, Hanoi, VietNam, 2006. 9, 137

[79] R. Poli, W. B. Langdon, and Stephen Dignum. On the limiting distribution of program sizes in tree-based genetic programming. Technical Report CSM-464, Department of Computer Science, University of Essex, December 2006. 39, 83

[80] R. Poli and N. F. McPhee. Covariant parsimony pressure for genetic programming. Technical Report CES-480, Department of Computing and Electronic Systems, University of Essex, UK, 2008. 43

[81] Riccardo Poli. A simple but theoretically-motivated method to control bloat in genetic programming. In Conor Ryan, Terence Soule, Maarten Keijzer, Edward Tsang, Riccardo Poli, and Ernesto Costa, editors, *Genetic Programming, Proceedings of EuroGP'2003*, volume 2610 of *LNCS*, pages 204–217, Essex, 14-16 April 2003. Springer-Verlag. 39

[82] Riccardo Poli, William B. Langdon, and Nicholas F. Mcphee. *A field guide to genetic programming*. March 2008. 4, 15, 46, 90, 146, 184

[83] Riccardo Poli, William B. Langdon, Nicholas F. McPhee, and John R. Koza. Genetic programming an introductory tutorial and a survey of techniques and applications. Technical Report CES-475, Department of Computing and Electronic Systems, University of Essex, UK, October 2007. 2

[84] Riccardo Poli, Nicholas F. McPhee, and Leonardo Vanneschi. The impact of population size on code growth in GP: analysis and empirical validation. In Maarten Keijzer, Giuliano Antoniol, Clare Bates Congdon, Kalyanmoy Deb, Benjamin Doerr, Nikolaus Hansen, John H. Holmes, Gregory S. Hornby, Daniel Howard, James Kennedy, Sanjeev Kumar, Fernando G. Lobo, Julian Francis Miller, Jason Moore, Frank Neumann, Martin Pelikan, Jordan Pollack, Kumara Sastry, Kenneth Stanley, Adrian Stoica, El-Ghazali Talbi, and Ingo Wegener, editors, *GECCO '08: Proceedings of the 10th annual conference on Genetic*

and evolutionary computation, pages 1275–1282, Atlanta, GA, USA, 12-16 July 2008. ACM. 83

[85] Riccardo Poli and Nicholas Freitag McPhee. Parsimony pressure made easy. In Conor Ryan and Maarten Keijzer, editors, *GECCO*, pages 1267–1274. ACM, 2008. 40

[86] Riccardo Poli, Nicholas Freitag McPhee, Luca Citi, and Ellery Crane. Memory with memory in genetic programming. *J. Artif. Evol. App.*, 2009:2:1–2:16, January 2009. 128

[87] Justinian Rosca. Entropy-driven adaptive representation. In *Proceedings of the Workshop on Genetic Programming: From Theory to Real-World Applications*, pages 23–32. Morgan Kaufmann. 10, 137

[88] Justinian P. Rosca. Genetic programming exploratory power and the discovery of functions. In John Robert McDonnell, Robert G. Reynolds, and David B. Fogel, editors, *Evolutionary Programming IV Proceedings of the Fourth Annual Conference on Evolutionary Programming*, pages 719–736, San Diego, CA, USA, 1-3 March 1995. MIT Press. 9, 137

[89] Michael Schmidt and Hod Lipson. Symbolic regression of implicit equations. In Rick L. Riolo, Una-May O'Reilly, and Trent McConaghy, editors, *Genetic Programming Theory and Practice VII*, Genetic and Evolutionary Computation, chapter 5, pages 73–85. Springer, Ann Arbor, 14-16 May 2009. 43

[90] Hans-Paul Schwefel. *Evolution and optimum seeking*. Sixth-generation computer technology series. Wiley, 1995. 16

[91] Sara Silva and Ernesto Costa. Dynamic limits for bloat control in genetic programming and a review of past and current bloat theories. *Genetic Programming and Evolvable Machines*, 10(2):141–179, 2009. 85

[92] Sara Silva and Stephen Dignum. Extending operator equalisation: Fitness based self adaptive length distribution for bloat free GP. In Leonardo Vanneschi, Steven Gustafson, Alberto Moraglio, Ivanoe De Falco, and Marc Ebner, editors, *Proceedings of the 12th European Conference on Genetic Programming, EuroGP 2009*, volume 5481 of *LNCS*, pages 159–170, Tuebingen, April 15-17 2009. Springer. 11, 136, 144, 169, 170, 186

[93] Sara Silva and Stephen Dignum. Extending operator equalisation: Fitness based self adaptive length distribution for bloat free gp. In Leonardo Vanneschi, Steven Gustafson, Alberto Moraglio, Ivanoe De Falco, and Marc Ebner, editors, *Genetic Programming*, volume 5481 of *Lecture Notes in Computer Science*, pages 159–170. Springer Berlin / Heidelberg, 2009. 12, 41, 55, 82, 84, 86, 183

[94] Sara Silva, James A. Foster, Miguel Nicolau, Penousal Machado, and Mario Giacobini, editors. *Genetic Programming - 14th European Conference, EuroGP 2011, Torino, Italy, April 27-29, 2011. Proceedings*, volume 6621 of *Lecture Notes in Computer Science*. Springer, 2011. 197, 202

[95] Sara Silva and Leonardo Vanneschi. Operator equalisation, bloat and overfitting: a study on human oral bioavailability prediction. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 1115–1122, New York, NY, USA, 2009. ACM. 4, 8, 43, 46, 84, 94, 101

[96] Guido Smits and Mark Kotanchek. Pareto-front exploitation in symbolic regression. In Una-May O'Reilly, Tina Yu, Rick L. Riolo, and Bill Worzel, editors, *Genetic Programming Theory and Practice II*, chapter 17, pages 283–299. Springer, Ann Arbor, 13-15 May 2004. 43

[97] Terence Soule and James A. Foster. Code size and depth flows in genetic programming. In John R. Koza, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max Garzon, Hitoshi Iba, and Rick L. Riolo, editors, *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pages 313–320, Stanford University, CA, USA, 13-16 July 1997. Morgan Kaufmann. 37, 38

[98] Walter Alden Tackett. *Recombination, selection, and the genetic construction of computer programs*. PhD thesis, Los Angeles, CA, USA, 1994. Not available from Univ. Microfilms Int. 36, 39

[99] Belpae Tony. Evolving visual feature detectors. In *Proceedings of the 5th European Conference on Advances in Artificial Life*, ECAL '99, pages 266–270, London, UK, 1999. Springer-Verlag. 40

[100] C Tricot. *Curves and Fractal Dimension*. Springer, Heidelberg, 1995. 44

[101] Leonardo Trujillo, Sara Silva, Pierrick Legrand, and Leonardo Vanneschi. An empirical study of functional complexity as an indicator of overfitting in genetic programming. In Sara Silva, James A. Foster, Miguel Nicolau, Mario Giacobini, and Penousal Machado, editors, *Proceedings of the 14th European Conference on Genetic Programming, EuroGP 2011*, volume 6621 of *LNCS*, pages 263–274, Turin, Italy, 27-29 April 2011. Springer Verlag. 44

[102] Cliodhna Tuite, Alexandros Agapitos, Michael O'Neill, and Anthony Brabazon. A preliminary investigation of overfitting in evolutionary driven model induction: Implications for financial modelling. In Cecilia Di Chio, Anthony Brabazon, Gianni Di Caro, Rolf Drechsler, Marc Ebner, Muddassar Farooq, Joern Grahl, Gary Greenfield, Christian Prins, Juan Romero, Giovanni Squillero, Ernesto Tarantino, Andrea G. B. Tettamanzi, Neil Urquhart, and A. Sima Uyar, editors, *Applications of Evolutionary Computing, EvoApplications 2011: EvoCOMNET, EvoFIN, EvoHOT, EvoMUSART, EvoSTIM, EvoTRANSLOG*, volume 6625 of *LNCS*, pages 121–130, Turin, Italy, 27-29 April 2011. Springer Verlag. 104

[103] Nguyen Quang Uy, Nguyen Xuan Hoai, Michael O'Neill, and Bob McKay. The role of syntactic and semantic locality of crossover in genetic programming. In Robert Schaefer, Carlos Cotta, Joanna Kolodziej, and Guenter Rudolph, editors, *PPSN 2010 11th International Conference on Parallel Problem Solving From Nature*, volume 6239 of *Lecture Notes in Computer Science*, pages 533–542, Krakow, Poland, 11-15 September 2010. Springer. 139

[104] Nguyen Quang Uy, Nguyen Xuan Hoai, Michael O'Neill, R. I. McKay, and Edgar Galvan-Lopez. Semantically-based crossover in genetic programming: application to real-valued symbolic regression. *Genetic Programming and Evolvable Machines*. Online first. 139, 185

[105] L. Vanneschi, D. Rochat, and M. Tomassini. Multi-optimization for generalization in symbolic regression using genetic programming. In G. Nicosia *et al.*,

editor, *Proceedings of the second annual Italian Workshop on Artificial Life and Evolutionary Computation (WIVACE 2007)*, 2007. 82, 102, 184

[106] Leonardo Vanneschi, Francesco Archetti, Mauro Castelli, and Ilaria Giordani. Classification of oncologic data with genetic programming. *Journal of Artificial Evolution and Applications*, 2009, 2009. 82

[107] Leonardo Vanneschi, Mauro Castelli, and Sara Silva. Measuring bloat, overfitting and functional complexity in genetic programming. In Juergen Branke, Martin Pelikan, Enrique Alba, Dirk V. Arnold, Josh Bongard, Anthony Brabazon, Juergen Branke, Martin V. Butz, Jeff Clune, Myra Cohen, Kalyanmoy Deb, Andries P Engelbrecht, Natalio Krasnogor, Julian F. Miller, Michael O'Neill, Kumara Sastry, Dirk Thierens, Jano van Hemert, Leonardo Vanneschi, and Carsten Witt, editors, *GECCO '10: Proceedings of the 12th annual conference on Genetic and evolutionary computation*, pages 877–884, Portland, Oregon, USA, 7-11 July 2010. ACM. 43, 44, 65, 67, 70, 76, 104, 113, 117, 121, 128, 176

[108] Leonardo Vanneschi and Steven Gustafson. Using crossover based similarity measure to improve genetic programming generalization ability. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, GECCO '09, pages 1139–1146, New York, NY, USA, 2009. ACM. 180

[109] Leonardo Vanneschi, Denis Rochat, and Marco Tomassini. Multi-optimization improves genetic programming generalization ability. In *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, GECCO '07, pages 1759–1759, New York, NY, USA, 2007. ACM. 183

[110] Leonardo Vanneschi and Sara Silva. Using operator equalisation for prediction of drug toxicity with genetic programming. In *EPIA*, pages 65–76, 2009. 8, 84, 94, 101

[111] Ekaterina J. Vladislavleva, Guido F. Smits, and Dick den Hertog. Order of nonlinearity as a complexity measure for models generated by symbolic regression via pareto genetic programming. *IEEE Transactions on Evolutionary Computation*, 13(2):333–349, April 2009. 6, 50, 51, 82, 102, 104, 128, 183, 184

[112] Darrell Whitley. The genitor algorithm and selection pressure: Why rank-based allocation of reproductive trials is best. In *Proceedings of the Third International Conference on Genetic Algorithms*, pages 116–121. Morgan Kaufmann, 1989. 30

[113] Stephan M. Winkler, Michael Affenzeller, and Stefan Wagner. Using enhanced genetic programming techniques for evolving classifiers in the context of medical diagnosis. *Genetic Programming and Evolvable Machines*, 10(2):111–140, June 2009. 42

[114] David S. Wishart, Craig Knox, An Chi Guo, Savita Shrivastava, Murtaza Hassanali, Paul Stothard, Zhan Chang, and Jennifer Woolsey. Drugbank: a comprehensive resource for in silico drug discovery and exploration. *Nucleic Acids Research*, 34:D668–D672, 2006. 90

[115] D.H. Wolpert and W.G. Macready. No free lunch theorems for optimization. *Evolutionary Computation, IEEE Transactions on*, 1(1):67 –82, apr 1997. 15

[116] Bart Wyns, Peter De Bruyne, and Luc Boullart. Characterizing diversity in genetic programming. In Pierre Collet, Marco Tomassini, Marc Ebner, Steven Gustafson, and Anikó Ekárt, editors, *Proceedings of the 9th European Conference on Genetic Programming*, volume 3905 of *Lecture Notes in Computer Science*, pages 250–259, Budapest, Hungary, 10 - 12 April 2006. Springer. 10, 137

[117] Fumitaka Yoshida and John G. Topliss. Qsar model for drug human oral bioavailability. *Journal of Medical Chemistry*, 43:2575–2585, 2000. 90

[118] Byoung-Tak Zhang and Heinz Mühlenbein. Balancing accuracy and parsimony in genetic programming. *Evol. Comput.*, 3:17–38, March 1995. 40