SEMANTIC CROSSOVER OPERATORS BASED ON THE FIRST AND SECOND PARTIAL DERIVATIVES OF THE FITNESS FUNCTION FOR GENETIC PROGRAMMING

THESIS

A thesis submitted for the degree of
PHD. IN ELECTRICAL ENGINEERING

presents
M.C. Ranyart Rodrigo Suárez Ponce de León

Dr. Mario Graff Guerrero
Thesis Advisor

Dr. Juan José Flores Romero
Thesis Co-Advisor

Universidad Michoacana de San Nicolás de Hidalgo
Facultad De Ingeniería Eléctrica
División de Estudios de Posgrado

Morelia, Michoacán, México
Enero 2018

Contents

Nomenclature

AGX	Approximately Geometric Crossover
ANNs	Artificial Neural Networks
AR	Auto-Regressive Model
ARIMA	Auto-Regressive Integrated Moving Average
ARMA	Auto-Regressive Moving Average
BDTs	Binary Decision Trees
BER	Balanced Error Rate
DE	Differential Evolution
DTs	Decision Trees
EAs	Evolutionary Algorithms
ES	Evolution Strategies
GAs	Genetic Algorithms
GE	Gene Expression Programming
GP	Genetic Programming
GPPDE	Semantic Crossover with first partial derivative information
GPPDE2	Semantic Crossover with second partial derivative information
GPPDEC	Semantic Crossover combining GPPDE and GPPDE2

GSGP	Geometric Semantic Genetic Programming
KLX	Krawiec and Lichocki Geometric Crossover
LARS	Least Angle Regression
LASSO	Least Absolute Shrinkage and Selection Operator
LGX	Locally Geometric Crossover
RBF	Radial Basis Function Kernel for SVMs
RF	Random Forests
SVMs	Support Vector Machines

List of Symbols

α, β	Values propagated by the Backpropagation Algorithm
$\frac{\partial E}{\partial v}$	First partial derivative of E w.r.t v
$\frac{\partial^2 E}{\partial v^2}$	Second partial derivative of E w.r.t v
E	Error Function $E = (y - \hat{y})^2$
e	Difference between target and output $e = (y - \hat{y})$
u	Cross point in the second parent
v	Cross point in the first parent

List of Figures

List of Tables

Abstract

Semantic genetic operators in evolutionary algorithms, particularly in GP, have been proved to be a better heuristic than traditional syntactic operators by improving the performance of GP over certain problems. A semantic operator, crossover or mutation, is guided by the behavior (semantics) of the individual, rather than its syntax, in order to create new and fitter individuals. In this work, we propose to use the information provided by the first and second partial derivatives of the error function (fitness function) *w.r.t.* the crossing point, to develop novel semantic crossover operators for GP that outperform the traditional crossover.

A semantic crossover operator based on partial derivatives will guide the crossover operation by the minimization of the error contribution of the subtree rooted at the crossing point. In order to develop these new operators, a new methodology is proposed to compute the first and second partial derivatives of the fitness function with respect to the crossing point, this methodology is inspired in the backpropagation algorithm used to train ANNs. The information provided by the partial derivatives will guide a search in the second parent in order to find more suitable values for the subtree rooted at the crossing point.

The semantic operators presented in this work are tested in three different kind of problems commonly solved with GP: Regression, Classification and Feature Selection. Besides comparing the semantic crossover operators against the traditional syntactical crossover other semantic methods are also included in the comparison, for Regression problems. For Classification and Feature Selection the comparison is with other traditional techniques used for such tasks. The obtained results show a major improvement from traditional crossover for GP in the problems tested and are competitive against other semantic operators and traditional techniques.

Resumen

Los operadores genéticos semánticos en los algoritmos evolutivos, especialmente en PG, han demostrado ser una heurística con mejores resultados que los operadores genéticos sintácticos tradicionales debido a que aumentan el desempeño de PG. Un operador semántico utiliza la información proporcionada por el comportamiento (semántica) del individuo, de tal forma que su descendencia esté mejor adaptada que el individuo original. Este trabajo propone utilizar la información proporcionada por la primera y segunda derivada parcial de la función de error con respecto al punto de cruza seleccionado, con la finalidad de desarrollar diferentes operadores semánticos para PG.

Un operador semántico de cruza basado en las derivadas parciales guiará el proceso de cruza de manera que minimise el error que proporciona el punto de cruza. Para poder desarrollar estos nuevos operadores semánticos de cruza, se ha propuesto una nueva metodología para poder calcular las derivadas parciales de la función de error con respecto al punto de cruza, dicha metodología está inspirada en el algoritmo de propagación hacia atrás que es usado para entrenar Redes Neuronales. La información que proporcionan las derivadas parciales es usada para realizar una búsqueda en el segundo padre para encontrar valores más apropiados para el subárbol cuya raíz es el punto de cruza.

Los operadores semánticos presentados en este trabajo fueron probados en tres diferentes clases de problemas: Regresión, Clasificación y Selección de características. Además de comparar los operadores semánticos de cruza contra la cruza sintáctica tradicional también otros operadores semánticos se han incluido en la comparativa en los problemas de Regresión. Para Clasificasión y Selección de características la comparación fue hecha contra técnicas tradicionales usadas para este tipo de problemas. Los resultados obtenidos muestran una mejora considerable respecto a la cruza tradicional de PG en los problemas usados mientras que son competitivos contra otras técnicas semánticas y técnicas tradicionales.

KeyWords— Programación Genética, Operadores Semánticos, Regresión Simbólica, Clasificación, Selección de Características

Chapter 1

Introduction

Genetic Programming (GP) ([Koza92]), is an evolutionary algorithm where the individuals represent computer programs that are designed to solve a particular task. *Evolutionary Algorithms (EAs)* are optimization algorithms based on the concepts of the theory of *Evolution* developed by Charles Darwin. Other examples of EAs are Genetic Algorithms (GAs), Differential Evolution (DE), Evolution Strategies (ES), Gene Expression Programming (GE), etc. These techniques are alternatives to traditional optimization algorithms due to some characteristics, for example the ability to self-adapt the search for optimum solutions on the fly [Fogel97].

GP is an optimization algorithm that solves problems typically represented in a *Supervised Learning* approach; this means that GP requires a *target* which will be guiding the optimization process. Different metrics are used (traditionally distance metrics) to measure how similar the output of the individuals' (computer programs) and the target are. The goal for GP is to create individuals that recreate the target as closely as possible. This process is called the *training* stage and the target is called the *training set*. In other words, GP creates computer programs that model the training set, those models are represented by the best individual at the end of the GP run. Besides the training set a *test set* is required to measure how well the individuals generalize to new data. The test set is not used by GP in the optimization process.

Regarding the construction of the individuals in GP, these are created by selecting elements from two sets: the function set and the terminal set. The function set contains all the functions that can be part of the computer program and the terminal set contains constants, inputs, or functions without arguments. One of the most popular representation

of an individual in GP is as a tree data structure, where leafs are elements of the terminal set, and inner nodes are elements of the function set. These trees are $n - ary$ trees, where n is given by the arity of the functions used in the individuals. The evaluation of the root of the tree is the output of the individual.

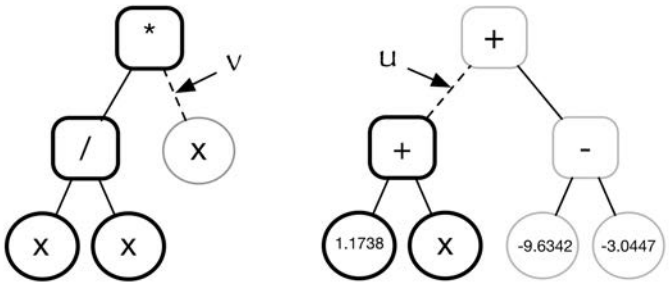
EAs create an initial population of individuals and then apply *evolutionary operators* to resemble the way natural selection acts in nature and promotes the survival of the fitter individuals. The evolutionary operators used in GP are *crossover* and mutation. The crossover operator takes two individuals (these individuals are called parents) and produce offspring which contain certain parts of both parents. On the other hand, mutation acts selecting one individual and changing one part of its genetic information resulting in a new mutated individual.

If individuals in GP are represented as trees, the crossover operator randomly selects one node in each parent; those points will be called *crossing points*. Each crossing point can be seen as the root of a sub-tree in the parent. Traditional crossover swaps the subtrees genereted by the crossing points between the parents. Performing this kind of crossover gives two new individuals: the first parent with a sub-tree generated by the crossint point in the second parent and, analogously, the second parent with a sub-tree generated by the crossing point in the first parent. Although two individuals are created by the crossover, only one will be added to the population, the other individual is discarded. Figure 1.1 depicts a crossover where v and u are the crossing points in both parents, it can be observed that the offspring is generated by replacing x (light gray), from the first father, with sub-tree rooted below u of the second parent, resulting in Figure 1.1 (c)

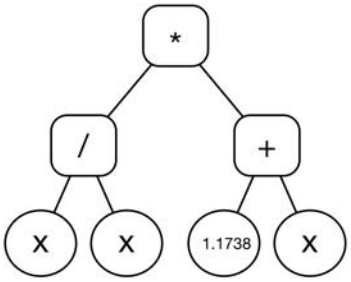
1.1. Motivations

Traditionally, the crossover operator computes the crossing points randomly in both of the parents. Because of this, there is no guarantee that the resulting offspring will be fitter, according to the metric being used, than the parents. Furthermore, the two individuals generated by the crossover are not compared under the metric in order to discard the less fit of them, this process is also random (in some GP systems the crossover is fixed, meaning that the first child is always discarded or vice versa).

The crossover operator is said to be a *syntactical* operator because it acts on the syntax of the individuals. The syntax of an individual corresponds to its tree representation.



(a) Parent 1 (b) Parent 2



(c) Offspring

Figure 1.1: Random individuals selected for crossover and their respective offspring (given the crossing points v and u)

The crossover operator takes the syntax of the parents and performs a swap of sub-trees in order to produce offspring. However, this strategy does not take into account the fitness of the parents to produce the offspring. Syntactic crossover operator works with any information of the behavior of the individuals (semantics). The crossover acts over the syntax of the individuals and produce offspring that hopefully, will be better adapted than its parents. It seems to be a much better strategy for the crossover to look at the behavior of the individuals rather than their syntax, in order to produce offspring, crossing individuals which have *desired* behaviors over individuals whose behavior is not suitable for the problem. With this last idea in mind, it is straightforward to ask whether there are better procedures to select the crossover points in the parents, looking at their semantics, instead of computing them randomly and acting merely on the syntax.

This last idea is the main motivation on which we have focused our attention to develop the present work: propose a different way to perform the crossover in GP based on the semantics of the individuals. The semantics of an individual in GP is commonly unders-

tood as the output of the individual. In other words, semantics refers to the behavior of the individual. The metric used to evaluate the fitness of the individuals is called the *fitness function* and this function takes as an argument the output of the individual (semantics) rather than its syntax, to evaluate its fitness.

1.2. Objectives

The main objective of this work is to develop novel semantic crossover operators that outperform the traditional syntatic crossover for GP. The crossover operators developed in this work will have to accomplish two goals. The first goal is that the crossover has to semantically guide the crossover by the behavior of the parents rather than their syntax and, in this way, performing more *conscious* (non-blind) crosses between individuals than the traditional syntactic crossover operator. The second goal is simple: the performance of these crossovers have to increase compared to the syntactical one.

In order to accomplish the objectives, the crossover operators presented in this work, computes the partial derivative of the fitness function *w.r.t.* one node of the tree (the crossing point); with that information, crossover decides the crossing point in the second parent in order to swap the sub-tree that minimizes the fitness function according to the derivative. This operator uses the information of the semantics of the individuals in order to perform the crossover between parents, therefore this operator falls in the category of *Semantic* operators in GP. Chapter 2 addresses the topic of semantic operators.

1.3. Achievements

Partial Derivatives in GP

In order to develop the semantic crossover operators proposed in this work, a methodology to compute partial derivatives in GP had to be created. This methodology is inspired on the backpropagation algorithm used to train ANNs. The core ideas of the backpropagation algorithm were used to compute the first and second partial derivatives of the error function *w.r.t.* a certain node in an individual of GP.

Semantic Crossover Operator for GP

This work presents three different crossover operators for GP. The difference between them is the degree of the partial derivative being used. Results show a substantial increase in performance with respect to GP's traditional crossover.

Classification, Regression and Feature Selection

GP is often used in Classification an Regression problems. The semantic crossover presented in this work were tested under these problems and outperform the traditional crossover, opening the possibility to apply them in those tasks to real-world problems.

1.4. List of publications

- [Suárez15] Suárez, Ranyart Rodrigo, Mario Graff, and Juan J. Flores. "Semantic Crossover Operator for GP based on the Second Partial Derivative of the Error Function." Research in Computing Science 94 (2015): 87-96.

- [Suárez14] Suárez, Ranyart Rodrigo, José María Valencia-Ramírez, and Mario Graff. "Genetic programming as a feature selection algorithm." Power, Electronics and Computing (ROPEC), 2014 IEEE International Autumn Meeting on. IEEE, 2014.

- [Valencia-Ramirez14] Valencia-Ramirez,J.M.,Raya,J.A., Cedeno,J.R., Suárez, Ranyart Rodrigo, Es- calante, H. J., y Graff, M "Comparison between Genetic Programming and full model selection on classification problems." Power, Electronics and Computing (ROPEC), 2014 IEEE International Autumn Meeting on. IEEE, 2014.

- [Tellez17] Tellez, E. S., Miranda-Jiménez, S., Graff, M., Moctezuma, D., Suárez, Ranyart Rodrigo, y Siordia, O. S "A simple approach to multilingual polarity classification in Twitter", Pattern Recognition Letters, 2017.

1.5. Thesis Outline

Chapter 2 presents the related work. The chapter starts with the classification of the different semantic-aware approaches in GP, not necessarily crossover methods but a more general view of the different techniques that try to initialize the population of individuals

in order to have semantic variety among them. Naturally, the chapter also covers a brief review of the semantic crossover operators that can be distinguished in the literature and that are going to be compared to our proposal later in this thesis.

Chapter 3 presents the core ideas behind our semantic crossover operators: develop a methodology than can compute partial derivatives in GP and, using this information, develop semantic crossover operators for GP. The chapter explains all the grinds that make possible to compute first and second partial derivatives in GP as well as the modifications required by GP in order for this methodology to work. Three different crossover operators are presented in the chapter; the difference between them is the order of the derivative being used as well as the heuristic that uses the information provided by the derivative.

In Chapter 4 our semantic crossover operators are tested over a set of benchmark problems that consists of *Symbolic Regression* problems. The definition of Symbolic Regression is provided in the chapter. We decided to test our operators in Symbolic Regression due to two main reasons: Symbolic Regression is perhaps, the most common problem where the advantages of GP are used and the other reason is that there exists a couple of different semantic crossovers found in the literature that test their performance over the same set of problems. Due to this we can compare the performance of our operators to the aforementioned.

Chapter 5 presents a different set of tests applied to crossover operators. This time our operators are tested in *Classification* problems. Classification is one of the most common problem that arises in the area of Artificial Intelligence and hence the importance of the developing of better heuristics to solve it. This time, to the best of our knowledge, we could not find semantic crossover operators that solves benchmark problems in Classification, therefore we compare the performance of our approaches v.s. traditional techniques used to solve the task.

Chapter 6 presents the last series of tests applied to our semantic crossover operators. Due to its conceptualization, GP selects different variables of the problem being solved to build the individuals that will be part of the population. In problems with high dimensionality (large number of variables), GP selects a subset of such variables to solve the task. This is, that at the end of the run, GP acts as a filter of redundant variables or outliers. The problem of filtering this kind of non useful information is called *Feature Selection*, and different problems of this category are used to test our operators. Again, different traditional techniques that are commonly used to solve the feature selection problem were

used for comparison purposes.

Chapter 7 presents the final thoughts of this work as well as some future ideas that can be implemented in short time. The main idea of this chapter is to discuss the achievements obtained by our semantic crossovers and how the main objectives proposed in this work have been accomplished.

Chapter 2

Literature Survey

Semantic awareness in GP refers to genetic operators (or strategies) that use the information of semantics of the individuals, instead of their syntax, to perform the evolutionary operators like crossover and mutation, and even, to initialize the population. Perhaps, one of the first attempts to include semantic awareness into GP was made by [Blickle94], which proposed to select as crossover points only those nodes that have an impact on the fitness function, avoiding those crossover points that did not provide offspring with different semantics. To accomplish this, Brickle proposed the concept of *redundancy*, which refers to the loss of valuable genetic material of individuals leading GP to get trapped in local optima. Under this concept of redundancy in GP, the crossover proposed by Brickle, selects as crossing points, only the nodes in parents which are not redundant.

Since Brickle's work, many researchers have focus their attention in develop semantic awareness strategies for GP. The majority of these works include the notion of semantics into the crossover operator. According to Vanneschi [Vanneschi14], three major categories of semantic awareness in GP can be distinguished in the literature: diversity methods, direct methods and indirect methods

2.1. Diversity, Direct and Indirect Methods

Following the classification of *semantic-aware* strategies for GP done by Vanneschi, diversity methods refer to strategies that promote semantic diversity among the individuals in order to avoid local optima or in other words, promote a more broad search. The idea is to preserve different genetic information among the population leading to individuals that

exhibit different semantics as possible. These kind of strategies are mostly implemented at population level like population initialization such as the one proposed by [Beadle09].

On the other hand, direct methods work on the semantic space by defining semantic operations that translate into syntaxis transformations ([Moraglio12a, Vanneschi14]). Working on the semantic space has the benefit of knowing some properties of all the offspring generated, for example, Moraglio's operator [Moraglio12a] has the characteristic that the offspring cannot be worse (in terms of fitness) than the worse of the parents. Similarly, Graff [Graff15] developed an operator whose offspring is at least as fit as the best of the parents. Direct methods are also called *geometric* operators (or semi-geometric), these operators are called geometric because the crossover operator, under a metric $d : \mathbb{R}$ two parents $p1$ and $p2$ produce offspring that lie in the d-segment between the parents. Other examples of directs methods are Krawiec's and Pawlak's operators [Krawiec09, Moraglio12a, Pawlak14].

Finally, indirect methods, such as our proposal, act on the syntax of the individuals, and then, certain predefined semantic conditions are checked. This category is where all the majority of semantic operators in GP falls into. This class of operators is more general and is simply called semantic operators because the operations to generate new individuals are driven by certain rules or conditions constrained to the semantics of the generated individuals. For example, Beadley [Beadle08, Beadle09] proposed a crossover operator that is semantically driven in the sense that it promotes individuals which are semantically different from their parents (predefining a way to measure this semantic difference). The offspring are added to the next generation only if they are not semantically equivalent to the parents (forcing to have different semantics among the population).

Following Beadley's idea and extending it to other domains, Nguyen [Nguyen09] and Uy [Uy10, Uy11] proposed crossovers that also produced offspring semantically different from their parents although the way this difference is measured differs from Beadle's. Ruberto [Ruberto14] introduced the concept of error vector and error space. Briefly, the idea is that the optimal solution can be constructed from two individuals that are aligned in the vector space. Consequently, the objective of the search procedure can be changed from finding the closest individual to the desired behavior (i.e., the origin in the error space) to finding aligned individuals. This novel GP strategy has shown success in solving two complex real-life applications in drug discovery, namely, human oral availability and median lethal dose.

The crossover operators developed in this work, rely on the Backpropagation al-

gorithm, specifically on the graphic representation described in [Rojas96]. This algorithm has been widely used for years to train Artificial Neural Networks (ANNs). Particularly, the backpropation algorithm is used to adjust the weights of the connections that exist between neurons. The usage of the backpropagation in GP is not new, the algorithm has been used in the past to optimize the constants of trees (individuals in GP). Its application to GP is somewhat straightforward because GP and ANNs share some characteristics. In GP, the individuals are represented as trees and the output of the nodes are fed to other nodes and so on. Similarly, the neurons in ANNs are processing units called *perceptrons* (which can be seen as functions with multiple arguments) that receive as inputs the output of other neurons. Examples of the usage of backpropagation algorithm in GP are [Smart04, Zhang04a, Graff13], where the backpropagation algorithm is used to update the values of the constants, very similar to what is done when optimizing the weights in an ANN. However, these ideas applied to optimize the constants in individuals cannot be directly used to select the crossover points given that the crossover points contain functions (root of sub-trees) instead of constants.

The work that presents the most similarities with the crossover operators presented in this works is from Pawlak's research [Pawlak14]. Roughly, the idea presented by Pawlak is to pass the desired semantics of subtasks (subtrees) during crossover, using the backpropagation algorithm. The desired semantics for these subtasks (which are simpler than the main task, i.e., reproducing the target or training data), are previously computed with reverse computation. Our proposal differs from Pawlak's given that the information passed by the backpropagation algorithm is the error contribution of each node, provided by the partial derivatives of the error function contrarily to Pawlak's where the information passed by the backpropagation algorithm are several previously computed values.

This work is based in a previous semantic crossover operator presented by Graff *et.al.* [Graff14]. The main idea of this proposal is to compute the partial derivative of the fitness function with respect to the node selected as the cross point in the first parent and with this information chose the second cross point in the other parent. The selection of the cross point in the second parent is done checking which sub-tree in the individual has the most suitable output according to the derivative propagated for the first parent. This can be accomplished by adequating the backpropagation algorithm in GP. Particularly, the backpropagation is used to propagate the derivative of the error function to the desired node (the crossing point for the first parent).

The novel semantic operators presented in this work take the idea presented by Graff *et.al.* and goes one step further by computing the second partial derivative of the error function. The second order derivative provides new information that is used to develop novel heuristics about the selection of the crossing point in the second parent, and outperform the traditional syntactical crossover operator for GP. The next chapter presents the methodology for computing the partial derivatives of the error function in GP and how to use it to develop the semantic crossover operators.

Most of the times, GP is used in Regression and Classification problems. However, due to the way that GP produces new individuals, it have also been used in problems of Feature Selection, much lesser times than Regression and Classification though. Due to the fact that Regression, Classification and Feature Selection are the most common problems solved with GP, we decided to test our approach on these problems. In the next subsection, a brief review of the use of GP over these problems is presented.

2.2. GP and Regression

Regression refers to a problem when output data of a certain model or system is known but the model itself is not. The user must select one known model to fit the data and then tune its parameters, this is the case of Auto Regressive (AR) models and its variants ARMA and ARIMA, for example. However, in Symbolic regression the user does not suppose anything about the model that generated the data and infers both the model and its parameters.

GP suits perfectly in the case of Symbolic Regression, GP itself does not suppose anything about the data and the fitness of the individuals is driven by how well their ouput fits the data under certain metric. The individuals generated by GP can be seen as different regression models and their paramaters are given by the different combinations of the function and terminal sets. For example, in [Augusto00] GP is used to fit different sampled polynomial functions, known a-priori, and results show that GP can infer the exact polynomial function in some cases.

The Semantic approaches in GP addressed in the 2.1 Subsection are all tested in some type of symbolic regression problem (boolean or real valued), so, the references provided in that subsection are examples of the application of GP to Regression problems.

2.3. GP and Classification

According to Espejo *et al.* [Espejo10], three major categories of GP into Classification can be distinguished: Preprocesing, Model Extraction and Ensemble of Classifiers. The preprocessing of the data in Classification refers to transformations of the original data, these transformations are often aimed at reducing noise in the data. Model extraction in Classification is the task of inducing a classifier from the data, it is the main task in Classification. An Ensemble in the Classification context, is the union of two of more classifiers that were constructed over the same data in order to obtain a larger and more *robust* (under certain criteria) classifier.

The obvious approach when using GP to infer a classifier is to treat each individual as a classifier and guide the evolution with a fitness function that evaluates the quality of each classifier (individual). Most of the works found in the literature that use GP for Classification rely on this approach, therefore there is a vast number of works done in the past over different areas. In this large number of publications of GP and model extraction, Espejo *et al.* [Espejo10] suggest certain divisions in order to have a clearer vision: GP for extracting decision trees, GP for learning rule-based systems and GP for learning discriminant functions. Next are some uses of GP for model extraction in Classification.

GP for extracting Decision Trees

Decision Tress (DTs) [Quinlan86] are trees designed to perform simple comparisons between data features and infer a classifier. The branches of the trees represent the diferent outcomes of the comparisons and similarly, perform other comparisons. If a DT performs only one comparison per node, with two possible outcomes it is called a Binary Decision Tree (BDT).

GP evolves tree structures and therefore the use of GP to evolve BDTs have been in the mind of researches of many areas: *Software Engineering* [Khoshgoftaar07], *Pattern Recognition* [Shirasaka98, Tanigawa00, Haruyama02, Oka00], *Benchnarks* [Folino99, Folino00, Eggermont02, Bot00b, Bot00a], *Medicine* [Estrada-Gil07, Mugambi04], *Finance* [Kuo07].

However, the application of GP to evolve BDTs has its limitations. BDTs are very simple tree structures and due to this, BDT's size is usually very short in comparison to the size of the trees evolved by GP. Therefore, using GP to evolve BDTs will result in trees much larger than BDTs inferred traditionally.

To overcome this issue, researchers have adopted some strategies the most common and obvious being two approaches: choosing for crossover those trees whose size is smaller given equal fitness and directly include the size of the individuals into the fitness function. Other strategies split the training data into n smaller sets and then use different GP for each of these sets leading to small BDTs specialized to classify a sub-set of the training set.

GP for learning rule-based systems

A rule-based system consists of a set of rules which classifies data. These rules are in the form of *IF(antecedent P)... THEN(consequent Q)*. The rules test the features of the data to satisfy a condition and then assigns a class given the result of the test. Many techniques have been used to infer the set of rules, being EAs among them [Casillas09, Carse07] and also GP.

Traditionally, when GP is used to infer the rules for a rule-based system two approaches are applied: the *chromosome = set of rules* or Pittsburgh approach [Smith80] and the *chromosome = rule* or Mighigan approach [Wilson95]. In the Pittsburgh approach the best individual of the run codifies the solution for the classification problem. The main drawback of this approach is that the solution is large and complex. In the Michigan approach the opposite occurs, the solutions are simpler but the drawback is that several runs of GP are required in order to infer the whole set of rules.

Wether Michigan or Pittsburgh approach are used to infer the rule set, reasearches of many areas have use GP to build such set, for example: *Pattern Recognition* [Stanhope98, De Stefano02], *Benchmarks* [Tsakonas06, De Falco02, Tan02, Carreno07, Eggermont99, Espejo05], *Finance* [Qing-Shan07, Sakprasat07, Garcia-Almanza08], *Medicine* [Yu07, Tan03, Bojarczuk00, Ngan99], *Biology* [Johnson00, Wang05], etc.

GP for learning discriminant functions

A discriminant function or in the case of Classification, a classification function, takes data's features as arguments (or sub-features created from original features) and maps or assigns a certain class for that features. Discriminant functions are concise and more efficient than classification rules but are hard to be understood and iterpreted since the discriminant functions are usually non-linear [Chien02]. A whole set of discriminant functions is required to solve a classification problem for k classes: $F = \{f_i | f_i : \mathbb{R}^n \rightarrow \mathbb{R}, 1 \leq$

$i \leq k\}$.

GP can be used to find such non-linear discriminant functions as we can se in works from many areas where a Classification problem occurs: *Benchmark* [Patterson07, Curry07, Li08, Cavaretta99, Muni04], *Pattern Recognition* [Tackett93, Teredesai04], *Image Classification* [Wijesinghe07, Petrović05, Chen07, Li07], *Signal Classification* [Teller95], *Communications* [Zhang08, Faraoun06, Mukkamala04], *Engineering* [Sette04, Zhang07, Hennessy05].

GP ensemble for Classification

An ensemble of classifiers refers to the use or combination of different classifiers (or instances of the same algorithm with different parameters' values) in the same classification problem. This creates a larger classifier that consists of all the classifiers being used. The idea behind this technique relies on the fact that different algorithms have different advantages and drawbacks. Hopefully, the ensemble of classifiers will benefit of the goods of each individual classifier in the ensemble and on the other hand, dismiss their particular drawbacks.

In order to combine all the algorithms included in the ensemble a method for interpreting the results must be defined. This usually is accomplished by a votation scheme, where all the algorithms are run in the classifiaction problem and then the predictions of the algorithms are compared to each other and the class with more number of votes, for a particular example is chosen as the final predicted class. In other ensemble approaches, this votation scheme use weighted votes in order to give more importance to some algorithms than others.

Naturally, GP have been included as part of ensemble methods in the past: *Benchmark* [Folino08, Thomason07], *Communications* [Zhang04b], *Medicine* [Hong06, Hengpraprohm08, Brameier01], *Biology* [Xu08, Imamura03].

2.4. GP and Feature Selection

In a nutshell, the problem of Feature Selection is basically the reduction of data dimensionality (Feature Selection will be explained in Chapter 6). This reduction of data's dimensionality is often mandatory in order to separate noise or redundant information that is not useful to the algorithm being used to solve the Classification problem. The use of GP for Feature Selection applied to Classification is mainly aimed at this preprocessing of

data, in other words, GP is used to reduce the dimensionality of data and then, with this new transformed data, a classifier is applied over the new data.

GP is used to reduce the number of data's dimensions for one reason: when data dimensionality is high it is unlikely for the best individual of the run to have all the features contained in the data. So, if the best individual skip some of the features to classify the data such features are probably noise (or not relevant to the problem) and can be filtered from the data. There are several examples of the use of GP to preprocess data related to classification problems: *Engineering* [Guo05], *Pattern Recognition* [Lin05], *Benchmark* [Neshatian08, Krawiec02, Smith05, Sherrah96, Sherrah97], *Medicine* [Guo06, Estébanez05], *Finance* [Estébanez08].

2.5. Summary

In this chapter, the problems inherent in the traditional crossover in GP have been addressed. Summarizing, these problems are mainly due to the fact that the traditional crossover is merely an exchange of syntactical information with any guarantee that the offspring will be better adapted. Although the traditional crossover also exchanges semantic information, is only a side effect of the crossover because the crossover is not guided by this information.

A brief review of semantic crossover operators reported in the literature has been presented. These operators have shown an increase in performance with respect to traditional crossover in GP. Semantic crossovers operators have been proposed for GP with the goal of producing fitter offspring. Although all the semantic crossover share this goal, the way that this goal is achieved differs among them. Three major categories can be distinguished: diversity methods, direct methods and indirect methods. In the next chapter, our proposal of semantic crossover operators is going to be presented, This contribution fits in the category of indirect methods.

The uses of GP addresed in this work correspond to three problems: Symbolic Regression, Classification and Feature Selection. Therefore, a brief review of several works done in these areas have been presented. The cited works show that researchers of several and different fields, use GP to solve the different problems that rise in their respective areas. This verifies the idea that GP can be used in a very wide range of areas.

Chapter 3

Partial Derivatives in Genetic Programming

The semantic crossovers operators proposed in this contribution produce an offspring by replacing the subtree rooted at a certain node from the first parent with the subtree rooted at the second crossing point (u) from the second parent. The goal is to minimize the offspring's fitness (e.g., distance between target behavior and offspring behavior) by carefully selecting the crossing point. The procedure used to minimize the offspring's fitness is as follows. First, it measures the error contribution associated to the subtree rooted at the crossing point. Second, having obtained the error contribution, a subtree with root at the second crossing point (i.e., u) is selected, from the second parent, such that it might reduce this error. This section describes the methodology used to tackle the first task of the semantic crossover procedure, and Section 3.3 describes the heuristics developed to select the second crossing point i.e., u.

In Artificial Neural Networks (ANN), the problem of measuring the error contribution of a particular node in a layer, specifically a node associated with a constant, has been known, for many years, using the Backpropagation algorithm. Backpropagation is used to optimize the weights in the neural network such that the net's output is as similar as possible to the desired output. Backpropagation is an iterative process, in each step or iteration the weights are updated using a gradient descent scheme; that is, using the information provided by the gradient of the error *w.r.t.* that weight, i.e., $\frac{\partial E}{\partial w}$, where E is the error function (in GP it is the fitness function) and w is the weight being optimized.

3.1. First Partial Derivative

Traditionally, Backpropagation is explained and implemented using matrix notation, which is very convenient given the ANN's structure; conversely, Raul Rojas in [Rojas96] explained it using graph operations. Specifically, the procedure used to compute $\frac{\partial E}{\partial w}$ was performed by storing information on the vertices in the forward step, and applying arithmetic operations to this information, and feeding the values to the network in the backward step of the algorithm. In this contribution a tree-based GP system is being used, consequently, the procedure proposed by Rojas can be implemented in a tree structure, and, one can obtain $\frac{\partial E}{\partial w}$, where w is any node of the tree. In order to have a clear picture of how to compute $\frac{\partial E}{\partial w}$, let us start explaining the procedure described by Rojas with a simple example.

Let us assume that one is interested in $\frac{\partial g(f(x))}{\partial x}$, so the first step would be to depict function $g(f(x))$ in a graph. Figure 3.1 represents $g(f(x))$ where the arrows indicate the computing path, that is, input x is fed to node f which produces $f(x)$; then, $f(x)$ is given to node g which in turn produces $g(f(x))$. This procedure is part of the forward step of Backpropagation; however, in order to compute $\frac{\partial g(f(x))}{\partial x}$, we need to store values at each node in order to use them in the backward step of the algorithm.

Figure 3.1: Function $g(f(x))$ represented as a directed graph.

Figure 3.2 presents the graph representing $g(f(x))$; it can be observed that each node is split in two parts: the lower part stores the value of computing the partial derivative of the node's function with respect to the input variable, and the upper part defines the function being applied to the input. From the figure, it is observed that node f stores $f'(x)$, and, respectively, node g stores $g'(f(x))$; please note that $f(x)$ is the input of node g, so it is straightforward to compute $g'(f(x))$. For example, let $f = \sin$ and $g = \cos$, then $cos(x)$ would be the first value stored and $-\sin(\sin(x))$ would be the second value stored.

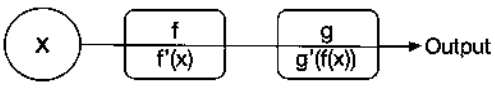


Figure 3.2: Stored functions for computing the first derivative.

Figure 3.2 depicts the forward step of the Backpropagation algorithm; on the other hand, the backward step is performed by traversing the graph from the output to the inputs, this can be observed from Figure 3.3. Note the change in the direction of the edges in order to show the path of the computation. The procedure to compute $\frac{\partial g(f(x))}{\partial x}$ can be seen as a chain of multiplications; 1 is submitted to the net at the output (right hand side of the figure). This is multiplied by the value stored at node g, the results is $g'(f(x))$, this result is then multiplied by the value stored at node f, yielding $g'(f(x)) \cdot f'(x)$, which is in fact $\frac{\partial g(f(x))}{\partial x}$. Following the previous example, let us compute $\frac{\partial \cos(\sin(x))}{\partial x}$, as mentioned above, the second value stored is $-\sin(\sin(x))$ and the first value stored is $\cos(x)$ hence $\frac{\partial \cos(\sin(x))}{\partial x} = -\sin(\sin(x))\cos(x)$.

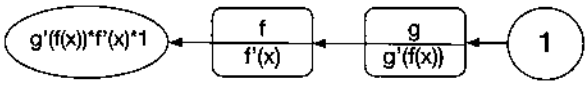
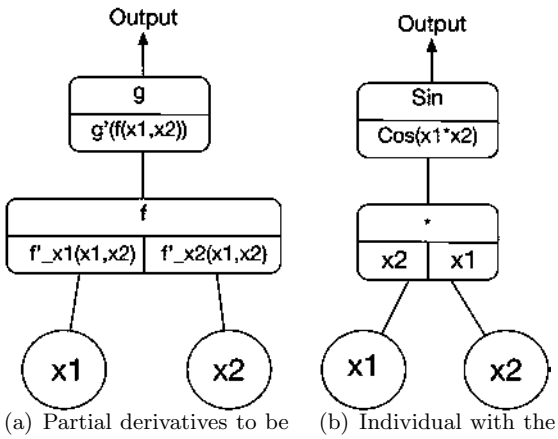


Figure 3.3: Back step for computing the first derivative.

Let us go beyond computing the derivative of a function with only one argument. Figure 3.4 presents the following function $g(f(x_1, x_2))$. From the figure, it can be observed that the lower part of node f is split in two, this is to store the partial derivatives with respect to each argument. In the forward step x_1 and x_2 are the inputs to node f, at this node we need to store $\frac{\partial}{\partial x_1} f(x_1, x_2)$ and $\frac{\partial}{\partial x_2} f(x_1, x_2)$ which are depicted at the lower left and right of node f, respectively. The next step $f(x_1, x_2)$ is the argument of node g and the output is $g(f(x_1, x_2))$. For example, let $f = *$ and $g = \sin$, then $\frac{\partial}{\partial x_1} f(x_1, x_2) = x_2$ and $\frac{\partial}{\partial x_2} f(x_1, x_2) = x_1$, that is x_2 and x_1 are stored at the left and right part of node f (see Figure 3.4 (b)). Node g stores $\cos(x_1 x_2)$, which corresponds to $\frac{\partial}{\partial x}\sin(x) = \cos(x)$ where x is the input, i.e., $x_1 \cdot x_2$.

The backward step is performed as follows: one is submitted to the output, then this is multiplied by $\frac{\partial}{\partial x} g(f(x_1, x_2))$, which is then passed to node f. At this point, there are two paths, following the left path to x_1 it is obtained $\frac{\partial}{\partial x} g(f(x_1, x_2)) \frac{\partial}{\partial x_1} f(x_1, x_2)$ and on the right path to x_2 it is obtained $\frac{\partial}{\partial x} g(f(x_1, x_2)) \frac{\partial}{\partial x_2} f(x_1, x_2)$, which corresponds to $\frac{\partial}{\partial x_1} g(f(x_1, x_2))$ and $\frac{\partial}{\partial x_2} g(f(x_1, x_2)$, respectively. Following the previous example, path to x_1 obtains in the backward step $\cos(x_1 x_2)x_2$ and path to x_2 obtains $\cos(x_1 x_2)x_1$, which correspond to $\frac{\partial}{\partial x_1}\sin(x_1 x_2)$ and $\frac{\partial}{\partial x_2}\sin(x_1 x_2)$, respectively.



(a) Partial derivatives to be stored

(b) Individual with the derivatives stored

Figure 3.4: Representation of partial derivatives stored in an individual.

3.2. Second Partial Derivative

One of the goals of this contribution is the use of the information provided by the second partial derivative to select the crossover point u of the second parent. In order to compute the second derivative in a graph, we need to store an extra value in each node in the graph, this extra value corresponds to the second derivative of the node's function. Figure 3.5 depicts function $h(g(f(x)))$ using the extra storage to keep f'', g'', as well as h''. In fact, the only difference in the forward step between the first and second derivative is this extra value that correspond to the second derivative.

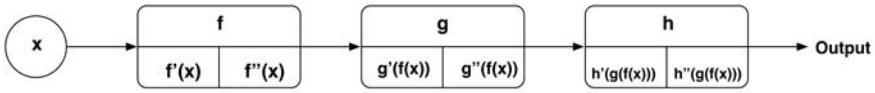


Figure 3.5: Information stored to compute second partial derivative

On the other hand, the backward step requires some additions to the procedure used to compute the first derivative. Figure 3.6 depicts the backward steps used to compute the second partial derivative: the dashed edge represents the first derivative process, and, the solid edges represent the additional process required by the second partial derivative at each node. The second derivative is computed as: $\alpha(x) \cdot c_2 + \beta(x) \cdot c_1^2$ where $\alpha(x)$ and

$\beta(x)$ are the first and second derivative values propagated; and c_1 and c_2 are the values stored at the node being processed (done in the forward step) that correspond to the first (c_1) and second (c_2) derivatives, respectively. Analogously, following this notation, the first derivative can be computed by $\alpha(x) \cdot c_1$. The process to compute both partial derivatives is depicted in Algorithm 1.

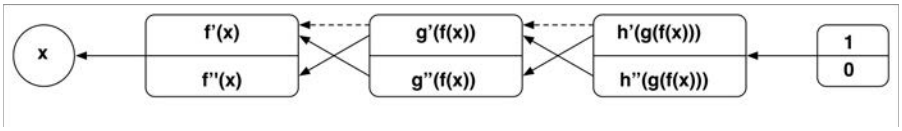


Figure 3.6: Backward step to compute the second derivatives with three functions

In order to clarify the process used to compute the second derivative, let us describe all the steps required to compute $\frac{\partial^2 h(g(f(x)))}{\partial^2 x}$ (depicted in Figure 3.6), i.e., the second partial derivative of function h, *w.r.t.* x using α, β, c_1 and c_2; α and β are enhanced with a subscript that indicates the node where the first or second derivative were computed, respectively.

Algorithm 1: Algorithm to compute $\frac{\partial^2 E}{\partial v^2}$ and $\frac{\partial E}{\partial v}$

```
1 function ComputeDerivatives (P, β, α);
  Input  : P: Path to node v, β_0 and α_0
  Output: β_n, α_n
2 for n = node in P and i = (1, length(P)) do
3 |   c_1^n = 1st. Value stored in n ;
4 |   c_2^n = 2nd. Value stored in n ;
5 |   β_i = α_{i-1} · c_2^n + β_{i-1} · (c_1^n)^2;
6 |   α_i = α_{i-1} · c_1^n ;
7 end
8 return β_i, α_i;
```

1. Values $\alpha(x)$ and $\beta(x)$ are set to 1 and 0, respectively.

2. In Node h:
   $\beta_h(x) = \alpha(x) \cdot c_2 + \beta(x) \cdot c_1^2 = h''(g(f(x)))$
   $\alpha_h(x) = \alpha_i(x) \cdot c_1 = h'(g(f(x)))$

3. In Node g:
   $\beta_g(x) = \alpha_h(x) \cdot c_2 + \beta_h(x) \cdot c_1^2 = h'(g(f(x)))g''(f(x)) + h''(g(f(x)))g'(f(x))^2$
   $\alpha_g(x) = \alpha_h(x) \cdot c_1 = h'(g(f(x)))g'(f(x)))$

4. In Node f:
   $\beta_f(x) = \alpha_g(x) \cdot c_2 + \beta_g(x) \cdot c_1^2 = h''(g(f(x)))g'(f(x))^2 f'(x)^2 + h'(g(f(x)))g''(f(x))f'(x)^2 +$
   $h'(g(f(x)))g'(f(x))f''(x)$
   $\alpha_f(x) = h'(g(f(x)))g'(f(x)))f'(x)$

5. Algorithm ends because it reached node x.

At the end of the previous steps, the last β computed by the algorithm is $\beta_f(x) = h''(g(f(x)))g'(f(x))^2 f'(x)^2 + h'(g(f(x)))g''(f(x))f'(x)^2 + h'(g(f(x)))g'(f(x))f''(x)$. In order to show that this term is indeed the second partial derivative of function h w.r.t. variable x, let us derive $\frac{\partial^2 h(g(f(x)))}{\partial^2 x}$ by formulae:

$$
\begin{aligned}
\frac{\partial^2 h(g(f(x)))}{\partial x^2} &= \frac{\partial}{\partial x}\left[h'(g(f(x)))g'(f(x))f'(x)\right] \\
&= \frac{\partial}{\partial x}\left[h'(g(f(x)))\right] g'(f(x))f'(x) + h'(g(f(x)))\frac{\partial}{\partial x}\left[g'(f(x))f'(x)\right] \\
&= h''(g(f(x)))g'(f(x))f'(x)g'(f(x))f'(x) + \\
&\quad h'(g(f(x)))\frac{\partial}{\partial x}\left[g'(f(x))\right] f'(x) + h'(g(f(x)))g'(f(x))\frac{\partial}{\partial x}\left[f'(x)\right] \\
&= h''(g(f(x)))g'(f(x))^2 f'(x)^2 + h'(g(f(x)))g''(f(x))f'(x)f'(x) + \\
&\quad h'(g(f(x)))g'(f(x))f''(x) \\
&= h''(g(f(x)))g'(f(x))^2 f'(x)^2 + h'(g(f(x)))g''(f(x))f'(x)^2 + \\
&\quad h'(g(f(x)))g'(f(x))f''(x).
\end{aligned}
$$

The formulae used to derive the aforementioned expression were the chain rule and the derivative of products of functions. It is important to note that this example considers only three functions, i.e., h, g and f, nonetheless the methodology can be easily extended to any number of functions because it relies on the chain rule.

3.2.1. Example of the computing of $\frac{\partial^2 E}{\partial v^2}$ in an individual

So far, we have described how Backpropagation algorithm computes the second partial derivatives of a function *w.r.t.* a node in a tree. However, in order to use the second

partial derivative information for guiding the crossover operation in GP, we need the second partial derivative of the error function *w.r.t.* the node selected as the crossover point. The error function considered in this work is $E = (y - \hat{y})^2$, where y is the target and $\hat{y}$ is the output of a given individual.

Specifically, given that E is the error function, we need to obtain the second partial derivative of E *w.r.t.* some node v, i.e., the crossover point in the first parent. To do so, the first step is to compute the second partial derivative of E *w.r.t.* the output of the individual i,e., $\hat{y}$ this step corresponds to the first term of the chain rule. This derivative is $\frac{\partial^2 E}{\partial \hat{y}^2} = \frac{\partial}{\partial \hat{y}}\left[\frac{\partial E}{\partial \hat{y}}\right] = \frac{\partial}{\partial \hat{y}}\left[-2(y - \hat{y})\right] = 2$. In order to compute $\frac{\partial^2 E}{\partial v^2}$, instead of starting the algorithm with the propagated values of $\alpha(x) = 1$ and $\beta(x) = 0$, as in the previous example, the algorithm starts with $\alpha_E(x) = \frac{\partial E}{\partial \hat{y}} = -2e$ and $\beta_E(x) = \frac{\partial^2 E}{\partial \hat{y}^2} = 2$ which are the first terms being propagated, where e is the difference between the target and the output of an individual i.e., $e = (y - \hat{y})$.

Let us explain the process with an example. Consider function that $\hat{y} = 1.5x^2 - 0.7x + 1.2$ depicted in Figure 3.7 (a) is codified in an individual, and, let us also assume that this individual is selected as the first parent for crossover. Additionally, the crossover point (node v) selected in this individual is the second constant of the function $\hat{y}$ i.e., -0.7.

Figure 3.7 (b) shows the steps made by backpropagation to compute the term $\frac{\partial^2 E}{\partial v^2}$. Note that in the figure, the root node is function E (although the root of the individual is the right child of node E), which takes two arguments y and $\hat{y}(x)$. Additionally, the target $y(x)$ and the part $1.5x^2$ of $\hat{y}(x)$ are simplified and the nodes involved in the path from the root to the crossover point are highlighted in red. Finally, the first and second derivatives stored for each function correspond only to the required derivatives for the highlighted path.

For the example in Figure 3.7 (b), backpropagation will perform the following steps:

1. $\beta_1 = \alpha_E \cdot c2 + \beta_E \cdot c1^2 = -2e \cdot 0 + 2 \cdot 1^2 = 2$
   $\alpha_1 = \alpha_E \cdot c1 = -2e \cdot 1$

2. $\beta_2 = \alpha_1 \cdot c2 + \beta_1 \cdot c1^2 = -2e \cdot 0 + 2 \cdot 1^2 = 2$
   $\alpha_2 = \alpha_1 \cdot c1 = -2e \cdot 1$

3. $\beta_3 = \alpha_2 \cdot c2 + \beta_2 \cdot c1^2 = -2e \cdot 0 + 2 \cdot (x)^2 = 2x^2$
   $\alpha_3 = \alpha_2 \cdot c1 = -2e \cdot x = -2ex$

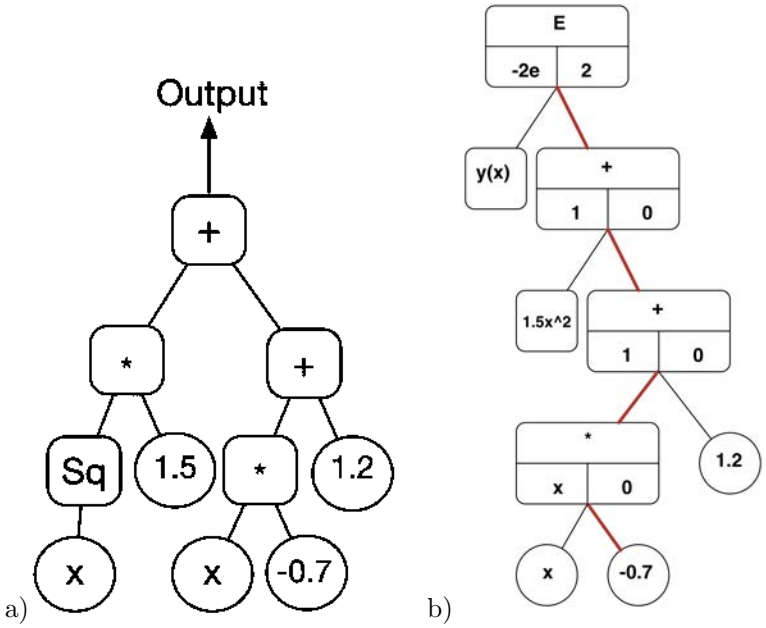


Figure 3.7: a) depicts a simple function. b) presents the corresponding tree with the information stored in the forward step, the path follow (red edges) to reach the crossover point and the error function to illustrate the backward step of $\frac{\partial^2 E}{\partial v^2}$.

4. Algorithm stops because it reached node v.

When node v is reached, the propagated terms are: $\alpha = -2ex$ and $\beta = 2x^2$. These values are indeed the first and second partial derivatives of $E = (y - ax^2 - vx - c)^2$ *w.r.t.* to v (note that $-\hat{y}$ is rewritten to $-ax^2 - vx - c$). The next equations show the derivatives.

$$\frac{\partial E}{\partial v} = 2(y - ax^2 - vx - c)(-x) = -2ex$$
$$\frac{\partial^2 E}{\partial v^2} = \frac{\partial}{\partial v} 2(y - ax^2 - vx - c)(-x) = \frac{\partial}{\partial v} 2vx^2 = 2x^2$$

3.2.2. Using $\frac{\partial^2 E}{\partial v^2}$ and the Newton Method

At this point, it is natural to ask: how can the information provided by the partial derivatives be used to improve the individuals in GP?. Perhaps, the first approach that came to mind is to perform the Newton Method, given that the error function E, is quadratic. In other words, treat the output of the node chosen as the crossover point as if the outputs were the initial points in the newton method, using the first and second derivatives to compute

the next iteration of the newton method. The new output given by this iteration is the desired output in the crossover, in other words, the desired output of the child produced by the crossover. The newton method is presented in Equation 3.1 where $\hat{\mathbf{v}}_i$ is the desired output, $\mathbf{v}_i$ is the initial point that corresponds to v's output at i^th training case, and $\frac{\partial E}{\partial v}$ and $\frac{\partial^2 E}{\partial v^2}$ are the first and second derivatives the terms, respectively.

$$\hat{\mathbf{v}}_i = \mathbf{v}_i - \frac{\partial E}{\partial v}\left(\frac{\partial^2 E}{\partial v^2}\right)^{-1} \tag{3.1}$$

Lets explain the newton method with the previous example (Figure 3.7) but this time more information is given:

* The samples $x = [-1, -0.5, 0, 0.5, 1]$
* The error $e = y(x) - \hat{y}(x) = [-1.5, -0.75, 0, -0.75, -1.5]$
* Output of node v, $\mathbf{v} = [-0.7, -0.7, -0.7, -0.7, -0.7]$
* First partial derivative $\frac{\partial E}{\partial v} = -2ex = [-3, -0.75, 0, -0.75, -3]$
* Second partial derivative $\frac{\partial^2 E}{\partial v^2} = 2x^2 = [2, 0.5, 0, 0.5, 2]$

where x contains the values for the function's variable, the derivatives $\frac{\partial E}{\partial v} = -2ex$ and $\frac{\partial^2 E}{\partial v^2} = 2x^2$ are the previously computed derivatives but this time evaluated at x and the error e is also evaluated at x.

Since the output of node v happens to be a constant of the function codified by the individual, let us take the sum of the evaluated first and second partial derivatives, -7.5 and 5 respectively. The first iteration of the newton method yields $\hat{\mathbf{v}} = -0.7 - \frac{-7.5}{5} = -0.7 + 1.5 = 0.8$. This means that the output of node v which is currently the constant -0.7 has to be 0.8. In other words, the first iteration of the Newton method indicates that the constant 0.7 need to be replaced with the constant 0.8 in order to minimize the error.

Finally, if the node v is changed to constant 0.8 the new individual would be $\hat{y}(x) = 1.5x^2 + 0.8 + 1.2$, which is indeed the function we used to build the target values for this example. Summarizing, the Newton method can be combined with backpropagation in GP to *fine tune* constants in individuals. However, the crossover point can be any node (function, variable or constant), consequently, the sum of partial derivatives cannot be used directly. Therefore, next section starts rectifying this scenario by proposing semantic

crossover operators that use the information provided by the derivatives to generate an offspring.

3.3. Semantic Crossover for Genetic Programming Using Partial Derivatives

So far, the Backpropagation algorithm has been adapted to GP in order to compute the first and second partial derivatives. It is of particular interest to compute the derivative of fitness function E *w.r.t.* node v given that this derivative contains information about the similarity between v's output and the target behavior. Specifically, the information given is whether v's output needs to be increased or decreased in order to minimize the fitness function.

With the information provided by the first and second partial derivatives we decided to develop three different crossover operators for GP. The main difference among the aforementioned operators rely on the degree of the partial derivative used to select the crossing points. In the next subsections the three crossovers operators and their definitions are presented.

3.3.1. Crossover based on the first derivative

That is, let us assume that node v is selected as crossover point in the first parent, then the idea is to select the crossover point of the second parent (i.e., u) using the information provided by the derivative. Equation 3.2 shows our first crossover operator based only on the information provided by the first derivative. It works as follows: let $\mathbf{k}$ be $\text{sign}(\frac{\partial E}{\partial v})$, $\mathbf{v}$ be v's output, $\mathbf{s}$ is the output of a given node s at the second parent, $\mathcal{S}$ contains all the nodes at the second parent, and, i iterates for all the training cases. Using this notation, the crossover point of the second parent, i.e., u, is computed as:

$$u = \arg\max_{s \in \mathcal{S}} \sum_{i=1}^{N} \text{sign}(\mathbf{v}_i - \mathbf{s}_i) \cdot \mathbf{k}_i. \tag{3.2}$$

In order to clarify Equation 3.2, let us analyze the two possible scenarios that would lead $\sum_i \text{sign}(\mathbf{v}_i - \mathbf{s}_i) \cdot \mathbf{k}_i$ to get its maximum value. Let us assume that u is the crossover point that produced this maximum value, and $\mathbf{u}$ contains u's output. In the case $\mathbf{k}_i > 0$, then the value $\mathbf{v}_i$ should be decreased, consequently $\mathbf{v}_i - \mathbf{u}_i$ should be positive.

On the other hand, $\mathbf{k}_i < 0$ implies that $\mathbf{v}_i$ should be increased, hence, $\mathbf{v}_i - \mathbf{u}_i$ should be negative; however, $\mathbf{k}_i < 0$ implies that the result is positive. Under these circumstances, $\sum_i \text{sign}(\mathbf{v}_i - \mathbf{s}_i) \cdot \mathbf{k}_i$ equals $|\mathcal{T}|$ (the cardinality of the training set). In other words, this crossover operator randomly selects the crossover point in the first parent, i.e., v, computes the derivative of fitness function E *w.r.t.* v, and based on this information, selects from $\mathcal{S}$ the subtree that maximizes the desired sign, i.e., slope. Algorithm 2 shows the steps required to compute the crossing point u according to the first partial derivative information.

Algorithm 2: Selection of second crossing point u based on first derivative information

```
1  GPPDE (∂E/∂v, v, P2);
   Input  : P2: Second parent, output of crosspoint v and its partial
            derivative ∂E/∂v
   Output: u: Cross point in second parent
2  k = sign(∂E/∂v) ;
3  N = number of training cases ;
4  S = nodes in P2 ;
5  crossPoints = array[length(S)] ;
6  for s = node in S do
7  |  sum = 0 ;
8  |  for i = 1 to N do
9  |  |  sum+ = sign(v_i − s_i) · k_i ;
10 |  end
11 |  crossPoints_s = sum ;
12 end
13 u = argmax(crossPoints) ;
14 return u ;
```

3.3.2. Crossover Based on the Newton Method

Given that the second derivative of E *w.r.t.* v (the node that represents the crossing point in the first parent) is available, the next step is to select node u using the information provided by the Newton method. That is, with the first iteration of the newton method

computes v's desired output. With this vector, one can perform a search in the second parent, among all possible subtrees, and select the subtree whose output has the minimum euclidean distance with the vector obtained by the newton iteration. Such scheme is presented in Equation 3.3, where $\hat{\mathbf{v}}_i$ is the first iteration of the newton method, $\mathbf{s}_i$ is s's output at the i^{th} training case, $\mathcal{S}$ contains all the nodes of the second parent, and N is the size of the training set.

$$u = \arg\min_{s \in \mathcal{S}} \sum_{i=1}^{N} \sqrt{(\hat{\mathbf{v}}_i - \mathbf{s}_i)^2} \tag{3.3}$$

Equation 3.3 shows a crossover operator which uses the information provided by partial derivatives to select the subtree in the second parent and perform the crossover. However, this scheme has a major drawback, the *Vanishing Gradient* problem [Hochreiter01]. This problem states that the magnitude of the gradient decreases exponentially in the front layers of an ANN. This problem affects the ANN training algorithms with a gradient descent strategy like backpropagation. To our purposes, the vanishing gradient problem means that if the crossover point selected corresponds to a deep node, the magnitude of the first partial derivative will be practically zero. In that case, the term $\frac{\partial E}{\partial v}$ in Equation 3.1 will be zero; as a result the new point is the same as the current point. Algorithm 3 shows the steps required to select the crossing point v according to the first iteration of the Newton method.

3.3.3. Crossover Based On The First Derivative And The Newton Method

It is worth to notice that the first crossover operator presented in this work, that uses only the first partial derivative information (Equation 3.2), is not affected by the vanishing gradient problem. This crossover operator is guided only by the *sign* of the derivative instead of by its magnitude, so it is immune to the decay of the gradient. Because of this, we decided to combine both strategies to develop a crossover operator that uses the information of the newton method with the sign of the first partial derivative Equation 3.4. This crossover is similar with the previous operator, with the difference that the euclidean distance is divided by a factor given by the first partial derivative. This factor represents the number of times the output of subtree s at the second parent is suitable according to the sign of the derivative. In fact, this information is what is being used in Equation 3.2.

Algorithm 3: Selection of second crossing point u based on second derivative information

GPPDE2 $(\frac{\partial E}{\partial v}, \frac{\partial^2 E}{\partial v^2}, \mathbf{v}, P2)$;

Input : $P2$: Second parent, output of crosspoint v and its firts and second partial derivatives $\frac{\partial E}{\partial v}$, $\frac{\partial^2 E}{\partial v^2}$

Output: u: Cross point in second parent

$\hat{\mathbf{v}} = \mathbf{v} - \frac{\partial E}{\partial v} \left(\frac{\partial^2 E}{\partial v^2} \right)^{-1}$;

$N =$ number of training cases ;

$S =$ nodes in $P2$;

$crossPoints = array[length(S)]$;

for $s = node\ in\ S$ **do**

  $sum = 0$;

  for $i = 1\ to\ N$ **do**

    $sum+ = \sqrt{(\hat{\mathbf{v}}_i - s_i)^2}$;

  end

  $crossPoints_s = sum$;

end

$u = argmin(crossPoints)$;

return u ;

$$u = \arg\min_{s \in \mathcal{S}} \frac{\sum_{i=1}^{N} \sqrt{(\hat{\mathbf{v}}_i - \mathbf{s}_i)^2}}{\sum_{i=1}^{N} \text{sign}(\mathbf{v}_i - \mathbf{s}_i) \cdot \mathbf{k}_i.} \tag{3.4}$$

The idea behind this operator is to compute the euclidean distance between the iteration of the newton method and the subtrees of the second parent, and reward those subtrees whose output follow the sign of the derivative by dividing their euclidean distance by the number of times its sign coincided with the the sign of the first partial derivative. On the other hand, subtrees whose output does not follow the sign of the derivative at any point, are penalized and are discarded for selection (this also avoids by zero division in Equation 3.4). Algorithm 4 depicts the combination of the two previous algorithms 2 and 3.

3.4. Summary

In summary, this contribution proposes three different procedures to select the crossover point of the second parent, namely point u. In order to do so, one needs to select two parents, select the crossover point of the first parent, i.e. v, and then compute the partial derivatives of the fitness function E *w.r.t.* v. With these derivatives, the three different procedures proposed to select u are described in Algorithms 2, 3 and 4. From now on, we will refer as GPPDE to the GP system implementing the first strategy which corresponds to first derivative information, GPPDE2 will be used to refer to the second proposal which uses second derivative information, and GPPDEC corresponds to the combination of the two later proposals and which uses first and second partial derivatives information.

In this chapter, we have presented our proposal of semantic crossover operators for GP. The main idea behind the three operators developed is using the information provided by the first and second partial derivative of the error function to guide the selection of the crossing point in the second parent. The first derivative can be interpreted as the direction on which the output of the sub-tree with the first crossing point as root, helps to minimize the error function. On the other hand, the second partial derivative can be used in conjunction with the first partial derivative and perform the first iteration of the Newton Method in order to minimize the error contribution of the sub-tree represented by the first crossing point. Particularly, the newton method can be applied in GP to *fine-tune* constants in

Algorithm 4: Selection of second crossing point u based on first and second derivatives information

GPPDEC $(\frac{\partial E}{\partial v}, \frac{\partial^2 E}{\partial v^2}, \mathbf{v}, P2)$;

Input : $P2$: Second parent, output of crosspoint v and its firts and second partial derivatives $\frac{\partial E}{\partial v}$, $\frac{\partial^2 E}{\partial v^2}$

Output: u: Cross point in second parent

$k = sign(\frac{\partial E}{\partial v})$;

$\hat{\mathbf{v}} = \mathbf{v} - \frac{\partial E}{\partial v}\left(\frac{\partial^2 E}{\partial v^2}\right)^{-1}$;

$N =$ number of training cases ;

$S =$ nodes in $P2$;

$crossPoints = array[length(S)]$;

for $s = node\ in\ S$ **do**

  $sum = 0$;

  for $i = 1\ to\ N$ **do**

    $sum+ = \frac{\sqrt{(\hat{\mathbf{v}}_i - s_i)^2}}{sign(\mathbf{v}_i - s_i) \cdot k_i}$;

  end

  $crossPoints_s = sum$;

end

$u = argmin(crossPoints)$;

return u ;

individuals.

Chapter 4

Symbolic Regression and Semantic Crossover

In order to test the crossover operators developed in the previous chapter, we decided to apply GP to different problems. The first of these problems is called *Symbolic Regression*. Before explaining what symbolic regression is, first we need to introduce the concept of regression. Traditionally, regression refers to the process of modelling the relationship between one or more explanatory variables X and one dependant variable y. If the relationship between variables is assumed to be linear, the problem is called linear regression. The parameters of the linear model are inferred from the data and estimated with different techniques, for example linear squares, Figure 4.1 shows a simple example of linear regression where there is one explanatory variable (x-axis) and one dependent variable (y-axis). In this example, the model that fits the data is $\hat{y} = \theta X + \epsilon$, where θ and ϵ are the estimated parameters.

However, there are data in many areas whose relationship between variables cannot be assumed to be linear. In those cases, other models are used to describe the data for example *polynomial regression*, where the relationship is modelled by an $n-th$ degree polynomial. Furthermore, when the relationship is not linear nor polynomial, or one just simply do not want to assume any relationship between the data, the problem is called symbolic regression. In symbolic regression, the model itself and its parameters are inferred from the data. In this scenario, the application of GP results quite effective because GP evolves trees that represent different models; each individual represents a model. The parameters of the

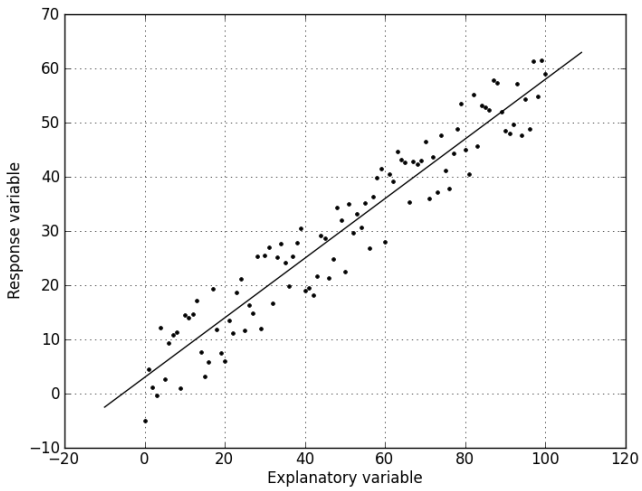


Figure 4.1: Example of Linear Regression

models represented by the individuals can be seen as a subset of the possible combinations of the function set and the terminal set.

4.1. Problems and Parameter Settings

Table 4.1 shows the problems used to testing semantic operators in symbolic regression whereas Table 4.2 shows the parameters used by the GP systems. Table 4.1 shows the 9 functions used for regression. In order to build the training set, the functions are sampled 21 times, uniformly distributed in the specified range. The validation set is built sampling the function another 21 additional points, but contrarily to the training set, these samples are randomly distributed in the range. We decided to include these functions because they have been used to test other semantic operators, so we can compare our proposals to other operators.

4.2. Results

Let us start with the symbolic regression problems using the Euclidean Distance as the performance measure. Each GP system was run 30 times, and, the average performance

Table 4.1: Symbolic Regression Functions

Problem	Formula	Range
Keijzer1	$0.3sin(x)(2\pi x)$	[-1,1]
Keijzer4	$x^3exp(-x)cos(x)sin(x)(sin^2(x)cos(x)-1)$	[0,10]
Nguyen6	$sin(x)+sin(x+x^2)$	[-1,1]
Nguyen7	$log(x+1)+log(x^2+1)$	[0,2]
Nonic	$\sum_{i=1}^{9} x^i$	[-1,1]
R1	$\frac{(x+1)^3}{x^2-x+1}$	[-1,1]
R2	$\frac{x^3-3x^3+1}{x^2+1}$	[-1,1]
R3	$\frac{x^6+x^5}{x^4+x^3+x^2+x+1}$	[-1,1]
Septic	$x^7-2x^6+x^5-x^4+x^3-2x^2+x$	[-1,1]

is reported for the training set, and, the median for the test set. The median allow us to compare our results to other semantic operators previously reported in the literature.

In the results tables, the best performance is in bold face in order to facilitate the reading. In addition to this, the cases where the difference on performance is statistically significant (with a confidence level of 95 %) are indicated with an $*$. The statistical analysis was done using the Wilcoxon signed rank test [Wilcoxon45]. Only the performance in the training set is under this statistical analysis.

Table 4.3 shows the results for the training set of the three proposed semantic operators. Additionally, the last column presents the results for traditional GP with syntactic crossover. From the table, it can be seen that the operator that trains better is GPPDEC, achieving the best results in 6 out of 9 problems (4 of them with a statistically significant difference). GPPDE won 3 problems and GPPDE2 won 1 problem. GP was in last place with zero best results.

In Table 4.4, the performance in the test set is presented. In this table, we can see that GPPDEC is the clear winner obtaining the best performance in 8 out of 9 problems, and tied with the other operators in 1 problem (all GP systems were particularly good in Nguyen6 problem). Again, in second place was GPPDE, which obtained better performance than than GPPDE2 and GP. Comparing the operators having the worst performance, namely GP and GPPDE2, it is observed that GPPDE2 is better in 7 out of 9 problems, and tied in 1 problem. This tells us that even the worst semantic operator proposed is able

Table 4.2: Parameter Settings in GP Systems for Symbolic Regression

Parameter	Value
Mutation Depth	random $\in [1, 5]$
Selection	Tournament size 4
Population Size	1024
Number of generations	100
Function Set (F)	+,-,*,/,exp,sin,cos,ln
Crossover rate	100 %
Mutation rate	0 %
Max length	512

Table 4.3: Performance of Semantic Crossover Operators and GP in Symbolic Regression Problems (Training Set). Best perfomance in boldface; when the best perfomance is statistically significant (with a confidence of 95 %) is indicated with * as superscript.

Problem	GPPDE	GPPDE2	GPPDEC	GP
Keijzer1	0.024	0.041	**0.022**	0.080
Keijzer4	**0.165**	0.205	0.178	0.264
Nguyen6	**0.000**	0.005	0.0002	0.009
Nguyen7	0.003	0.203	**0.001***	0.017
Nonic	0.040	0.085	**0.020***	0.156
R1	0.041	0.081	**0.033***	0.128
R2	0.109	**0.039**	0.050	0.210
R3	0.007	0.018	**0.003**	0.016
Septic	0.073	0.062	**0.024***	0.138

to achieve better results than the traditional syntactic crossover.

The last tables presented the average and median performance at the end of the evolutionary process, in order to complement this information, Figures 4.2 and 4.3 present the evolution of the fitness throughout the generations. Figures present 9 plots, each for every symbolic regression problem, containing the performance of the operators in terms of the euclidean distance (*w.r.t.* the target) at each generation. In order to keep the information easy to read, the plots include only the two best operators, namely GPPDEC and GPPDE, although traditional GP was added into the comparison in order to have a reference.

Table 4.4: Performance of Semantic Crossover Operators and GP in Symbolic Regression Problems (Test Set). Best performance in boldface.

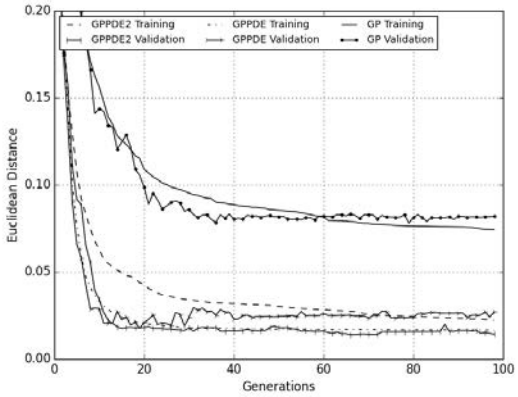
Problem	GPPDE	GPPDE2	GPPDEC	GP
Keijzer1	0.029	0.052	**0.014**	0.086
Keijzer4	0.572	0.636	**0.458**	0.754
Nguyen6	**0.000**	**0.000**	**0.000**	**0.000**
Nguyen7	0.002	0.013	**0.0005**	0.010
Nonic	0.050	0.084	**0.019**	0.130
R1	0.034	0.035	**0.015**	0.065
R2	0.074	0.029	**0.024**	0.297
R3	0.021	0.016	**0.007**	0.022
Septic	0.150	0.055	**0.044**	0.155

From these plots, lets look at the training curves. It can be seen that in none of the problems the training curve of GP trains faster or better than GPPDEC or GPPDE. This slow learning compared to the proposed semantic operators is more evident in the problems Keijzer1, Keijzer4, Nonic, R1, R2 and Septic. Both GPPDEC and GPPDE training curves present a faster learning compared to GP for most of the symbolic problems. Due to this, one can stop the learning process in early generations (10-20 generations) for the semantic operators, and still obtain better results than with traditional GP using a larger number of generations.

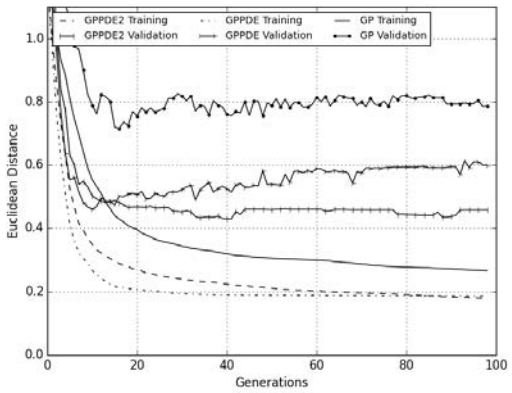
The shapes of the training for GPPDEC and GPPDE curves are very similar for most of the problems (almost identical in Nguyen6, Nguyen7, and R3). The only noticeable difference is the magnitude of the euclidean distance. It is important to notice that both algorithms exhibit training curves that converge in a small number of generations, the curves reached almost the same performance at early generations than the one obtained in the last generation (100^{th} generation).

The difference in the curves of GPPDEC and GPPDE is more evident in the test set. The plots of problems where overfitting is noticeable, like Keijzer4 and Septic, show this difference. GPPDEC's validation curve seems to minimize the effect of overfitting. For example, in Keijzer4, GPPDE obtains its minimum validation euclidean distance near the 10_{th} generation, and beyond that point, as the number of generations increased, so did the euclidean distance for validation. However, this behavior is not present in GPPDEC's

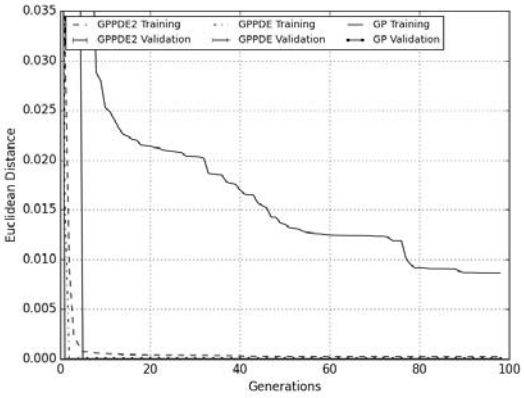
validation curve, where it appears that overfitting has been minimized. This minimization effect of the overfitting is also seen in Septic. This may be the reason why GPPDEC exhibits the least validation error of all the three semantic crossover operators.



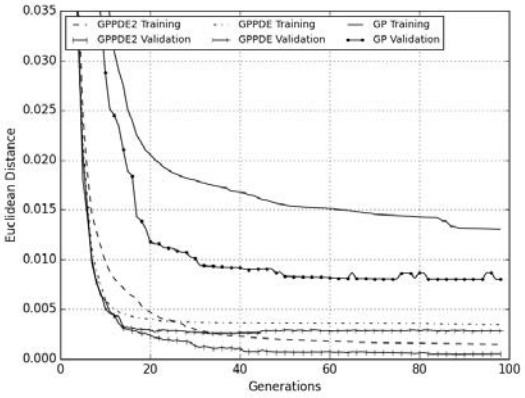
(a) Keijzer1



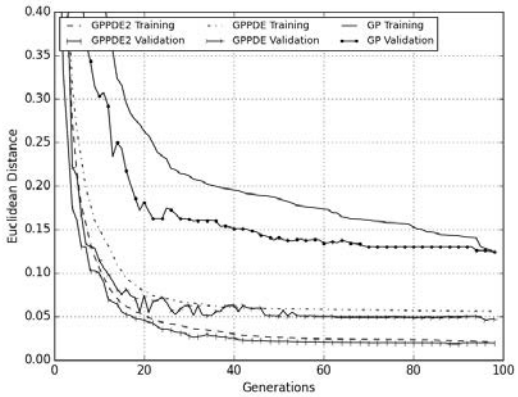
(b) Keijzer4



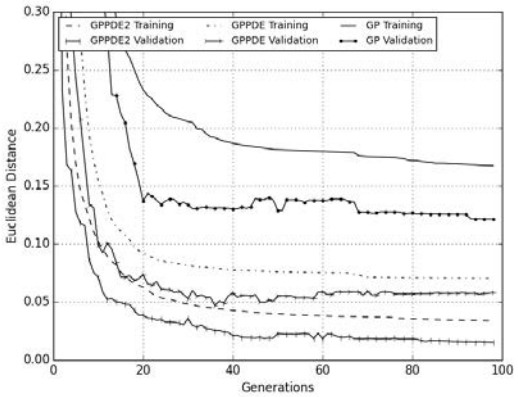
(c) Nguyen6



(d) Nguyen7

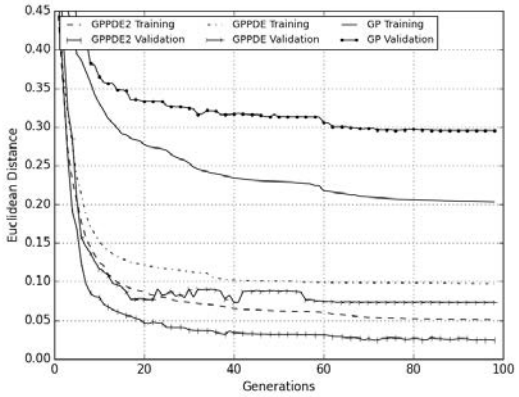


(e) Nonic

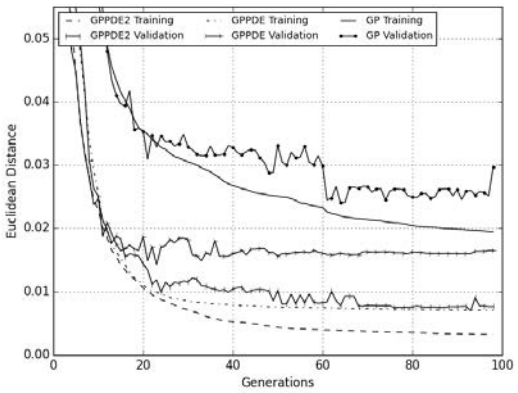


(f) R1

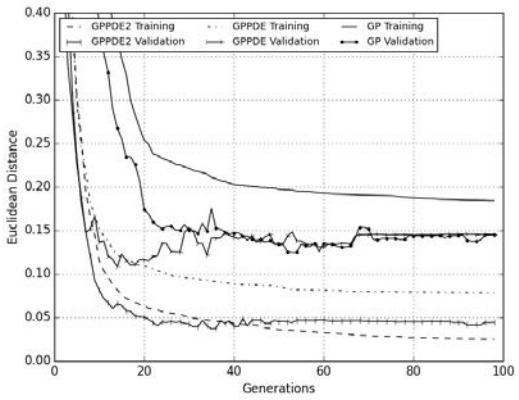
Figure 4.2: Symbolic Regression Problems 1-6



(a) R2



(b) R3



(c) Septic

Figure 4.3: Symbolic Regression Problems 7-9

Besides comparing the three semantic crossover operators proposed to GP, we have made another comparison against state of the art semantic crossover operators: Approximately Geometric Crossover (AGX) [Pawlak14], Locally Geometric Crossover (LGX) [Krawiec12], Krawiec and Lichocki Geometric Crossover (KLX) [Krawiec09], and Geometric Semantic Genetic Programming (GSGP) [Moraglio12b]. Of all the three operators presented in this work, only GPPDEC was included into this comparison because it was the operator that presented the best results among the proposed operators. Again, GP is included as reference.

The training results of this comparison are shown in Table 4.5, where the bold

Table 4.5: Performance of Different State of the art Semantic Crossovers on Symbolic Regression Problems (Training Set). Best performance in boldface.

Problem	GPPDEC	AGX	LGX	KLX	GSGP	GP
Keijzer	0.022	**0.001**	0.032	0.134	0.279	0.080
Keijzer4	0.178	**0.010**	0.140	0.455	0.667	0.264
Nguyen6	**0.0002**	0.001	0.002	0.027	0.197	0.009
Nguyen7	0.001	**0.000**	0.010	0.052	0.046	0.017
Nonic	0.020	**0.005**	0.041	0.223	0.759	0.156
R1	0.033	**0.005**	0.014	0.177	0.590	0.128
R2	0.050	**0.003**	0.034	0.163	0.206	0.210
R3	0.003	**0.001**	0.003	0.038	0.093	0.016
Septic	0.024	**0.004**	0.023	0.266	1.136	0.138

numbers indicate the best performance. As can be seen from the table, the three operators that presented better results were GPPDEC, AGX and LGX. AGX was the operator that presented the best results winning 8 of 9 problems with a wide margin. Comparing GPPDEC and LGX alone, each operator won 4 out of 9 problems, and they tied in one problem (R3). GPPDEC was able to train the best in Nguyen6, which was the only problem that AGX did not won in the training set.

Table 4.6 shows the validation results. In this case, GPPDEC was the best method for validation, it presented the best results with 5 problems won, followed by AGX and LGX with two problems each. Again, KLX, GSGP and GP were outperformed by GPPDEC, AGX and LGX.

4.3. Summary

In this chapter the three proposed semantic operators were tested in the Symbolic Regression problem. A set of benchmarks were used to measure the performance achieved by GPPDE, GPPDE2, and GPPDEC. The crossover operators outperformed traditional syntactical crossover. Different convergence plots were presented to show this statement. In such plots, it can be seen that the convergence of our semantic operators is faster and with less error than syntactical crossover. Moreover, the three operators were compared with other semantic operators reported in the literature. In this comparison, the semantic

Table 4.6: Performance of Different State of the art Semantic Crossovers on Symbolic Regression Problems (Test Set) Best performance in boldface.

Problem	GPPDEC	AGX	LGX	KLX	GSGP	GP
Keijzer	**0.014**	0.015	0.049	0.151	0.275	0.086
Keijzer4	0.458	**0.309**	0.478	0.585	0.911	0.754
Nguyen6	**0.000**	0.001	0.002	0.009	0.180	0.000
Nguyen7	**0.0005**	0.001	0.003	0.041	0.038	0.010
Nonic	**0.019**	0.061	0.081	0.172	0.618	0.130
R1	0.015	0.030	**0.008**	0.156	0.630	0.065
R2	0.024	**0.011**	0.038	0.133	0.215	0.297
R3	**0.007**	0.009	0.008	0.027	0.084	0.022
Septic	0.044	0.032	**0.026**	0.271	1.210	0.155

crossover operators proposed in this work achieved better performance than most of the semantic crossover operators included in the comparison.

Chapter 5

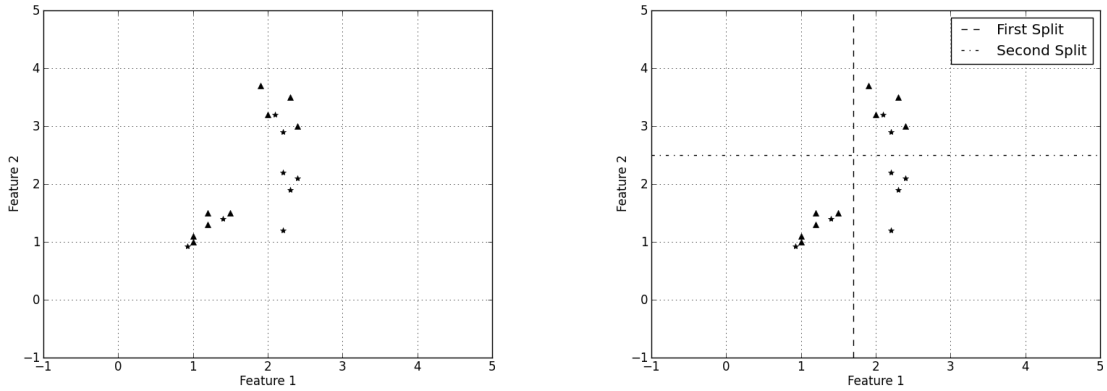
Classification and Semantic Crossover

The second problem on which we tested the semantic crossover operators is *Classification*. The problem of Classification, as its name suggests, consists in classify data in a certain number of classes. Data is presented in form of observations (examples), each with a certain number of features and the particular class that the observation belongs to. In this scheme, a particular observation of the data can belong only to one class (although in multi-class classification a certain observation can belong to two or more classes). Therefore, the problem of classification can be seen as the search of a model that maps the space of features to a class: $f : \Re^N \rightarrow C_k$, where f is the classifier function, N represents the number of real-valued features, C is a set of classes, and C_k represents the k^{th} class.

In Figure 5.1, a simple example of the problem of classification is presented. In the figure, there is data that belongs to one of two classes: star and triangle. The features that determine whether the objects belong to one class or another are simply called *feature1* and *feature2*. One of the most used approaches to classify data are *decision trees*. Decision trees compare the features' values of the observations with certain predefined rules and performs splits on the data. Depending on these splits, new data is labelled according to the majority of observations that belong to a certain class. In the right part of Figure 5.1, a pair of splits are proposed to classify data. The decision tree that performs such splits is shown in Figure 5.2.

The decision tree makes comparison between the features in the training data and

certain rules, for example $f1 > 1.7$, which means that the feature number one must be greater that 1.7, and then counts the number of elements of each class that follow the rule. In the previous rule, the objects that follow this rule are 5 triangles and 2 stars. These splits continue over the characteristics until all the data follow some rule. When new objects are presented to the decision tree, the value of their characteristics will follow some previous rule and it will be labelled to one class depending on which class had the majority of members for that rule.



(a) Original Data (b) Classified Data

Figure 5.1: Example of Classification

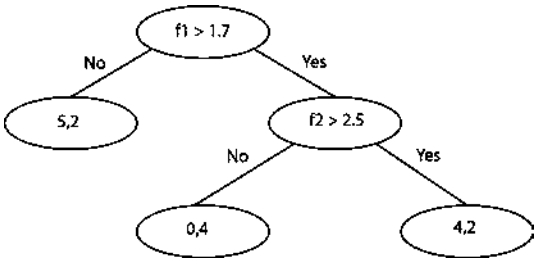


Figure 5.2: Example of a Decision Tree

Although in the last example, classification seems to be a simple task, it is not. In some problems, the big number of characteristics makes it difficult to create suitable rules to split the data. Many techniques have been proposed to solve classification problems, GP is one of them. Just like in regression problems, the characteristics of classification

Table 5.1: Classification Problems

Data Set	Input Features	Training set instances	Test set instances
Banana	2	400	4900
Titanic	3	150	2051
Thyroid	5	140	75
Diabetes	8	468	300
Breast-Cancer	9	200	77
Flare-Solar	9	666	400
Heart	13	170	100
Ringnorm	20	400	7000
Twonorm	20	400	7000
German	20	700	300
Image	20	1300	1010
Waveform	21	400	4600
Splice	60	1000	2175

are the explanatory variables and the outcome instead of being real value is an integer which represents the class' label. The only modification needed to apply GP to classification problems is how to interpret the output of the individuals, because individuals in GP have real-valued outputs. Rounding the output or defining threshold values are the most common strategies.

5.1. Problems and Parameter Settings

The crossover operators presented in this work were tested on 13 different classification problems, presented in Figure 5.1. These problems were divided into two categories, depending on the number of features. Problems with less than 20 features were put into one set and problems with more than 20 features in the other. The measure of performance used to compare the different crossovers is the *Balanced Error Rate* (BER). Additionally, support vector machines (SVMs) were added into the comparison. The SVMs included linear and RBF kernels, with default parameters (the SVM implementations can be found in [Pedregosa11]).

The parameters used for the GP systems are the standard, suggested by [Poli08,

Table 5.2: Parameter Settings in GP Systems for Classification problems

Parameter	Value
Mutation Depth	random $\in [1, 5]$
Selection	Tournament size 4
Population Size	100
Number of generations	500
Function Set (F)	+,-,*,/,exp,sin,cos,ln abs, sqrt, sigmoid, if, max min, ln, square, argmax
Crossover rate	50 %
Mutation rate	50 %
Max length	min($\frac{\mathbb{T}}{2}$, 256)

Koza92]. Those parameters are shown in Table 5.2. Nonetheless, there are some parameters that deserve an explanation, for example the difference in the crossover rate and population size between classification and regression problems is because this set of parameters have been used with success in the past by [Vanneschi13]. Also, it was decided to use a different function set F for classification problems based on the performance exhibited by GP on classification problems (see [Valencia-Ramirez14]).

The function set F used in the classification problems is formed by arithmetic functions, transcendental functions and other not so common functions: *max*, *min*, *if* and *argmax*. These four latter functions are implemented using arithmetic operators and function *exp* see Equations 5.1-5.4. The if function is a sort of conditional function that selects y or z depending on whether the value of x is 0 or 1, respectively. Argmax function returns the index of the subtree that has the highest value.

$$max(x, y) = \frac{x - y}{1 + e^{-100(x-y)}} + y \tag{5.1}$$

$$min(x, y) = \frac{y - x}{1 + e^{-100(x-y)}} + x \tag{5.2}$$

$$if(x, y, z) = \frac{y - z}{1 + e^{-100x}} + z \tag{5.3}$$

$$argmax(x) = \sum_i \frac{e^{\beta x_i}}{\sum_j e^{\beta x_j}} i \tag{5.4}$$

Regarding the length of the individuals in the GP systems, a maximum length of $min(\frac{|\mathcal{T}|}{2}, 256)$ is imposed for classification. This value was inspired by the degrees of freedom in a function whose parameters can be linearly identified. This is, in order to identify k parameters it is needed at least $k+1$ points. Roughly, in an expression with n nodes at least $\frac{n}{2}$ of these nodes are operands and the other half are variables, so assuming that each variable has a coefficient to be identified one needs at least $\frac{n}{2}$ examples.

The last consideration is the procedure used to classify. Each classification problem was treated as a symbolic regression. In order to obtain a label from a continuous value, the output of the individual is rounded, and, the output was limited to be in the range $[0-1]$, given that all the datasets have only two classes. In addition to this, following ideas presented on [Valencia-Ramirez14], an ensemble of k classifiers is used. That is, each system is initialized k times, with k different seeds, and the best individual of each system is kept in a set. Then, the best individuals are used to predict the test set. The class of each object corresponds to the one that receives the greater number of votes. Finally, the number of examples are balanced to have exactly the same instances for both classes. This was achieved by removing the necessary examples in the training set until both classes have the same number of elements. The features of the datasets are normalized to have zero mean and one standard deviation.

Finally, the Classification problems used to test our approach correspond to different benchmarks found in the literature, the benchmarks and their information can be obtained in `http://www.raetschlab.org/Members/raetsch/benchmark`.

5.2. Results

The results obtained for the training set are shown in Table 5.3. Comparing GP systems against SVMs, the first trained better in 9 out of the 13 problems. It is important to note that the best results for GP were achieved by GPPDE, training better in 7 problems. SVM with RBF kernel was in second place with the best results in 4 problems. Unlike in symbolic regression problems, GPPDEC and GPPDE2 did not won any problem in the training set at all, although GPPDEC had close results with GPPDE in problems: Titanic,

Diabetes, Flare-solar and outperform it in Twonorm.

In the case of SVMs, it is important to note that the selection of the kernel makes a considerable difference in the performance, being RBF the kernel that presents better results. The difference is that SVMs with RBF kernel trained the best in problems: Diabetes, Ringnorm, German and Splice. SVMs with linear kernel trained the best in zero problems, in fact, SVMs with linear kernel were outperformed by GP systems in 12 problems.

For the test set, the results are depicted in Table 5.4. In the test set, the opposite thing occurred, SVMs validate better in 10 out of 13 problems against GP systems. The best results were achieved by SVM with a RBF kernel, validating the best in 9 problems. GPPDE was the second place validating the best in 2 problems.

Comparing GPPDE and GP, it is clear that including information of the derivatives in the crossover, helps to improve the performance of GP. Just like in the regression problems, different plots of the convergence of the GP methods have been included to see more clearly the advantages of the derivatives over traditional GP. Figures 5.3 and 5.4 show these plots. In these figures it can be seen that the training curve of GP does not learn as fast as GPPDE or GPPDEC, and for most of the classification problems, GP does not achieve the same performance either.

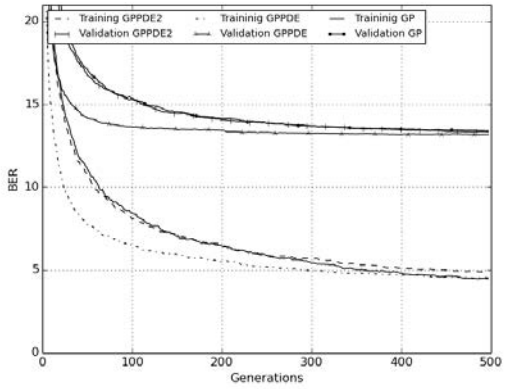
GPPDE is the clear winner for the GP systems in classification, in both training and tests sets. However, there are two problems that seem to be harder to generalize than the others for all the GP systems including GPPDE. These problems are Titanic and Breast-Cancer. In such graphics, the validation curves are noisy and the information learned in the training set is not useful to validate the problems. Something else worth to mention is how GPPDEC performs in the test set. The slight softener effect in the validation curves seen in symbolic regression problems is not present in classification.

Table 5.3: Performance of GP and SVM Systems on Classification Problems in Training Set. Best perfomance in boldface.

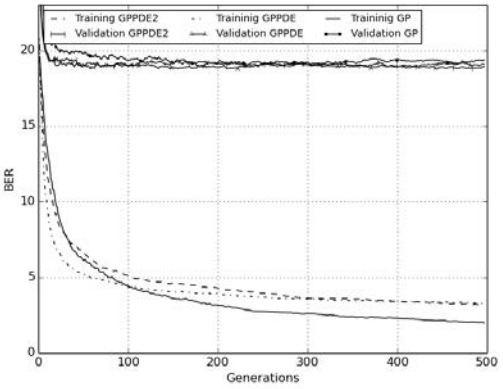
Problem	GPPDEC	GPPDE2	GPPDE	GP	SVM Linear	SVM RBF
Banana	4.8976 ± 1.6690	6.5953 ± 2.2780	4.5241 ± 1.6612	$\mathbf{4.4550 \pm 1.6100}$	44.7017	10.3337
Titanic	24.5118 ± 4.7161	$\mathbf{24.3845 \pm 5.1152}$	25.3618 ± 5.3653	24.4766 ± 4.8553	28.0303	26.7761
Thyroid	$\mathbf{0.0000 \pm 0.0000}$	0.2058 ± 0.5447	$\mathbf{0.0000 \pm 0.0000}$	$\mathbf{0.0000 \pm 0.0000}$	11.6121	3.3847
Diabetes	11.7787 ± 2.8863	13.3908 ± 3.6080	11.3865 ± 2.6753	$\mathbf{10.9930 \pm 2.9038}$	23.8205	17.0781
Breast-cancer	10.2325 ± 4.0565	13.1027 ± 4.9299	$\mathbf{7.8390 \pm 3.3359}$	8.8883 ± 3.6533	29.2102	18.8721
Flare-solar	27.2886 ± 2.0920	28.1160 ± 2.2980	$\mathbf{27.2879 \pm 1.7735}$	27.6304 ± 1.7388	31.9941	29.5441
Heart	3.1865 ± 1.8767	3.8543 ± 1.8566	3.2847 ± 1.7990	$\mathbf{1.9782 \pm 1.4423}$	12.9046	7.2450
Ringnorm	0.5477 ± 3.5798	0.3430 ± 0.4098	$\mathbf{0.0203 \pm 0.0992}$	0.1518 ± 0.4384	20.7716	0.3692
Twonorm	0.0756 ± 0.1786	0.0750 ± 0.1514	0.0681 ± 0.1380	$\mathbf{0.0261 \pm 0.1138}$	0.9521	0.4226
German	17.2762 ± 4.6591	20.0828 ± 4.7487	$\mathbf{14.5522 \pm 3.5633}$	16.7023 ± 4.4382	25.7178	**13.8590**
Image	2.5556 ± 1.7434	4.1390 ± 1.2309	$\mathbf{1.1680 \pm 0.4736}$	1.8036 ± 0.9888	16.5104	7.3035
Waveform	1.0913 ± 0.9628	1.6567 ± 1.1573	1.2049 ± 0.8462	$\mathbf{0.9836 \pm 1.1158}$	8.2755	4.0538
Splice	5.0507 ± 2.1862	6.3016 ± 2.2806	$\mathbf{3.8181 \pm 1.9212}$	4.6468 ± 1.6714	12.6342	**1.8736**

Table 5.4: Performance of GP and SVM Systems on Classification Problems in Validation Set. Best perfomance in boldface.

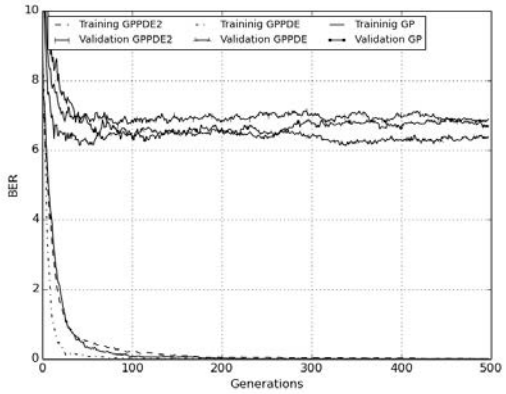
Problem	GPPDEC	GPPDE2	GPPDE	GP	SVM Linear	SVM RBF
Banana	13.3207 ± 1.3071	14.0086 ± 1.7875	13.1738 ± 1.4493	13.3993 ± 1.2938	46.3894	**11.7548**
Titanic	31.7737 ± 3.5047	31.9252 ± 3.4372	32.6755 ± 4.0815	32.2722 ± 3.7967	29.8575	**29.8437**
Thyroid	6.8695 ± 3.8875	6.4280 ± 3.4912	6.3888 ± 3.6412	6.7189 ± 3.2268	15.6444	**6.1575**
Diabetes	$\mathbf{26.2881 \pm 2.0697}$	26.7402 ± 2.3827	26.7969 ± 2.1747	26.9842 ± 2.0062	26.7470	26.9211
Breast-cancer	37.1104 ± 5.6245	36.7157 ± 5.5330	37.8349 ± 5.4793	36.7913 ± 5.2459	36.1889	**35.6376**
Flare-solar	33.3116 ± 1.9301	33.1648 ± 1.8204	33.2187 ± 1.7916	33.0648 ± 2.0894	32.9291	**32.5822**
Heart	18.8638 ± 3.6136	19.4722 ± 4.0256	19.1110 ± 3.9598	19.4058 ± 3.6437	**17.7606**	17.8571
Ringnorm	5.6080 ± 4.1133	5.9966 ± 1.5372	4.4865 ± 0.8190	6.2338 ± 2.7599	24.8896	**1.8325**
Twonorm	3.7455 ± 0.4606	3.7987 ± 0.5244	3.8085 ± 0.5512	3.7886 ± 0.4699	3.4861	**2.7158**
German	30.6172 ± 2.7325	30.5003 ± 2.5287	30.0875 ± 2.9162	30.8068 ± 2.8298	29.0928	**28.3244**
Image	5.4064 ± 2.1974	6.8151 ± 1.7931	$\mathbf{3.6739 \pm 0.5464}$	4.8303 ± 1.4477	16.9506	8.7921
Waveform	11.8906 ± 1.1360	12.0275 ± 1.1214	11.6855 ± 0.7742	12.3369 ± 1.1372	12.3945	**10.2681**
Splice	8.0533 ± 2.1862	9.5252 ± 2.2962	$\mathbf{7.5382 \pm 1.7733}$	8.3688 ± 1.5843	16.4734	11.4135



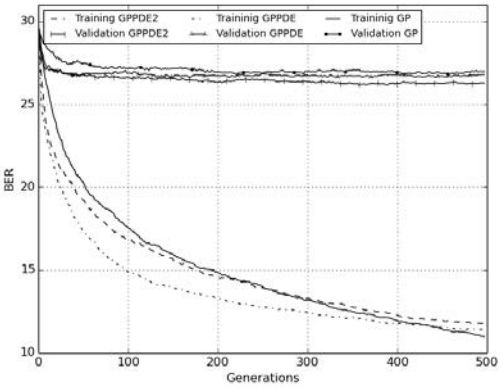
(a) Banana



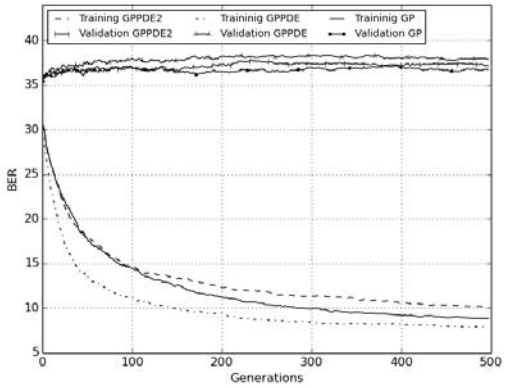
(b) Heart



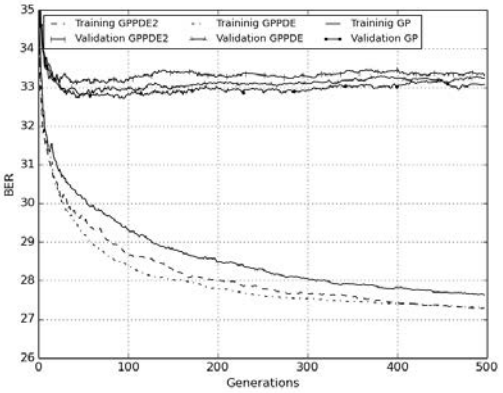
(c) Thyroid



(d) Diabetes

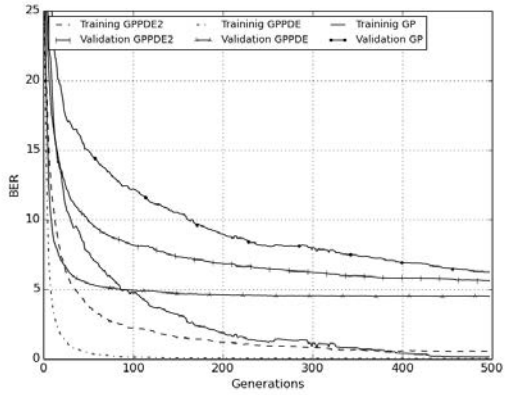


(e) Breast Cancer

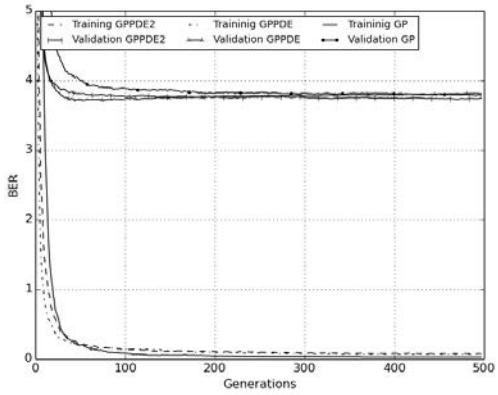


(f) Flare Solar

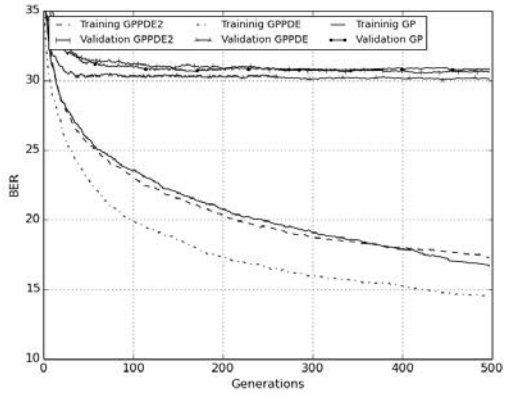
Figure 5.3: Classification Problems (low dimensionality)



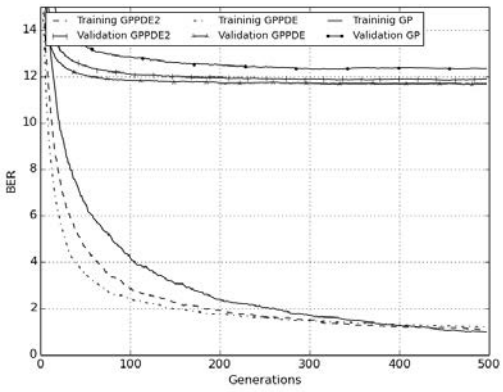
(a) Ringnorm



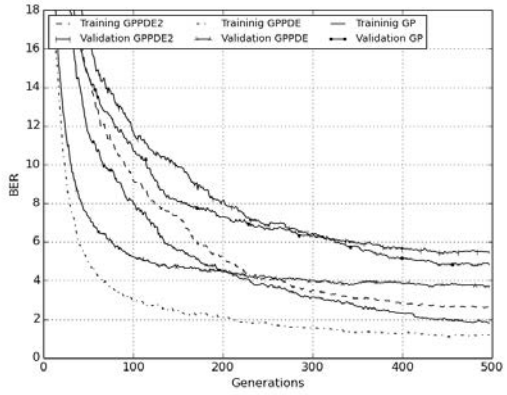
(b) Twonorm



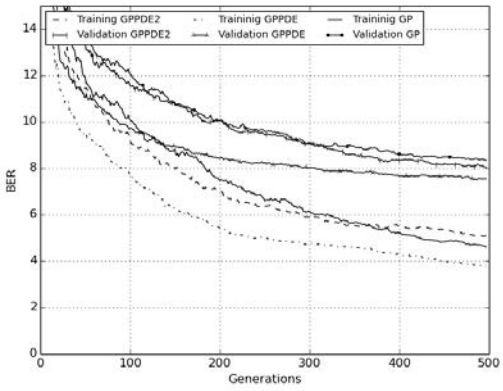
(c) German



(d) Waveform



(e) Image



(f) Splice

Figure 5.4: Classification Problems (high dimensionality)

5.3. Summary

In this chapter, our semantic crossover operators GPPDE, GPPDE2 and GPPDEC were tested over Classification problems. The classification problems used in the test correspond to a classical benchmark set, on which researches in a large number of areas test their classification models. Just like in the case of Regression, other methods have been included in the comparison to show how the performance of the semantic crossover operators can be compared with state of the art techniques. A Support Vector Machine (SVM) with two kernels was included in the comparison.SVM's are widely used methods for classification due to their high accuracy and because of this we decided to include them in the test.

The results for the semantic crossover operators proposed in this work, i.e., GPPDE, GPPDE2 and GPPDEC show an improvement over traditional syntactic crossover which can be seen again in the convergence plots that were depicted. Also, the performance is comparable against SVMs with RBM kernel and in most of the cases, superior than SVMs with linear kernel.

Chapter 6

Feature Selection and Semantic Crossover

When a system is being analyzed, it is common practice to collect data that will give us useful information for the analysis. In an ideal situation, only the data that best describes the system would be considered. Sometimes, this representative data is unknown. Making assumptions about it will lead to misunderstandings about the system.

In these cases the collected data contains both useful information and noise. The problem of *feature selection* [Kira92] consists in filtering the representative data contained in the data collected. Feature selection is needed to develop simpler models that will only consider relevant features. There are many problems that are related to feature selection: classification, clustering, regression, feature learning, online learning, etc.

Besides Classification and Regression, we have decided to test the proposed semantic operators and GP with feature selection problems. The motivation to do this is that, in [Suárez14], we compared traditional GP with techniques that are used for the feature selection problem like *LASSO*, *LARS* and, *Random Forests*.

Our results from [Suárez14], where we tested an ensemble of GP systems with syntactical crossover and other specialized algorithms on three different feature selection problems, proved that an ensenmble of GP is perfectly capable of performing Feature Selection over data on the run and without the need of any changes to the algorithm. Moreover, in some cases, the GP ensemble outperformed the other algorithms over certain criteria. In table 6.1, 6.2 and 6.3 the results for the test set over the problems considered are depicted.

Table 6.1: Results obtained by GP-E, RF and Lasso using a subset of Features for ARCENE's test dataset.

% Features	Accuracy		
	GP-E	RF	Lasso
100 %	0.7300	0.7400	0.6500
10 %	0.7300	0.7500	0.7700
5 %	**0.7900**	0.7200	0.7500
1 %	0.7300	0.7400	0.7700

Table 6.2: Results obtained by GP-E, RF and Lasso using a subset of Features for GISETTE's test dataset.

% Features	Accuracy		
	GP-E	RF	Lasso
100 %	0.9200	0.9530	0.8560
10 %	0.9250	0.9620	**0.9710**
5 %	0.9160	0.9570	0.9680
1 %	0.9150	0.9400	0.9640

Table 6.3: Results obtained by GP-E, RF and Lasso using a subset of Features for MADELON's test dataset.

% Features	Accuracy		
	GP-E	RF	Lasso
100 %	0.6300	0.8483	0.5767
10 %	0.7450	0.8617	0.5583
5 %	0.7867	**0.8733**	0.5750
1 %	0.6767	0.7717	0.5983

We now want to test if the proposed semantic crossover operators that use the partial derivatives, namely GPPDE, GPPDE2 and GPPDEC have an impact on the performance (wether this impact increase or decrease it) in Feature Selection problems with respect to GP.

6.1. GP as a Feature Selection Algorithm

GP is not often related to feature selection problems, however, GP performs feature extraction *on the run* without any modifications to the algorithm. The reason for this is explained by the way GP selects the problem's variables. GP creates a population of computer programs, which are created combining functions and terminals. The terminals are constants or problem's variables. These constants and variables are randomly selected every time GP creates a new individual. An individual may contain one or more variables more than once but it would be very strange for an individual to contain all the variables at least one time. Because of this, GP creates individuals that contain only a subset of the problem's variables.

In feature selection problems, the goal is to find a subset of variables that best describes the behavior of the system. When GP is applied to feature selection problems, the creation of individuals will consider only a subset of features. Once the evolution process has finished, the fittest individuals will contain only a subset of features. It can be inferred that this subset of features contains the features that affects the outcome to a greater degree . This is why GP is performing a feature extraction on the run.

Because the GP systems are being used for feature extraction, two steps are required. The first step consists of runing the GP systems to identify the most important features of the problem. This filtering of the features can be accomplished by inspecting the fittest individuals in the run. Because GP is a non-deterministic algorithm, one run of the system may be not enough to identify the most important variables.

We decided to perform 30 independent runs for the process of filtering. This is, at the end of each run, the best individual found is selected and the variables used in the model are identified. With this process, we can construct an histogram of variables, which counts how many times certain variable was used in the best individual of each run.

Once the most recurrent variables among the fittest individuals have been identified in the histogram, these variables are filtered from the original data, the second step is

performed and consists of a second run of the systems considering only this subset of variables.

In feature extraction, a common problem is to determine m, in other words, the number of features considered as *relevant* (ignoring the rest). All the algorithms try to minimize m without reducing the accuracy in the prediction. If the problem has many irrelevant features or noise, m can be approximately $10\,\% - 30\,\%$ of the total number of features. However, in order to not make any assumptions about the data, several values for m have been considered for the tests. These values start from as large as $50\,\%$ of the total features to as few as only $1\,\%$.

Summarizing, the methodology proposed for using GP for feature selection consists of two steps: the filtering of the features and the prediction of the classification problems. From the first 30 runs, the most relevant features are extracted. The second 30 runs considers only the features extracted in the first step and measures the performance of GP over the problem with the filtered features.

6.2. Problems and Parameter Settings

The GP-systems were tested over benchmark problems. These problems correspond to a contest in the area of feature selection that took place in 2003 (for the results of the competition see [Guyon04]). However, the system is still open for people who want to test their approaches over the problems (*http://www.nipsfsc.ecs.soton.ac.uk/*).

1.- ARCENE is a problem where the task is to identify prostate and ovarian cancer, the data was collected from two sources: The National Cancer Institute (NCI) and the Eastern Virginia Medical School (EVMS). The samples include patients with cancer and healthy or control patients. It is a binary classification problem with 10,000 features.

2.- GISETTE the task in this problem is to classify two confusable handwritten digits: the four and the nine. It is a binary classification problem with 5000 features.

3.- MADELON is a problem where random data has to be classified. The samples in the dataset are synthetic, it is a binary classification problem with 500 features.

The GP parameters used in the feature selection experiments are shown in Table 6.4. The election of these parameters gave good results in previous GP-related works [Graff13] and we decided to maintain some of them.

Table 6.4: GP Parameters for Feature Selection

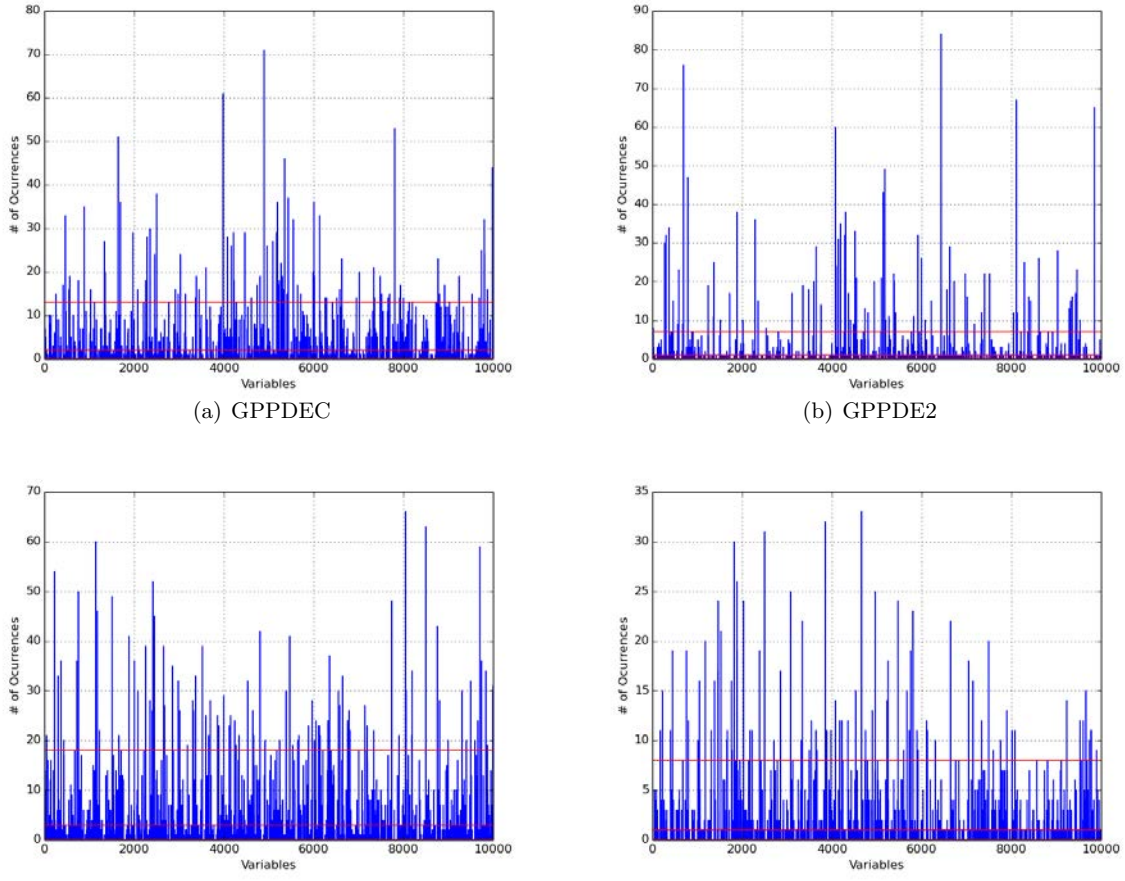
Parameter	Value
Population Size	1000
Number of Generations	50
Function Set ($\mathcal{F}$)	$\{+, -, \times, /, \lvert\cdot\rvert, \exp, \sqrt{}, \sin, \cos, \text{sigmoid}, \text{if}, \text{máx}, \text{mín}, \ln, square, \text{argmax}\}$
Crossover rate	90 %
Mutation rate	10 %
Mutation depth	random $\in [1, 5]$
Selection	Tournament of size 2

6.3. Results

Firt, let start discussing the histograms obtained by the GP systems for the three feature selection problems, Figures 6.1, 6.2 and 6.3 show these histograms. The histograms show that indeed, not all the features are selected by the best individuals the same number of times. The importance of the histograms rely on the data they generate.

With the histogram's data, the variables can be sorted by their importance, given the number of times that each variable was selected on the 30 runs. With this list of k features, a subset of m features is selected and these variables are the ones that can only be chosen by the GP system in the second run. It is worth noticing that each GP system performs its own filtering of features, in other words, all the algorithms performs the filtering step and the classification step separately from the others.

For example in the ARCENE problem, depicted in Figure 6.1, it can be seen that in GPPDEC, GPPDE2 and GPPDE at least three variables were selected more than 60 times by the best individual in the 30 runs, whereas in GP none of the variables were selected more than 35 times.



(a) GPPDEC (b) GPPDE2

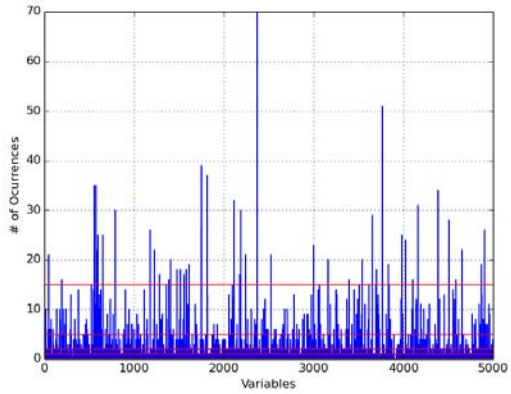
(c) GPPDE (d) GP

Figure 6.1: Histogram of features in ARCENE dataset

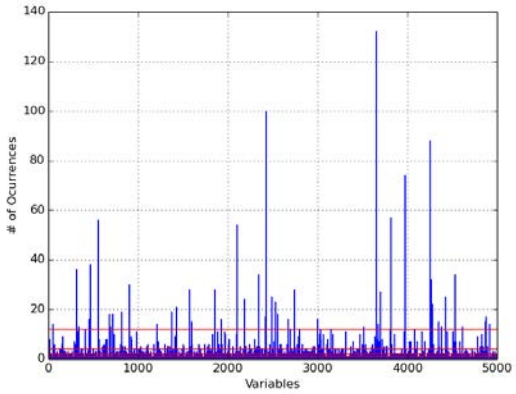
For GISETTE problem, Figure 6.2, GPPDE2 and GPPDE selected variables more than 100 times in constrast to GPPDEC and GP where the maximum number of ocurrences for the variables nevar passed 70 for GPPDEC and 50 for GP.

Finally, in the histograms for MADELON problem, shown in Figure 6.3, GPPDEC and GPPDE selected variables with more than 200 ocurrences, GPPDE2 selected variables with more than 150 ocurrences and for GP, again, the selection of any variable did not pass above 120 ocurrences.

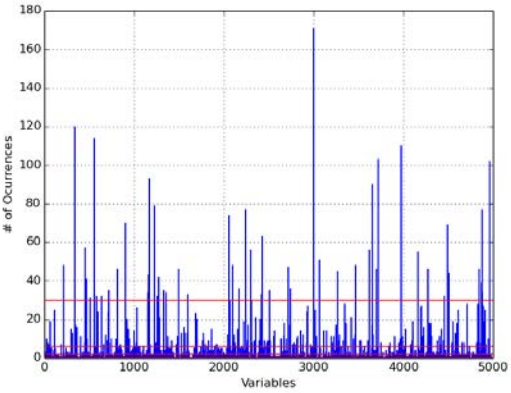
In order to test how these different outcomes in the filtering of variables would perfom in the classification task, we decided to test different values for m (the number



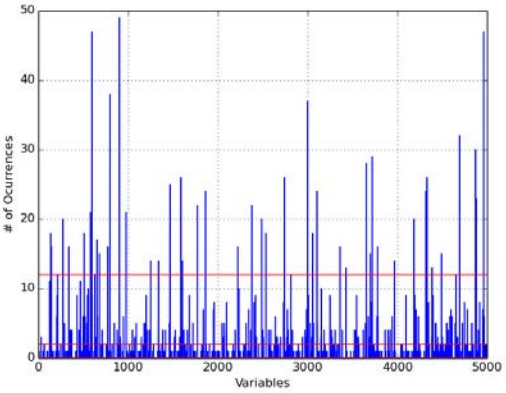
(a) GPPDEC



(b) GPPDE2



(c) GPPDE



(d) GP

Figure 6.2: Histogram of features in GISETTE dataset

of filtered features) to depict how this value afects the performance. In the plots, these different values of m are represented by the red lines. Each line represents a different value for $m = [1, 5, 10, 25, 50]$, where the values for m represent the percentage of the total features that are selected. For a given line, the features who are above the line are considered for selection. So, each line represent a cut in the features' space. These cuts are more evident in GISETTE and MADELON problems.

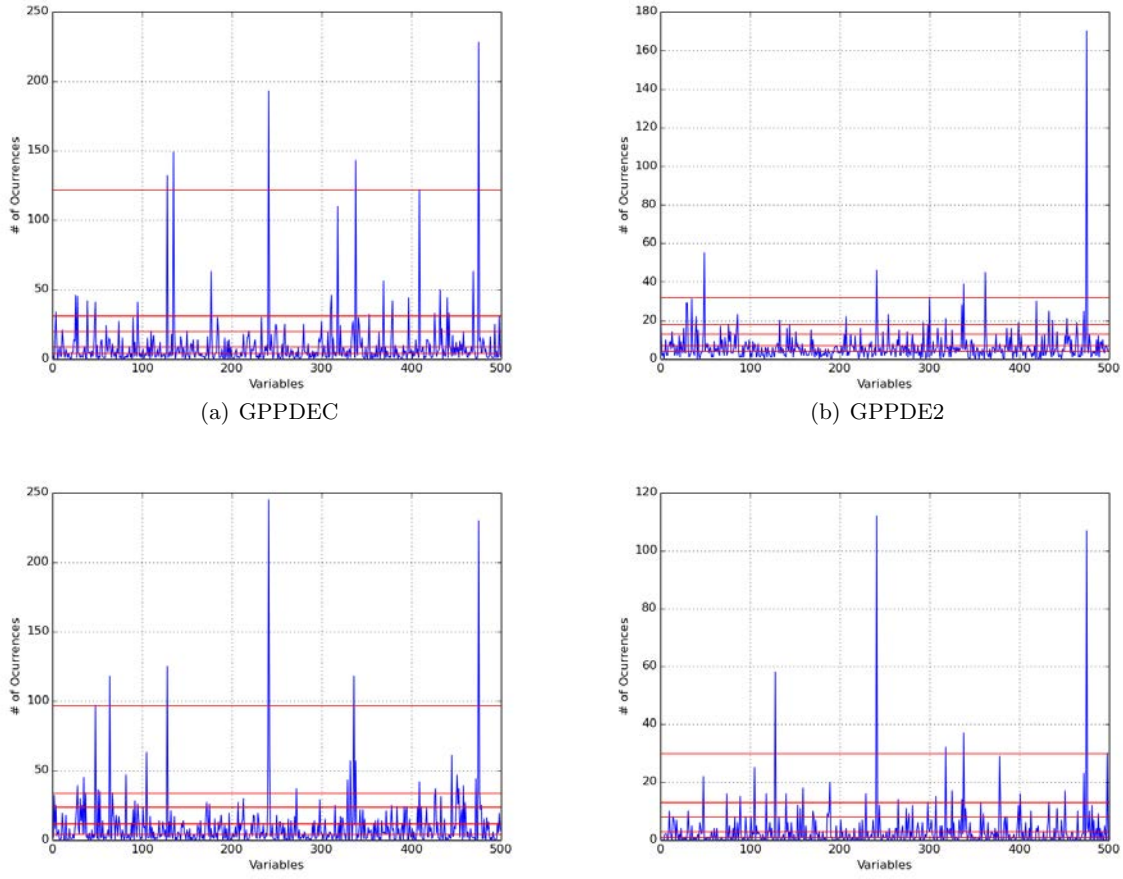
When the process of the selection of variables had concluded, the next step is test these different subset of features in the classification problem. These results for the classification task are given in terms of the accuracy, this is, the number of success ratio in the classification ($\frac{\#ofcorrectpredictions}{\#totalexamples}$). The last consideration regarding to the tests is that *Principal Component Analysis* (PCA) [Holland08] was added to the comparison to reduce the data dimensionality by the selection of different values for m and then use least squares to perform a linear classification over the data. Such scheme will be named PCA-LS. meaning that PCA is used to select a subset of features and Least Squares is used to perform the classification.

Let us start first with ARCENE in both Training and Validation sets. Tables 6.5 and 6.6 show the results. The tables present five different values for m which are: $[50, 25, 10, 5, 1]$ % of the total of features and the best result overall the values of m is in boldface. In the majority of the algorithms, one can see a very distinctive pattern in GP systems: the least the features used, the greater accuracy reached.

For example in Figure 6.5 as for GPPDE is concerned, the best training accuracy was obtained at 5 % of the features, this is 500 features instead of the original $10,000$ features. This behavior is also present in GPPDEC and GP, where their best accuracy was reached with 1 % of the features. PCA-LS manages to train perfectly for the different values m values.

For the Validation set in ARCENE dataset the best accuracy for GP systems was obtained by GPPDEC using 1 % of the features. Interestingly, GP decreased its accuracy when the number of features decreased, perhaps GP is overfitting in this problem. A little overfitting can be observed in GPPDE too. In the case of GPPDEC, the best accuracy in both training and validation sets was at 1 % of the features. Again, PCA-LS obtains the best validation results with an acuraccy of 0.8300 for $m = 25$ %.

The results of GISETTE dataset are presented in Tables 6.7 and 6.8. For the training set, the best result for the GP-systems was acomplished by GPPDE at 1 % of



(a) GPPDEC (b) GPPDE2

(c) GPPDE (d) GP

Figure 6.3: Histogram of features in MADELON dataset

features, followed by GPPDE2 at the same % of features. PCA-LS trains with no error for this problem too. The results in the validation set, presented in Figure 6.8 show that GP, GPDPE1 and GPPDE reach their maximum accuracy at %1 of features, whereas GPPDEC had a slightly drecrease in accuracy from 0.8919 at 5 % of features to 0.8883 at 1 % of features. The best result was accomplished by GPPDE. In the validation set all the GP-systems outperformed PCA-LS with which had a maximum accuracy of 0.8030 at $m = 25\,\%$.

Finally, for the MADELON dataset the results are presented in Tables 6.9 and 6.10. In the training set, the best result was reached by GPPDE but this time at 10 % of the features, contrarily to GPPDEC, GPPDE2 and GP who reach ther best values at 5 % and 1 % of features. PCA-LS reach the best training accuracy of 0.8305 at 50 % of the features.

In the validation set there is a tie in the best result with GPPDE2 and GPPDE at 5 % of the features. It is important to note that at 1 % of the features, contrarily to ARCENE and GISETTE, GP achieves its best performance compared to other values of m. PCA-LS had poor validation results and perhaps is overfitting the data.

Additionally, we have included the official results of the feature selection competition from which the datasets were collected, namely *NIPS 2003 Feature Selection Challenge* [Guyon05]. According to the authors, several methods were submitted to the competition but three categories were the more common: SVMs, ANNs and Tree methods (Decision trees and Random Forests).

Tables 6.11 and 6.12 show the results obtained for the methods submitted at December 1st 2003 and December 8th. 2003 respectively. The results presented in such tables are the result of the evaluation of the submitted methods by the organizers and are presented for all the problems in the challenge. Unfortunately, the results are not problem by problem and we can not compare to them but they give an illustration of what the performance is like for other techniques. In the results tables, the score is the performance achieved by the methods and the percentage of features used to achieved such performance is also reported.

Table 6.5: Training results for the ARCENE dataset

% of Features	GPPDEC	GPPDE2	GPPDE	GP	PCA-LS
50	0.8796 ± 0.0521	0.9080 ± 0.0454	0.9640 ± 0.0354	0.8693 ± 0.0611	**1**
25	0.9086 ± 0.0413	0.8830 ± 0.0469	0.9753 ± 0.0192	0.8996 ± 0.0402	**1**
10	0.8973 ± 0.0481	0.9033 ± 0.0560	0.9503 ± 0.0500	0.8863 ± 0.0477	**1**
5	0.9053 ± 0.0393	0.9403 ± 0.0369	0.9810 ± 0.0210	0.8993 ± 0.0403	**1**
1	0.9100 ± 0.0371	0.9110 ± 0.0419	0.9756 ± 0.0329	0.9163 ± 0.0427	**1**

Table 6.6: Validation results for the ARCENE dataset

% of Features	GPPDEC	GPPDE2	GPPDE	GP	PCA-LS
50	0.6553 ± 0.0791	0.5453 ± 0.0513	0.6616 ± 0.0485	0.6623 ± 0.0530	0.8200
25	0.6716 ± 0.0602	0.6260 ± 0.0679	0.6650 ± 0.0667	0.6633 ± 0.0639	**0.8300**
10	0.6546 ± 0.0627	0.6663 ± 0.0718	0.6660 ± 0.0539	0.6503 ± 0.0409	0.8100
5	0.6850 ± 0.0484	0.6520 ± 0.0577	0.6433 ± 0.0550	0.6613 ± 0.0453	0.7800
1	0.6883 ± 0.0579	0.6846 ± 0.0535	0.6496 ± 0.0502	0.6176 ± 0.0767	0.7300

Table 6.7: Training results for the GISETTE dataset

% of Features	GPPDEC	GPPDE2	GPPDE	GP	PCA-LS
50	0.8480 ± 0.0604	0.8738 ± 0.0638	0.8946 ± 0.0438	0.8539 ± 0.0264	**1**
25	0.8192 ± 0.0547	0.8922 ± 0.0167	0.8936 ± 0.0338	0.8648 ± 0.0177	**1**
10	0.8424 ± 0.0437	0.8892 ± 0.0281	0.8976 ± 0.0277	0.8818 ± 0.0196	**1**
5	0.8919 ± 0.0486	0.8932 ± 0.0208	0.9065 ± 0.0332	0.8903 ± 0.0154	**1**
1	0.8883 ± 0.0245	0.9049 ± 0.0124	0.9255 ± 0.0188	0.8992 ± 0.0085	**1**

Table 6.8: Validation results for the GISETTE dataset

% of Features	GPPDEC	GPPDE2	GPPDE	GP	PCA-LS
50	0.8344 ± 0.0559	0.8599 ± 0.0361	0.8750 ± 0.0398	0.8445 ± 0.0261	0.7750
25	0.8109 ± 0.0495	0.8777 ± 0.0161	0.8742 ± 0.0317	0.8533 ± 0.0174	0.7890
10	0.8303 ± 0.0433	0.8728 ± 0.0246	0.8787 ± 0.0286	0.8697 ± 0.0223	0.8030
5	0.8678 ± 0.0451	0.8812 ± 0.0232	0.8868 ± 0.0313	0.8816 ± 0.0161	0.7710
1	0.8747 ± 0.0221	0.8923 ± 0.0117	$\mathbf{0.9073 \pm 0.0187}$	0.8770 ± 0.0094	0.7780

Table 6.9: Training results for the MADELON dataset

% of Features	GPPDEC	GPPDE2	GPPDE	GP	PCA-LS
50	0.5807 ± 0.0571	0.6361 ± 0.0246	0.6225 ± 0.0682	0.6244 ± 0.0287	**0.8305**
25	0.6175 ± 0.0514	0.6281 ± 0.0158	0.6474 ± 0.0627	0.6371 ± 0.0560	0.8275
10	0.6182 ± 0.0550	0.6329 ± 0.0243	0.7021 ± 0.0895	0.6554 ± 0.0479	0.7275
5	0.6354 ± 0.0531	0.6464 ± 0.0291	0.6945 ± 0.0763	0.6817 ± 0.0601	0.6755
1	0.6480 ± 0.0581	0.6408 ± 0.0302	0.6681 ± 0.0391	0.6690 ± 0.0312	0.6235

Table 6.10: Validation results for the MADELON dataset

% of Features	GPPDEC	GPPDE2	GPPDE	GP	PCA-LS
50	0.5596 ± 0.0579	0.6105 ± 0.0288	0.6030 ± 0.0541	0.6198 ± 0.0287	0.4500
25	0.6036 ± 0.0483	0.6248 ± 0.0106	0.6227 ± 0.0533	0.6334 ± 0.0578	0.5100
10	0.6058 ± 0.0487	0.6258 ± 0.0203	0.6685 ± 0.0789	0.6512 ± 0.0512	0.4850
5	0.6247 ± 0.0442	$\mathbf{0.6881 \pm 0.0238}$	$\mathbf{0.6881 \pm 0.0565}$	0.6865 ± 0.0655	0.4766
1	0.6374 ± 0.0467	0.6366 ± 0.0261	0.6490 ± 0.0223	0.6631 ± 0.0294	0.4850

Table 6.11: Results from NIPS 2003 competition (December 1st.)

Method (Team)	Score	Features (%)
BayesNN-DFT (Neal/Zhang)	88.0	80.3
BayesNN-DFT (Neal/Zhang)	86.2	80.3
BayesNN-small (Neal)	68.7	4.7
BayesNN-large (Neal)	59.6	60.3
RF+RLSC (Torkkola/Tuv)	59.3	22.5
final2 (Chen)	52.0	24.9
SVMBased3 (Zhili/Li)	41.8	29.5
SVMBased4 (Zhili/Li)	41.1	29.5
final1 (Chen)	40.4	6.2
transSVM2 (Zhili)	36.0	29.5
BayesNN-E (Neal)	29.5	96.8
Collection2 (Saffari)	28.0	7.7
Collection1 (Saffari)	20.7	32.3

6.4. Summary

The main objective of this chapter was to compare GP to GPPDEC, GPPDE2 and GPPDE for feature selection problems. The importance of Feature selection is that it helps to filter the important features, therefore, saving computing time and storage space. At the same time it helps to increase the algorithm's performance because it removes features that do not provide problem's information, this non important features only provide noise, and this noise decreases the performance of the algorithm.

In this chapter, different GP systems were used to perform feature selection over a set of benchmarks problems. Feature Selection is not a problem on which GP is applied very often even though that due to the form that GP creates individuals, GP performs a feature selection by on the run. The performance of the different GP systems in feature selection problems were compared to each other.

We had previously tested an ensemble of GP for feature selection problems and compared to LASSO and Random Forest obtaining good results. This time, the results were compared to PCA and Least Squares obtaining better results in 2 out of 3 problems. The results presented here show that the semantic operators proposed in this work extend

Table 6.12: Results from NIPS 2003 competition December 8th.

Method (Team)	Score	Features (%)
BayesNN-DFT (Neal/Zhang)	71.4	80.3
BayesNN-large (Neal)	66.3	60.3
BayesNN-small (Neal)	61.1	4.7
final 2-3 (Chen)	49.1	24.9
BayesNN-large (Neal)	49.1	60.3
final2-2 (Chen)	40.0	24.6
Ghostminer1 (Ghostminer)	37.1	80.6
RF+RLSC (Torkkola/Tuv)	35.4	22.4
Ghostminer2 (Ghostminer)	35.4	80.6
RF+RLSC (Torkkola/Tuv)	34.3	22.4
FS+SVM (Lal)	31.4	20.9
Ghostminer3 (Ghostminer)	26.3	80.6
CBAMethod3E (CBAGroup)	21.1	12.8
Nameless (Navot/Bachrach)	12.0	32.3

the capabilities of GP for feature selection, in terms of achieving better performance or reducing overfitting. The results of the original Feature Selection competition from which the problems were taken were also reported as an illustration of the performance of other methods in such task.

Chapter 7

Conclusions and Future Work

In this work, a methodology to compute the first and second partial derivatives in GP have been introduced. Particularly, the derivative of the fitness function *w.r.t.* some node. If these derivatives are computed for the node selected as the crossing point, this opens the possibility to develop a crossover operator for GP. Moreover, this operator would be semantic, because the crossover between individuals is being driven by their behavior rather than their syntax.

Based on this methodology, three semantic operators have been created: GPPDE, GPPDE2 and GPPDEC. The differences between these three operators are the degree of the derivative and how to interpret the derivative. GPPDE computes the first partial derivative and uses the sign of this derivative to select the crossing point in the second parent. GPPDE2 computes the first and second partial derivatives and performs an iteration of the newton method and uses this information to perform the crossover. Finally, GPPDEC is a combination of the later two crossovers and checks the sign of the first derivative with the information of the first iteration of the newton method.

These three crossover operators were tested on two classes of problems commonly tackled with GP: Symbolic Regression and Classification. For Symbolic Regression, results show that a combination of the information provided by the first and second derivatives, this is, GPPDEC performed the best of all the three methods. On the other hand, in Classification problems the method that proved to be the best approach was GPPDE. In both classes of problems, the semantic operators created outperformed the traditional syntactical operator used in GP. Different convergence plots were provided to illustrate the behavior of the semantic operators throughout the evolutionary process in order to prove

that they are a better strategy than a syntactic operator.

Additionally, we have included state of the art methods for all the problems tested to compare the performance of the semantic operators presented in this work. For Symbolic Regression, a comparison with other semantical crossover operators found in the literature was presented and for Classification SVMs were included to the comparison. In the first problem, results show that GPPDEC was the overall second place out of the 6 methods included. And for Classification, GPPDE proved to be a competitive strategy against SVMs and even achieving better results than SVMs depending on the kernel used.

Moreover, another class of problems was included in this work: Feature Selection. In previous work we have tested GP over Feature Selection problems with interesting results. This time GPPDE, GPPDE2, GPPDEC were compared against GP for Feature Selection problems. In order to have a reference in this kind of problems, PCA with linear squares were included into the comparison. Results for Feature Selection show that GP-systems recognize important features and improve the performance of the classification task. Again, GPPDE, GPPDE2 and GPPDEC outperformed the syntactical crossover. The problems used in the Feature Selection task were obtained from the NIPS 2003 Feature Selection Challenge, the results from the original competition were also depicted for illustrative purposes.

The results obtained in this work encourage us to perform a deeper research about how to interpret partial derivatives in GP in order to develop better crossover operators than the traditional syntactic ones and, at the same time, to provide new strategies to the semantic ones.

Future work can be developed in the area of mutation. Further research can include a mutation operator guided by the partial derivatives of 1st. and 2nd. degree like GPPDE and GPPDE2 and a combination of both, like GPPDEC. The idea will be the same on which these semantic operators rely on: compute the desired partial derivative in the mutation point and with this information search in the possible mutations which one resembles the most with the heuristic being used.

Another idea that can be implemented in short time is the fine tuning of constants in the individuals. The idea here is that at the end of the GP run, take the best individual and tune some of its constants with the Newton method (like in Section 3.2) and try to boost the performance of the best individual in the run.

Lastly, regarding Feature Selection and GP and given the results, another work that can be easily implemented is the combination of GP with other techniques to solve the

task. This is, GP will be used only in the selection of the subset of features that are going to be used in the classification task and the, with this reduction of data's dimensionality, perform the classification with other methods like SVMs that show great generalization ability in the Feature selection results.

Finally, all these last ideas only confirm that there is so much area of improvement regarding to GP and GP with the crossovers presented in this work. We believe that GPPDE, GPPDE2 and GPPDEC are one successful example of the combination of traditional techniques like Newton Method with EAs like GP will lead to better strategies to solve real-world problems.

References

[Augusto00] Augusto, D. A. y Barbosa, H. J. Symbolic regression via genetic programming. *En Neural Networks, 2000. Proceedings. Sixth Brazilian Symposium on*, págs. 173–178. IEEE, 2000.

[Beadle08] Beadle, L. y Johnson, C. G. Semantically driven crossover in genetic programming. *En IEEE Congress on Evolutionary Computation*, págs. 111–116. 2008.

[Beadle09] Beadle, L. y Johnson, C. G. Semantic analysis of program initialisation in genetic programming. *Genetic Programming and Evolvable Machines*, 10(3):307–337, 2009.

[Blickle94] Blickle, T. y Thiele, L. Genetic programming and redundancy. *choice*, 1000:2, 1994.

[Bojarczuk00] Bojarczuk, C. C., Lopes, H. S., y Freitas, A. A. Genetic programming for knowledge discovery in chest-pain diagnosis. *IEEE Engineering in Medicine and Biology Magazine*, 19(4):38–44, 2000.

[Bot00a] Bot, M. C. Improving induction of linear classification trees with genetic programming. *En Proceedings of the 2nd Annual Conference on Genetic and Evolutionary Computation*, págs. 403–410. Morgan Kaufmann Publishers Inc., 2000.

[Bot00b] Bot, M. C. y Langdon, W. B. Application of genetic programming to induction of linear classification trees. *En European Conference on Genetic Programming*, págs. 247–258. Springer, 2000.

[Brameier01] Brameier, M. y Banzhaf, W. Evolving teams of predictors with linear genetic programming. *Genetic Programming and Evolvable Machines*, 2(4):381–407, 2001.

[Carreno07] Carreno, E., Leguizamón, G., y Wagner, N. Evolution of classification rules for comprehensible knowledge discovery. *En Evolutionary Computation, 2007. CEC 2007. IEEE Congress on*, págs. 1261–1268. IEEE, 2007.

[Carse07] Carse, B. y Pipe, A. G. Introduction: genetic fuzzy systems. *International Journal of Intelligent Systems*, 22(9):905–907, 2007.

[Casillas09] Casillas, J. y Carse, B. Special issue on genetic fuzzy systems: Recent developments and future directions. *Soft Computing-A Fusion of Foundations, Methodologies and Applications*, 13(5):417–418, 2009.

[Cavaretta99] Cavaretta, M. J. y Chellapilla, K. Data mining using genetic programming: The implications of parsimony on generalization error. *En Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*, tomo 2, págs. 1330–1337. IEEE, 1999.

[Chen07] Chen, Z. y Lu, S. A genetic programming approach for classification of textures based on wavelet analysis. *En Intelligent Signal Processing, 2007. WISP 2007. IEEE International Symposium on*, págs. 1–6. IEEE, 2007.

[Chien02] Chien, B.-C., Lin, J. Y., y Hong, T.-P. Learning discriminant functions with fuzzy attributes for classification using genetic programming. *Expert Systems with Applications*, 23(1):31–37, 2002.

[Curry07] Curry, R., Lichodzijewski, P., y Heywood, M. I. Scaling genetic programming to large datasets using hierarchical dynamic subset selection. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 37(4):1065–1073, 2007.

[De Falco02] De Falco, I., Della Cioppa, A., y Tarantino, E. Discovering inter-

esting classification rules with genetic programming. *Applied Soft Computing*, 1(4):257–269, 2002.

[De Stefano02] De Stefano, C., Della Cioppa, A., y Marcelli, A. Character preclassification based on genetic programming. *Pattern Recognition Letters*, 23(12):1439–1448, 2002.

[Eggermont99] Eggermont, J., Eiben, A. E., y van Hemert, J. I. A comparison of genetic programming variants for data classification. *En International Symposium on Intelligent Data Analysis*, págs. 281–290. Springer, 1999.

[Eggermont02] Eggermont, J. Evolving fuzzy decision trees with genetic programming and clustering. *En European Conference on Genetic Programming*, págs. 71–82. Springer, 2002.

[Espejo05] Espejo, P. G., Romero, C., Ventura, S., y Hervás, C. Induction of classification rules with grammar-based genetic programming. *En Conference on Machine Intelligence*, págs. 596–601. 2005.

[Espejo10] Espejo, P. G., Ventura, S., y Herrera, F. A survey on the application of genetic programming to classification. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 40(2):121–144, 2010.

[Estébanez05] Estébanez, C., Valls, J., Aler, R., y Galván, I. A first attempt at constructing genetic programming expressions for eeg classification. *Artificial Neural Networks: Biological Inspirations–ICANN 2005*, págs. 665–670, 2005.

[Estébanez08] Estébanez, C., Valls, J. M., y Aler, R. Gppe: a method to generate ad-hoc feature extractors for prediction in financial domains. *Applied Intelligence*, 29(2):174–185, 2008.

[Estrada-Gil07] Estrada-Gil, J. K., Fernández-López, J. C., Hernández-Lemus, E., Silva-Zolezzi, I., Hidalgo-Miranda, A., Jiménez-Sánchez, G., y Vallejo-Clemente, E. E. Gpdti: A genetic programming decision tree induc-

tion method to find epistatic effects in common complex diseases. *Bioinformatics*, 23(13):i167–i174, 2007.

[Faraoun06] Faraoun, K. y Boukelif, A. Genetic programming approach for multi-category pattern classification applied to network intrusions detection. *International Journal of Computational Intelligence and Applications*, 6(01):77–99, 2006.

[Fogel97] Fogel, D. B. The advantages of evolutionary computation. *En BCEC*, págs. 1–11. 1997.

[Folino99] Folino, G., Pizzuti, C., y Spezzano, G. A cellular genetic programming approach to classification. *En Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation-Volume 2*, págs. 1015–1020. Morgan Kaufmann Publishers Inc., 1999.

[Folino00] Folino, G., Pizzuti, C., y Spezzano, G. Genetic programming and simulated annealing: A hybrid method to evolve decision trees. *En European Conference on Genetic Programming*, págs. 294–303. Springer, 2000.

[Folino08] Folino, G., Pizzuti, C., y Spezzano, G. Training distributed gp ensemble with a selective algorithm based on clustering and pruning for pattern classification. *IEEE Transactions on Evolutionary Computation*, 12(4):458–468, 2008.

[Garcia-Almanza08] Garcia-Almanza, A. L. y Tsang, E. P. Evolving decision rules to predict investment opportunities. *International Journal of Automation and Computing*, 5(1):22–31, 2008.

[Graff13] Graff, M., Pena, R., y Medina, A. Wind speed forecasting using genetic programming. *En 2013 IEEE Congress on Evolutionary Computation*, págs. 408–415. IEEE, 2013.

[Graff14] Graff, M., Graff-Guerrero, A., y Cerda-Jacobo, J. Semantic crossover based on the partial derivative error. *En European Conference on Genetic Programming*, págs. 37–47. Springer, 2014.

[Graff15] Graff, M., Tellez, E. S., Villasenor, E., y Miranda-Jiménez, S. Semantic genetic programming operators based on projections in the phenotype space. *Research in Computing Science*, 94:73–85, 2015.

[Guo05] Guo, H., Jack, L. B., y Nandi, A. K. Feature generation using genetic programming with application to fault classification. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 35(1):89–99, 2005.

[Guo06] Guo, H. y Nandi, A. K. Breast cancer diagnosis using genetic programming generated feature. *Pattern Recognition*, 39(5):980–987, 2006.

[Guyon04] Guyon, I., Gunn, S., Ben-Hur, A., y Dror, G. Result analysis of the nips 2003 feature selection challenge. *En Advances in Neural Information Processing Systems*, págs. 545–552. 2004.

[Guyon05] Guyon, I., Gunn, S., Ben-Hur, A., y Dror, G. Result analysis of the nips 2003 feature selection challenge. *En Advances in neural information processing systems*, págs. 545–552. 2005.

[Haruyama02] Haruyama, S. y Zhao, Q. Designing smaller decision trees using multiple objective optimization based gps. *En Systems, Man and Cybernetics, 2002 IEEE International Conference on*, tomo 6, págs. 5–pp. IEEE, 2002.

[Hengpraprohm08] Hengpraprohm, S. y Chongstitvatana, P. A genetic programming ensemble approach to cancer microarray data classification. *En Innovative Computing Information and Control, 2008. ICICIC'08. 3rd International Conference on*, págs. 340–340. IEEE, 2008.

[Hennessy05] Hennessy, K., Madden, M. G., Conroy, J., y Ryder, A. G. An improved genetic programming technique for the classification of raman spectra. *Knowledge-Based Systems*, 18(4):217–224, 2005.

[Hochreiter01] Hochreiter, S., Bengio, Y., Frasconi, P., y Schmidhuber, J. Gradient

flow in recurrent nets: the difficulty of learning long-term dependencies, 2001.

[Holland08] Holland, S. M. Principal components analysis (pca). *Department of Geology, University of Georgia, Athens, GA*, págs. 30602–2501, 2008.

[Hong06] Hong, J.-H. y Cho, S.-B. The classification of cancer based on dna microarray data that uses diverse ensemble genetic programming. *Artificial intelligence in Medicine*, 36(1):43–58, 2006.

[Imamura03] Imamura, K., Soule, T., Heckendorn, R. B., y Foster, J. A. Behavioral diversity and a probabilistically optimal gp ensemble. *Genetic Programming and Evolvable Machines*, 4(3):235–253, 2003.

[Johnson00] Johnson, H. E., Gilbert, R. J., Winson, M. K., Goodacre, R., Smith, A. R., Rowland, J. J., Hall, M. A., y Kell, D. B. Explanatory analysis of the metabolome using genetic programming of simple, interpretable rules. *Genetic Programming and Evolvable Machines*, 1(3):243–258, 2000.

[Khoshgoftaar07] Khoshgoftaar, T. M. y Liu, Y. A multi-objective software quality classification model using genetic programming. *IEEE Transactions on Reliability*, 56(2):237–245, 2007.

[Kira92] Kira, K. y Rendell, L. A. The feature selection problem: Traditional methods and a new algorithm. *En AAAI*, págs. 129–134. 1992.

[Koza92] Koza, J. R. *Genetic programming: on the programming of computers by means of natural selection*, tomo 1. MIT press, 1992.

[Krawiec02] Krawiec, K. Genetic programming-based construction of features for machine learning and knowledge discovery tasks. *Genetic Programming and Evolvable Machines*, 3(4):329–343, 2002.

[Krawiec09] Krawiec, K. y Lichocki, P. Approximating geometric crossover in semantic space. *En Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, GECCO '09, págs. 987–994.

ACM, New York, NY, USA, 2009. ISBN 978-1-60558-325-9. doi: 10.1145/1569901.1570036. 00032.
URL `http://doi.acm.org/10.1145/1569901.1570036`

[Krawiec12] Krawiec, K. y Pawlak, T. Locally Geometric Semantic Crossover. *En Proceedings of the 14th Annual Conference Companion on Genetic and Evolutionary Computation*, GECCO '12, págs. 1487–1488. ACM, New York, NY, USA, 2012. ISBN 978-1-4503-1178-6. doi:10.1145/2330784.2331005. 00014.
URL `http://doi.acm.org/10.1145/2330784.2331005`

[Kuo07] Kuo, C.-S., Hong, T.-P., y Chen, C.-L. Applying genetic programming technique in classification trees. *Soft Computing-A Fusion of Foundations, Methodologies and Applications*, 11(12):1165–1172, 2007.

[Li07] Li, Y.-M., Wang, M., Cui, L.-J., y Huang, D.-M. A new classification arithmetic for multi-image classification in genetic programming. *En Machine Learning and Cybernetics, 2007 International Conference on*, tomo 3, págs. 1683–1687. IEEE, 2007.

[Li08] Li, G., Wang, J. F., Lee, K. H., y Leung, K.-S. Instruction-matrix-based genetic programming. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 38(4):1036–1049, 2008.

[Lin05] Lin, Y. y Bhanu, B. Evolutionary feature synthesis for object recognition. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 35(2):156–171, 2005.

[Moraglio12a] Moraglio, A., Krawiec, K., y Johnson, C. G. Geometric semantic genetic programming. *En International Conference on Parallel Problem Solving from Nature*, págs. 21–31. Springer, 2012.

[Moraglio12b] Moraglio, A., Krawiec, K., y Johnson, C. G. Geometric semantic genetic programming. *En* C. A. C. Coello, V. Cutello, K. Deb, S. Forrest, G. Nicosia, y M. Pavone, eds., *Parallel Problem Solving from Nature - PPSN XII*, nº 7491 en Lecture Notes in Computer Science,

págs. 21–31. Springer Berlin Heidelberg, ene. 2012. ISBN 978-3-642-32936-4, 978-3-642-32937-1.

[Mugambi04] Mugambi, E. M., Hunter, A., Oatley, G., y Kennedy, L. Polynomial-fuzzy decision tree structures for classifying medical data. *Knowledge-Based Systems*, 17(2):81–87, 2004.

[Mukkamala04] Mukkamala, S., Sung, A. H., y Abraham, A. Modeling intrusion detection systems using linear genetic programming approach. *En International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems*, págs. 633–642. Springer, 2004.

[Muni04] Muni, D. P., Pal, N. R., y Das, J. A novel approach to design classifiers using genetic programming. *IEEE transactions on evolutionary computation*, 8(2):183–196, 2004.

[Neshatian08] Neshatian, K. y Zhang, M. Genetic programming and class-wise orthogonal transformation for dimension reduction in classification problems. *En European Conference on Genetic Programming*, págs. 242–253. Springer, 2008.

[Ngan99] Ngan, P. S., Wong, M. L., Lam, W., Leung, K. S., y Cheng, J. C. Medical data mining using evolutionary computation. *Artificial Intelligence in Medicine*, 16(1):73–96, 1999.

[Nguyen09] Nguyen, Q. U., Nguyen, X. H., y ONeill, M. Semantic aware crossover for genetic programming: the case for real-valued function regression. *En Genetic Programming*, págs. 292–302. Springer, 2009.

[Oka00] Oka, S. y Zhao, Q. Design of decision trees through integration of c4. 5 and gp. *En Proc. 4th Jpn.-Australia Joint Workshop Intell. Evol. Syst*, págs. 128–135. 2000.

[Patterson07] Patterson, G. y Zhang, M. Fitness functions in genetic programming for classification with unbalanced data. *AI 2007: Advances in Artificial Intelligence*, págs. 769–775, 2007.

[Pawlak14] Pawlak, T., Wieloch, B., y Krawiec, K. Semantic Backpropagation for Designing Search Operators in Genetic Programming. *IEEE Transactions on Evolutionary Computation*, Early Access Online, 2014. ISSN 1089-778X. doi:10.1109/TEVC.2014.2321259.

[Pedregosa11] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., y Duchesnay, E. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[Petrović05] Petrović, N. y Crnojević, V. Impulse noise detection based on robust statistics and genetic programming. *En International Conference on Advanced Concepts for Intelligent Vision Systems*, págs. 643–649. Springer, 2005.

[Poli08] Poli, R., Langdon, W. B., y McPhee, N. F. *A field guide to genetic programming*. Published via `http://lulu.com` and freely available at `http://www.gp-field-guide.org.uk`, 2008. (With contributions by J. R. Koza).
URL `http://www.gp-field-guide.org.uk`

[Qing-Shan07] Qing-Shan, C., De-Fu, Z., Li-Jun, W., y Huo-Wang, C. A modified genetic programming for behavior scoring problem. *En Computational Intelligence and Data Mining, 2007. CIDM 2007. IEEE Symposium on*, págs. 535–539. IEEE, 2007.

[Quinlan86] Quinlan, J. R. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986.

[Rojas96] Rojas, R. *Neural Networks: A Systematic Introduction*. Springer, 1ª ed$\underline{\text{ón}}$., jul. 1996. ISBN 3540605053.

[Ruberto14] Ruberto, S., Vanneschi, L., Castelli, M., y Silva, S. ESAGP – A semantic GP framework based on alignment in the error space. *En* M. Nicolau, K. Krawiec, M. I. Heywood, M. Castelli, P. Garci-

Sanchez, J. J. Merelo, V. M. R. Santos, y K. Sim, eds., *17th European Conference on Genetic Programming*, tomo 8599 de *LNCS*, págs. 150–161. Springer, Granada, Spain, 23-25 abr. 2014.

[Sakprasat07] Sakprasat, S. y Sinclair, M. C. Classification rule mining for automatic credit approval using genetic programming. *En Evolutionary Computation, 2007. CEC 2007. IEEE Congress on*, págs. 548–555. IEEE, 2007.

[Sette04] Sette, S., Wyns, B., y Boullart, L. Comparing learning classifier systems and genetic programming: A case study. *Engineering Applications of Artificial Intelligence*, 17(2):199–204, 2004.

[Sherrah96] Sherrah, J., Bogner, R. E., y Bouzerdoum, B. Automatic selection of features for classification using genetic programming. *En Intelligent Information Systems, 1996., Australian and New Zealand Conference on*, págs. 284–287. IEEE, 1996.

[Sherrah97] Sherrah, J. R., Bogner, R. E., y Bouzerdoum, A. The evolutionary pre-processor: Automatic feature extraction for supervised classification using genetic programming. *Genetic Programming*, págs. 304–312, 1997.

[Shirasaka98] Shirasaka, M., Zhao, Q., Hammami, O., Kuroda, K., y Saito, K. Automatic design of binary decision trees based on genetic programming. *En Proc. The Second Asia-Pacific Conference on Simulated Evolution and Learning (SEAL'98*. Citeseer, 1998.

[Smart04] Smart, W. y Zhang, M. Continuously evolving programs in genetic programming using gradient descent. *En Proceedings of 2004 Asia-Pacific Workshop on Genetic Programming*. 2004.
URL `http://homepages.mcs.vuw.ac.nz/~mengjie/papers/will-meng-apwgp04.pdf`

[Smith80] Smith, S. F. A learning system based on genetic adaptive algorithms. 1980.

[Smith05] Smith, M. G. y Bull, L. Genetic programming with a genetic algorithm for feature construction and selection. *Genetic Programming and Evolvable Machines*, 6(3):265–281, 2005.

[Stanhope98] Stanhope, S. y Daida, J. Genetic programming for automatic target classification and recognition in synthetic aperture radar imagery. *En Evolutionary Programming VII*, págs. 735–744. Springer, 1998.

[Suárez14] Suárez, R. R., Valencia-Ramírez, J. M., y Graff, M. Genetic programming as a feature selection algorithm. *En Power, Electronics and Computing (ROPEC), 2014 IEEE International Autumn Meeting on*, págs. 1–5. IEEE, 2014.

[Suárez15] Suárez, R. R., Graff, M., y Flores, J. J. Semantic crossover operator for gp based on the second partial derivative of the error function. 2015.

[Tackett93] Tackett, W. A. Genetic programming for feature discovery and image discrimination. *En ICGA*, págs. 303–311. 1993.

[Tan02] Tan, K. C., Tay, A., Lee, T. H., y Heng, C. Mining multiple comprehensible classification rules using genetic programming. *En Evolutionary Computation, 2002. CEC'02. Proceedings of the 2002 Congress on*, tomo 2, págs. 1302–1307. IEEE, 2002.

[Tan03] Tan, K. C., Yu, Q., Heng, C., y Lee, T. H. Evolutionary computing for knowledge discovery in medical diagnosis. *Artificial Intelligence in Medicine*, 27(2):129–154, 2003.

[Tanigawa00] Tanigawa, T. y Zhao, Q. A study on efficient generation of decision trees using genetic programming. *En Proceedings of the 2nd Annual Conference on Genetic and Evolutionary Computation*, págs. 1047–1052. Morgan Kaufmann Publishers Inc., 2000.

[Teller95] Teller, A. y Veloso, M. Program evolution for data mining. *International Journal of Expert Systems Research and Applications*, 8(3):213–236, 1995.

[Tellez17] Tellez, E. S., Miranda-Jimnez, S., Graff, M., Moctezuma, D., Surez, R. R., y Siordia, O. S. A simple approach to multilingual polarity classification in twitter. *Pattern Recognition Letters*, 94:68 – 74, 2017. ISSN 0167-8655. doi:https://doi.org/10.1016/j.patrec.2017.05.024. URL `http://www.sciencedirect.com/science/article/pii/S0167865517301721`

[Teredesai04] Teredesai, A. M. y Govindaraju, V. Issues in evolving gp based classifiers for a pattern recognition task. *En Evolutionary Computation, 2004. CEC2004. Congress on*, tomo 1, págs. 509–515. IEEE, 2004.

[Thomason07] Thomason, R. y Soule, T. Novel ways of improving cooperation and performance in ensemble classifiers. *En Proceedings of the 9th annual conference on Genetic and evolutionary computation*, págs. 1708–1715. ACM, 2007.

[Tsakonas06] Tsakonas, A. A comparison of classification accuracy of four genetic programming-evolved intelligent structures. *Information Sciences*, 176(6):691–724, 2006.

[Uy10] Uy, N. Q., Hien, N. T., Hoai, N. X., y ONeill, M. Improving the generalisation ability of genetic programming with semantic similarity based crossover. *En European Conference on Genetic Programming*, págs. 184–195. Springer, 2010.

[Uy11] Uy, N. Q., Hoai, N. X., ONeill, M., McKay, R. I., y Galvan-Lopez, E. Semantically-based crossover in genetic programming: application to real-valued symbolic regression. *Genetic Programming and Evolvable Machines*, 12(2):91–119, 2011.

[Valencia-Ramirez14] Valencia-Ramirez, J. M., Raya, J. A., Cedeno, J. R., Suarez, R. R., Escalante, H. J., y Graff, M. Comparison between Genetic Programming and full model selection on classification problems. *En 2014 IEEE International Autumn Meeting on Power, Electronics and Computing (ROPEC)*, págs. 1–6. nov. 2014. doi:10.1109/ROPEC.2014.7036349.

[Vanneschi13] Vanneschi, L., Castelli, M., Manzoni, L., y Silva, S. A new implementation of geometric semantic GP and its application to problems in pharmacokinetics. *En* K. Krawiec, A. Moraglio, T. Hu, A. . Etaner-Uyar, y B. Hu, eds., *Genetic Programming*, nº 7831 en Lecture Notes in Computer Science, págs. 205–216. Springer Berlin Heidelberg, ene. 2013. ISBN 978-3-642-37206-3, 978-3-642-37207-0.

[Vanneschi14] Vanneschi, L., Castelli, M., y Silva, S. A survey of semantic methods in genetic programming. *Genetic Programming and Evolvable Machines*, 15(2):195–214, jun. 2014. ISSN 1389-2576, 1573-7632. doi: 10.1007/s10710-013-9210-0.
URL `http://link.springer.com/article/10.1007/s10710-013-9210-0`

[Wang05] Wang, S. X. y Lichodzijewski, P. Boolean genetic programming for promoter recognition in eukaryotes. *En Evolutionary Computation, 2005. The 2005 IEEE Congress on*, tomo 1, págs. 683–690. IEEE, 2005.

[Wijesinghe07] Wijesinghe, G. y Ciesielski, V. Using restricted loops in genetic programming for image classification. *En Evolutionary Computation, 2007. CEC 2007. IEEE Congress on*, págs. 4569–4576. IEEE, 2007.

[Wilcoxon45] Wilcoxon, F. Individual comparisons by ranking methods. *Biometrics Bulletin*, 1(6):80, dic. 1945. ISSN 00994987. doi:10.2307/3001968.
URL `http://www.jstor.org/discover/10.2307/3001968?uid=3738664&uid=2&uid=4&sid=21102014980993`

[Wilson95] Wilson, S. W. Classifier fitness based on accuracy. *Evolutionary computation*, 3(2):149–175, 1995.

[Xu08] Xu, C.-G. y Liu, K.-H. A gp based approach to the classification of multiclass microarray datasets. *Advanced Intelligent Computing Theories and Applications. With Aspects of Artificial Intelligence*, págs. 340–346, 2008.

[Yu07] Yu, J., Yu, J., Almal, A. A., Dhanasekaran, S. M., Ghosh, D., Worzel, W. P., y Chinnaiyan, A. M. Feature selection and molecular classification of cancer using genetic programming. *Neoplasia*, 9(4):292IN1–303IN3, 2007.

[Zhang04a] Zhang, M. y Smart, W. Genetic programming with gradient descent search for multiclass object classification. *En* M. Keijzer, U.-M. OReilly, S. Lucas, E. Costa, y T. Soule, eds., *Genetic Programming*, nº 3003 en Lecture Notes in Computer Science, págs. 399–408. Springer Berlin Heidelberg, ene. 2004. ISBN 978-3-540-21346-8, 978-3-540-24650-3.

[Zhang04b] Zhang, Y. y Bhattacharyya, S. Genetic programming in classifying large-scale data: an ensemble method. *Information Sciences*, 163(1):85–101, 2004.

[Zhang07] Zhang, L. y Nandi, A. K. Fault classification using genetic programming. *Mechanical Systems and Signal Processing*, 21(3):1273–1284, 2007.

[Zhang08] Zhang, Y., Li, H., Niranjan, M., y Rockett, P. Applying cost-sensitive multiobjective genetic programming to feature extraction for spam e-mail filtering. *En European Conference on Genetic Programming*, págs. 325–336. Springer, 2008.