IMPROVING SEARCH IN GEOMETRIC SEMANTIC GENETIC PROGRAMMING

LUIZ OTÁVIO VILAS BÔAS OLIVEIRA

IMPROVING SEARCH IN GEOMETRIC SEMANTIC GENETIC PROGRAMMING

Tese apresentada ao Programa de Pós-Graduação em Ciência da Computação do Instituto de Ciências Exatas da Universidade Federal de Minas Gerais como requisito parcial para a obtenção do grau de Doutor em Ciência da Computação.

Orientadora: Gisele Lobo Pappa
Coorientador: Fernando Esteban Barril Otero

Belo Horizonte

28 de setembro de 2016

LUIZ OTÁVIO VILAS BÔAS OLIVEIRA

IMPROVING SEARCH IN GEOMETRIC SEMANTIC GENETIC PROGRAMMING

Thesis presented to the Graduate Program in Computer Science of the Federal University of Minas Gerais in partial fulfillment of the requirements for the degree of Doctor in Computer Science.

Advisor: Gisele Lobo Pappa
Co-Advisor: Fernando Esteban Barril Otero

Belo Horizonte
September 28, 2016

© 2016, Luiz Otávio Vilas Bôas Oliveira.
Todos os direitos reservados

Ficha catalográfica elaborada pela Biblioteca do ICEx - UFMG

Oliveira, Luiz Otávio Vilas Bôas.

O48i Improving search in geometric semantic genetic programming. / Luiz Otávio Vilas Bôas Oliveira. – Belo Horizonte, 2016.
xxvii, 119 f.: il.; 29 cm.

Tese (doutorado) - Universidade Federal de Minas Gerais – Departamento de Ciência da Computação.

Orientadora: Gisele Lobo Pappa.
Coorientador: Fernando Esteban Barril Otero.

1. Computação - Teses. 2. Programação genética (Computação). I. Orientadora. II. Coorientador. III. Título.

CDU 519.6*82(043)

UNIVERSIDADE FEDERAL DE MINAS GERAIS
INSTITUTO DE CIÊNCIAS EXATAS
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

FOLHA DE APROVAÇÃO

Improving search in geometric semantic genetic programming

LUIZ OTAVIO VILAS BOAS OLIVEIRA

Tese defendida e aprovada pela banca examinadora constituída pelos Senhores:

PROFA. GISELE LOBO PAPPA - Orientadora
Departamento de Ciência da Computação - UFMG

PROF. FERNANDO ESTEBAN BARRIL OTERO - Coorientador
University of Kent - UK

PROF. FERNANDO JOSÉ VON ZUBEN
Departamento de Engde Computação e Automação Industrial - UNICAMP

PROF. THIAGO FERREIRA DE NORONHA
Departamento de Ciência da Computação - UFMG

PROF. RICARDO HIROSHI CALDEIRA TAKAHASHI
Departamento de Matemática - UFMG

PROF. VINÍCIUS VELOSO DE MELO
Instituto de Ciência e Tecnologia - UNIFESP

Belo Horizonte, 28 de setembro de 2016.

Acknowledgments

I would like to thank my parents, José Carlos and Ângela, for everything. You always supported me and trusted in my decisions, no matter how crazy they could seem. To my girlfriend, Amanda, for the patience and for always trusting in me, even when I didn't. I'm coming home...

A special thanks to my advisor, Dr. Gisele, that despite wanting to punch me for my jokes, never did it. Kidding. Thank you for all the help, for the precious advices and for the effort you employed to help me with a zillion of issues. Also to my co-advisor, Dr. Fernando, and to Professor Takahashi, for all the ideas and support.

I also would like to thank my friends and coworkers. Not only for the help, ideas and suggestions, but also for the beers and the laughs. The list is long, so: "Thanks everybody". I also would like to thank specific friends that had a crucial role in this thesis. Thanks Alex, Luís (with 's'), Júlio and Andinho.

This thesis was conducted with CNPq support.

"The true sign of intelligence is not knowledge but imagination."
(Albert Einstein)

Resumo

Nos últimos anos diversos trabalhos surgiram na tentativa de introduzir informação semântica no processo evolutivo da Programação Genética (PG). Em particular, a PG Semântica Geométrica (PGSG) foi proposta como um método que atua sobre a sintaxe dos programas pais, produzindo descendentes respeitando algum critério semântico.

Nesta tese, focamos nas questões em aberto existentes na PGSG e métodos relacionados, aplicados à regressão simbólica. Apresentamos a definição de semântica adotada nesta tese e uma visão geral dos métodos que exploram a semântica enquadrados nesta definição. Em seguida, apresentamos e atacamos questões relativas à PGSG.

Em primeiro lugar, investigamos o impacto do cruzamento semântico geométrico com diferentes funções de distância sobre a busca, e a possibilidade de ajustar de forma ótima seus coeficientes em vez de escolhê-los aleatoriamente. Os resultados mostram que a distância de Manhattan tem melhor desempenho em termos de erro no teste, e que a otimização dos coeficientes do cruzamento não consegue melhorar significativamente a busca.

Também apresentamos a Regressão Sequencial Simbólica (RSS), uma tentativa de controlar o crescimento exponencial do tamanho dos indivíduos causado pela utilização do cruzamento semântico geométrico. Depois de gerar uma função subótima com PG canônica, a RSS aproxima os erros de saída através de outra função, em uma iteração posterior, e concatena-as com o operador de cruzamento. Uma análise experimental mostra que a RSS tem um desempenho semelhante à PGSG, gerando soluções menores.

Em adição, esta tese explora um arcabouço heurístico, chamado de dispersão geométrica (DG), para a construção de operadores que movem os indivíduos para áreas menos densas do espaço de busca, ao redor do vetor de saída alvo. Os resultados experimentais indicam que os operadores de dispersão geométrica podem melhorar a busca e espalhar as soluções ao redor da solução alvo.

Por último, apresentamos um estudo do impacto da seleção de instâncias de treinamento, a fim de reduzir a dimensionalidade do espaço semântico. Duas abordagens são consideradas: (i) aplicar métodos de seleção de instâncias como uma etapa de pré-processamento, antes que os pontos de treinamento sejam apresentados à PGSG;

(ii) incorporar a seleção de instâncias na evolução realizada pela PGSG. A análise experimental mostra que o desempenho da PGSG melhora com a redução de instâncias durante a evolução.

Abstract

In the last few years a variety of works has emerged attempting to introduce semantic awareness to the evolutionary process performed by Genetic Programming (GP). In particular, Geometric Semantic GP (GSGP) was proposed as a method that acts on the syntax of the parent programs producing offspring respecting a semantic criterion.

In this thesis we focus on the open issues presented by GSGP and related methods, applied to symbolic regression. We present the definition of semantics adopted in this thesis and an overview of the methods that explore semantics framed in this definition. Then we present and tackle issues regarding GSGP.

We first investigate the impact of the geometric semantic crossover with different distance functions on the search, and the possibility of optimally adjusting its coefficients instead of choosing them randomly. The results show that the Manhattan distance has best performance in terms of test error, and that optimizing the crossover coefficients cannot improve significantly the search.

We also present the Sequential Symbolic Regression (SSR), an attempt to control the exponential growth on the size of the individuals caused by the use of the geometric semantic crossover. After generating a suboptimal function with a canonical GP, SSR approximates the output errors by another function, in a subsequent iteration, and concatenates them with the crossover operator. An experimental analysis shows that SSR has similar performance to GSGP while generating smaller solutions.

In addition, this thesis explores a heuristic framework, called Geometric Dispersion (GD), to construct operators that move individuals to less dense areas of the search space around the target output vector. Experimental results indicate that GD operators can improve the search and spread the solutions around the target solution.

Last, we present a study of the impact of selecting training instances in order to reduce the semantic space dimensionality. Two approaches are considered: (i) to apply current instance selection methods as a pre-process step before training points are given to GSGP; (ii) to incorporate instance selection to the evolution of GSGP. The experimental analysis shows that GSGP performance is improved by using instance reduction during the evolution.

List of Figures

List of Tables

List of Abbreviations

ADF Automatic Defined Function

AGD Additive Geometric Dispersion

AGS Adapted Geometric Semantic

AGX Approximately Geometric Crossover

AM Adaptive Mutation

APV Adjusted *P-Values*

AQ Analytic Quotient

BDD Binary Decision Diagram

CCISR Class Conditional Instance Selection for Regression

CI Competent Initialization

CNN Condensed Nearest Neighbor

CTS Competent Tournament Selection

DAM Doubly Adaptive Mutation

EC Evolutionary Computation

ECGP Embedded Cartesian Genetic Programming

ENN Edited Nearest Neighbor

ESAGP Error Space Alignment Genetic Programming

EGSGP Efficient Geometric Semantic Genetic Programming

GA Genetic Algorithm

GD Geometric Dispersion

GLiB Genetic Library Builder

GP Genetic Programming

GRR Genetic Recursive Regression

GRSR Genetic Recursive Symbolic Regression

GSGP Geometric Semantic Genetic Programming

GSM Geometric Semantic Mutation

GSX Geometric Semantic Crossover

GSXE Geometric Semantic Crossover for fitness function based on Euclidean distance

GSXE-C Optimized Convex Euclidean-Based Geometric Semantic Crossover

GSXE-L Optimized Non-Convex Euclidean-Based Geometric Semantic Crossover

GSXM Geometric Semantic Crossover for fitness function based on Manhattan distance

IQR Interquartile Range

LGX Locally Geometric Semantic Crossover

MAE Mean Absolute Error

MDD Mean Dimension Distribution

MDL Minimum Description Length

MGD Multiplicative Geometric Dispersion

ML Machine Learning

MSE Mean Squared Error

POGP Pair Optimization Genetic Programming

PSE Probabilistic instance Selection based on the Error

R^2 Coefficient of Determination

RHH Ramped Half-and-Half

RMSE Root Mean Squared Error

ROBDD Reduced Ordered Binary Decision Diagram

RST Random Sample Technique

SAC Semantic Aware Crossover

SDC Semantically Driven Crossover

SDI Semantically Driven Initialization

SDM Semantically Driven Mutation

SGI Semantic Geometric Initialization

SSC Semantic Similarity based Crossover

SSE Sum of Squared Errors

SSHC Semantic Stochastic Hill Climber

SSM Semantic Similarity based Mutation

SSR Sequential Symbolic Regression

SX(+) Approximately Geometric Semantic Crossover(+)

TCNN Threshold Condensed Nearest Neighbor

TENN Threshold Edited Nearest Neighbor

UPDRS Unified Parkinson's Disease Rating Scale

Contents

Chapter 1

Introduction

Evolutionary computation is a research field within computer science that is inspired by the process of natural evolution proposed by Darwin, which is strongly based on the concept of survival of the fittest. One of the branches of the field comprises Genetic Programming (GP), a search mechanism that explores the space of possible computational programs for a solution capable of solving a target problem [Koza, 1992a; Banzhaf et al., 1998].

The GP evolution process starts with an initial population of individuals representing candidate solutions to solve the problem at hand. At each generation, individuals are selected with probability proportional to a fitness function to undergo genetic operations, such as crossover and mutation, to compose the next generation's population. This process is repeated until the fulfilment of some criterion, usually a minimum fitness value or a maximum number of generations.

The evolution process improves candidate solutions autonomously, guided only by high-level information about the problem being solved. This characteristic places GP within the Machine Learning (ML) field [Mitchell, 1997], with most of its applications in supervised learning. Given a finite set of input-output pairs representing the training cases, supervised learning comprises a set of ML methods that induce a model mapping the set of inputs to their respective outputs [Murphy, 2012].

Particularly, GP is widely applied to a type of supervised learning named symbolic regression. This task involves finding a mathematical expression, in symbolic form, which fits (or approximately fits) the training cases. Different from other forms of regression, which adjust the parameters of an equation in a predetermined form—e.g. linear and polynomial regressions—symbolic regression searches both the parameters and the form of equations simultaneously [Koza, 1992a]. This thesis focus on the task of symbolic regression in problems where the inputs and outputs of the training cases are defined in the real domain.

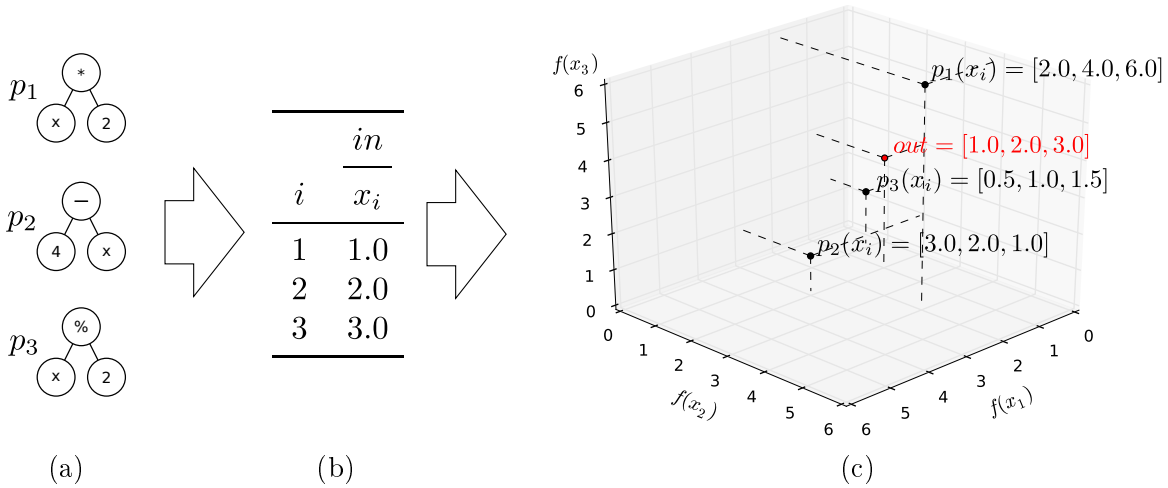
One of the concerns involving symbolic regression regards the generalization capability of the model produced by GP. It is important that the solutions evolved are not prone to overfitting, i.e., the model adjusted to the data should not be excessively meticulous, modelling even the noise inherent to the data. Overfitting leads to models characterized by poor performance, unable to generalize the learned mapping to new data [Murphy, 2012].

Traditionally GP uses two different genetic operators, namely crossover and mutation, to modify individuals during the evolution process. The crossover operator exchanges code between two parents to generate two offspring and the mutation operator replaces part of the individual by a randomly generated segment. Both operators should guarantee the production of valid offspring in terms of the structure of the programs, defined by their syntax—such as the function arity or the type of input data—but they are blind regarding their effect on the behaviour of the offspring, i.e., their semantics. While syntax refers to the structure, semantics is concerned with the meaning of the program represented by the individual. The fact that these methods disregard semantics may produce negative effects to the search process. For instance, the canonical crossover operator may act as a destructive operator, creating offspring worse than their parents by splitting useful program blocks [Banzhaf et al., 1998; Majeed and Ryan, 2006]. Nguyen et al. [2009b, 2011, 2013], for example, showed that the crossover destructive effect can be reduced through the use of semantic information.

There are different definitions for the semantics of a program in the literature. It can be its canonical representation with one-to-one correspondence with its semantics—e.g., Binary Decision Diagrams (BDD) for Boolean expressions [Beadle and Johnson, 2008, 2009a,b]. It can be a description of its behaviour through logical formalism [Johnson, 2007]. Or it can be the mathematical function computed by the program—i.e., the set of outputs the program generates when applied to a given set of inputs [Moraglio et al., 2012]. The latter definition is the one adopted in this thesis.

In the supervised learning context, the semantics of a program is described as the vector of outputs the program generates when applied to the set of inputs defined by the set of training cases. This vector can be represented as a point in an n-dimensional metric space $\mathcal{S}$ (called semantic space), where n is the size of the training set. Figure 1.1 depicts the evaluation of the semantics of an individual and its representation in the semantic space. Notice that the target output *out*—i.e., the vector of real values being searched, defined by the outputs given in the training set—is also representable in the semantic space.

Several works involving the use of semantic awareness in GP—hereafter referred as semantic GP methods—presented positive impact on the search performance (see the



(a) (b) (c)

	in				
i	x_i				
1	1.0				
2	2.0				
3	3.0				

(b)

	in	*out*	Population semantics		
i	x_i	y_i	$p_1(x_i)$	$p_2(x_i)$	$p_3(x_i)$
1	1.0	1.0	2.0	3.0	0.5
2	2.0	2.0	4.0	2.0	1.0
3	3.0	3.0	6.0	1.0	1.5

(d)

Figure 1.1: Example of semantics calculation. (a) The functions—syntactically represented as program trees—are applied to (b) the set of inputs, generating output vectors, represented in the (c) semantic space. (d) The table summarizes the picture.

survey presented by Vanneschi et al. [2014a] for an overview of these works), which led different authors to propose distinct methodologies to organize semantic GP methods [Nguyen, 2011; Vanneschi et al., 2014a; Pawlak, 2014].

In this thesis we adopt a taxonomy inspired by the work of Pawlak [2015] that discerns *geometric* and *non-geometric* semantic GP methods. The former category, which is the main focus of this thesis, includes methods that exploit geometric shapes to describe the spatial relation between parents and offspring in the semantic space. The latter category refers to the remaining methods, including mainly trial-and-error approaches that repeatedly apply standard syntactic operators, with no underlying geometric semantic properties, until certain semantic conditions are met.

1.1 Motivation

One of the most popular works in the field of geometric semantic GP proposed the geometric semantic crossover and mutation operators in the context of the Geometric Semantic Genetic Programming (GSGP) [Moraglio et al., 2012]. These operators search the space of the underlying semantics of the programs, benefiting the search process. This is because the fitness function spanning the semantic space has a unimodal conic shape, which can be efficiently optimized through evolutionary computation techniques, according to formal evidence presented by Moraglio [2011].

The geometric semantic crossover operator combines two parents, resulting in one offspring located in the metric segment between the parents, and the geometric semantic mutation operator generates offspring by applying perturbations to the parents, ensuring that the offspring is placed in a hypersphere with the parent as the centre.

Given its properties, GSGP has been applied to a wide range of problems [Vanneschi et al., 2013, 2014b, 2015; Castelli et al., 2013a,b, 2014b, 2015b,c,d,e]. However the geometric semantic operators present some issues:

Issue 1. *The set of possible offspring generated by the geometric semantic crossover is delimited, in the semantic space, by the set of available parents, which is directly related to the initial population.*

By construction, the geometric semantic crossover guarantees that the resulting offspring are enclosed, in the semantic space, by the convex hull[1] of the semantics of the available parents. If the only genetic operator involved in the search is the geometric semantic crossover, then the set of possible individuals generated during evolution is delimited by the convex hull of the semantics of the initial population [Pawlak, 2015]. Even with other genetic operators, the initial population has an important role on the search performed by the geometric semantic crossover.

The effect of the population initialization in the context of the GSGP has already been explored in the literature. Pawlak [2015] presents a trial-and-error approach to generate individuals capable of expanding the population's convex hull and Pawlak and Krawiec [2016] introduce a method to initialize the population such that the desired output is guaranteed to be inside the population's convex hull. However, these methods present some drawbacks, as waste of resources and training overfitting, which indicate that this issue should be further investigated.

[1]The *convex hull* of a set S of points in $\mathbb{R}^n$ is the set of all convex combinations of points in S [Lay, 2012].

Another strategy to tackle this limitation involves moving the individuals during the evolution regarding their impact on the population's convex hull. However, to the best of our knowledge, so far there is no work in the literature exploring this strategy.

Issue 2. *The geometric semantic crossover operator deserves further investigation regarding its impact on the search process.*

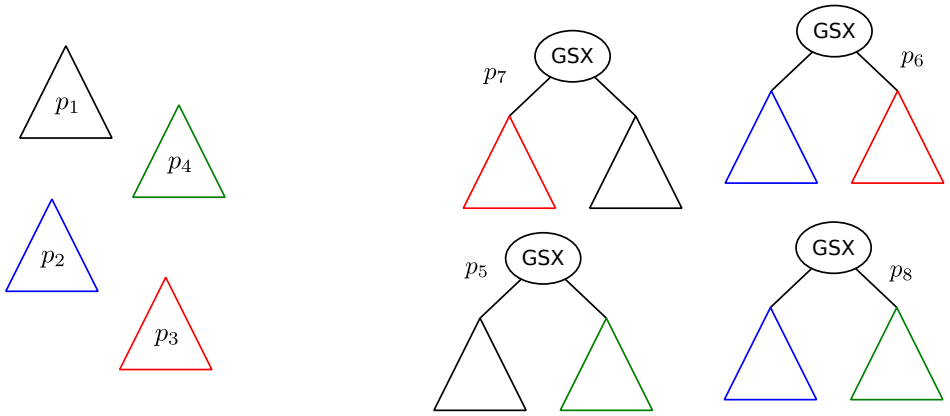
The fact that the offspring generated by the geometric semantic crossover operator are always placed in the segment connecting their parents, as previously explained, limits the space covered by this operator. However, it is not clear what is the impact of the operator in the search process in terms of training and test error. Furthermore, by definition this operator can be further improved by using mathematical tools to optimally combine the individuals. A study analysing the effects of optimally combining individuals on the search process within the geometric semantic crossover operator is worthy being investigated.

Issue 3. *By construction, the size of the offspring generated by the geometric semantic crossover is always larger than the size of their parents and grows exponentially in the number of generations.*

The geometric semantic crossover operator generates one offspring from two parents through a convex combination of them. Hence, the expected size of an individual generated by this operator in generation g is proportional to two times the expected size of the individuals in generation $g-1$ (as presented in the Figure 1.2), leading to exponential growth of the individuals. This uncontrolled growth excessively increases the memory needed to store the individuals and the computational effort required to compute the fitness and semantics of the individuals, which may be prohibitive in practical applications.

Two approaches to attack this problem have been presented in the literature: (1) implementing GSGP operators in a more compact manner, through the use of pointers to the subtrees stored in memory [Vanneschi et al., 2014b] or (2) simplifying the trees during the evolution process [Moraglio et al., 2012]. The first approach solves the problem of excessive memory consumption and computation effort, but does not affect the final size of the solutions. The second approach, on the other hand, is still underexplored and deserves more attention.

A third approach, not presented in the literature, involves employing the geometric semantic crossover to combine solutions outside the evolutionary process, reducing the frequency with which this operator is applied.



(a) Generation i (b) Generation $i+1$

Figure 1.2: Exponential growth over the generations promoted by the geometric semantic crossover operator (GSX). (a) The GSX operator is applied to the population $P = \{p_1, p_2, p_3, p_4\}$ generating (b) the population $P' = \{p_5, p_6, p_7, p_8\}$ with individuals two times larger.

Issue 4. *The semantic space searched by geometric semantic operators is defined by the training cases, such that each training case corresponds to one dimension of the space. Thus, larger sets imply higher dimensional semantic spaces.*

The semantics of a program is described by an n-element vector whose elements correspond to the outputs the program generates when applied to each input defined by the set of training cases. These vectors can be represented in an n-dimensional metric space called semantic space, where the search takes place in GSGP.

Larger training sets lead to higher dimensional semantic spaces, increasing the search space explored by GSGP operators and making the search harder. However, there is a lack of effort to decrease the size of the semantic space in order to improve the search.

1.2 Objectives

The overall objective of this thesis is to explore open issues in geometric semantic GP methods and propose improvements to cope with shortcomings related to them. In particular, we intend to answer the following questions, regarding the issues presented in Section 1.1.

1. How can the population's convex hull be expanded in order to contain the desired output in the semantic space? (Issue 1)

2. Is the geometric semantic crossover beneficial to the search? Is there an optimal strategy to combine their parents? (Issue 2)

3. How can we overcome the growth problem related to geometric semantic operators? (Issue 3)

4. Can instance selection algorithms be used to reduce the semantic space? What is the impact of these methods in the search? (Issue 4)

1.3 Contributions

This thesis explores the use of semantic information to improve the GP search process. More precisely, the investigation of several aspects related to semantics in GP performed in this thesis have generated the following contributions so far:

A generic framework for geometric dispersion operators (Issue 1): This thesis presents a framework[2] for building a new class of geometric semantic operators, called geometric dispersion operators, which disperses the population through the semantic space such that the convex hull described by the individuals increases the chance of including the desired output [Oliveira et al., 2016c,d]. These operators adopt a greedy strategy by equalizing the concentration of individuals around the target output considering each dimension of the semantic space separately.

A study of the impact of the geometric semantic crossover operator in the GSGP evolution (Issue 2): This thesis analyses the impact of using different distance functions within the geometric semantic crossover in the search process and presents two strategies to numerically optimize the crossover coefficients. We empirically investigate the impact of this optimization on the performance of the geometric semantic crossover, and also the effects of the geometric semantic crossover per se in the search process [Albinati et al., 2014, 2015].

Size reduction of the resulting individuals in GSGP: This thesis presents a first attempt to maintain the size of the solutions in a manageable level while applying the geometric semantic crossover operator (Issue 3). However, the method is not a geometric semantic method—it employs the geometric semantic crossover to "concatenate" the functions generated by a canonical GP method [Oliveira et al., 2015, 2016b].

A Study of the impact of instance selection methods on GSGP search (Issue 4): Given that the search performed by GSGP operators is directly connected to the semantic space, an analysis of the effects of selecting instances prior to the evo-

[2]The source code is freely available at: `https://github.com/luizvbo/gsgp-gd`.

lutionary process was performed in this thesis. Along with instance selection methods from the literature, we proposed a new method, called probabilistic instance selection based on the error, which allows the reduction of the search space [Oliveira et al., 2016a].

The remainder of this thesis is organised as follows. Chapter 2 describes the main concepts of genetic programming in its canonical form, describing aspects relevant to its functioning. Chapter 3 defines semantics in the GP context and presents an overview of the geometric and non-geometric semantic methods. Chapter 4 presents several studies regarding different aspects of the geometric semantic operators, with a greater focus on the geometric semantic crossover operator. Chapter 5 introduces a new strategy to perform symbolic regression by iteratively learning solutions from a transformed set of problems. Chapter 6 presents a framework to build operators that redistribute the population around the desired output vector. Finally, Chapter 7 proposes strategies to reduce the training set used to feed GSGP in order to reduce the dimensionality of the semantic space. Chapter 8 concludes the thesis, summarizing our findings and presenting perspectives for future work.

1.4 Publications

This thesis has resulted so far in the following publications:

1. Oliveira, L. O. V. B., Miranda, L. F., Pappa, G. L., Otero, F. E. B., and Takahashi, R. H. C. (2016a). Reducing dimensionality to improve search in semantic genetic programming. In Handl, J., Hart, E., Lewis, P. R., López-Ibánez, M., Ochoa, G., and Paechter, B., editors, *Proc. of the 14th International Conference on Parallel Problem Solving from Nature (PPSN XIV)*, volume 9921 of *LNCS*, pages 375--385. Springer International Publishing.

2. Oliveira, L. O. V. B., Otero, F. E. B., Miranda, L. F., and Pappa, G. L. (2016b). Revisiting the sequential symbolic regression genetic programming. In *Proc. of the 5th Brazilian Conference on Intelligent System (BRACIS)*. (to appear).

3. Oliveira, L. O. V. B., Otero, F. E. B., and Pappa, G. L. (2016c). A dispersion operator for geometric semantic genetic programming. In *Proc. of the the 2016 Genetic and Evolutionary Computation Conference*, GECCO '16, pages 773--780. ACM. **Received the Best Paper Award in the track "Genetic Programming"**.

4. Oliveira, L. O. V. B., Otero, F. E. B., and Pappa, G. L. (2016d). A generic framework for building dispersion operators in the semantic space. In *Genetic Programming Theory and Practice XIV*. Springer International Publishing. (to appear).

5. Oliveira, L. O. V. B., Otero, F. E. B., Pappa, G. L., and Albinati, J. (2015). Sequential symbolic regression with genetic programming. In Riolo, R., Worzel, B., and Kotanchek, M., editors, *Genetic Programming Theory and Practice XII*, Genetic and Evolutionary Computation, pages 73–90. Springer International Publishing.

6. Albinati, J., Pappa, G. L., Otero, F. E. B., and Oliveira, L. O. V. B. (2015). The effect of distinct geometric semantic crossover operators in regression problems. In Machado, P., Heywood, M. I., McDermott, J., Castelli, M., García-Sánchez, P., Burelli, P., Risi, S., and Sim, K., editors, *18th European Conference, EuroGP 2015*, volume 9025 of *Lecture Notes in Computer Science*, pages 3–15. Springer International Publishing. **Nominated for Best Paper Award at EuroGP 2015**.

7. Albinati, J., Pappa, G. L., Otero, F. E. B., and Oliveira, L. O. V. B. (2014). A study of semantic geometric crossover operators in regression problems. Semantic Methods in Genetic Programming. Workshop at Parallel Problem Solving from Nature 2014 conference.

Chapter 2

Genetic Programming

As an Evolutionary Computation (EC) technique, Genetic Programming (GP) [Koza, 1992a; Banzhaf et al., 1998] solves problems through iterative modifications in a population of individuals, each individual representing a candidate solution to the problem being tackled, in the form of a computer program. The GP process simulates the natural selection, where each individual competes with the others to survive based on its fitness. The individuals that survive the selection step undergo genetic operations simulating genetic changes occurring in biology before being added to the new population.

The evolution process performed by GP can be viewed as a search in the space of all (GP representable) solutions to the problem. GP distinguishes from other EC techniques by representing solutions as interpretable programs, usually represented as tree structures, which must be interpreted or executed in order to evaluate their fitness [Poli et al., 2008]. Besides the wide range of its applications, GP is usually associated with the Machine Learning (ML) field [Banzhaf et al., 1998; Eiben and Smith, 2003]—the field from computer science that studies computational techniques capable of automatically improving their performance in a task through experience [Mitchell, 1997].

Algorithm 1 presents an overview of a traditional GP method. The first step of the procedure is to generate an initial population of candidate solutions. At each generation, the method evaluates the population members, as specified by the fitness function, and assigns them a fitness value. A set of these individuals is selected with probability proportional to their fitness and submitted to the genetic operators, such as crossover and mutation. The resulting individuals replace the current population and the generation finishes. The method then verifies if the stopping criteria were satisfied, such as reaching a maximum number of generations or finding a satisfactory

Algorithm 1: Canonical GP method

```
   Input: Training data (T), list of GP parameters (parameters)
1  Procedure GP(T, parameters)
2      P ← Initialize the population;
3      repeat
           // Each loop iteration corresponds to one GP generation.
4          Evaluate the individuals from P;
5          parents ← Select individuals according to their fitness;
6          offspring ← Perform genetic operations on parents;
7          P ← offspring;
8      until the stopping criteria have been met;
9      return best individual of P
```

solution (based on a new evaluation of the population). In affirmative case, GP stops its execution and returns the best solution found; otherwise, it starts a new generation.

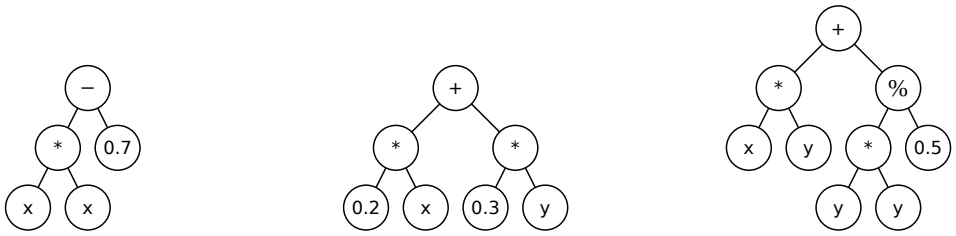
The next sections scrutinize the main aspects relevant to the search process performed by GP.

2.1 Representation

In biology, the term genotype refers to the organism's genetic coding and the phenotype corresponds to the observable characteristics of an organism, such as its size and shape and its behaviour. The phenotype is the product of the individual's genotype along with the environment it has experienced, e.g., temperature, sunlight incidence and food consumption [Futuyma, 2013]. In GP, the phenotype refers to the behaviour of the program represented by an individual when executed and the genotype refers to the actual representation of the individual [Banzhaf et al., 1998].

Parse trees are the most common form of representing candidate solutions in GP [Poli et al., 2008], although other representations have been proposed in literature, such as linear sequences of instructions [Brameier and Banzhaf, 2007], graph structures [Poli, 1999] and the Cartesian GP representation [Miller, 1999].

The trees evolved are composed of elements from the *terminal* and *function* sets. The first set defines the elements that appear in the leaves of the tree, i.e., the terminals, such as variables and constants; the second set defines the elements that appear internally, such as arithmetic operations, trigonometric functions and Boolean operations, and their respective arities, i.e., the number of arguments they accept. For instance, Figure 2.1 presents three different trees generated from the function set



(a) $x^2 - 0.7$ (b) $0.2x + 0.3y$ (c) $xy + (y^2 \% 0.5)$

Figure 2.1: Example of parse trees used in GP and their respective equivalent functions.

$\mathcal{F} = \{+, -, *, \%\}$ (all elements have 2-arity) and the terminal set $\mathcal{T} = [0, 1] \cup \{x, y\}$, such that $\mathcal{T} \subseteq \mathbb{R}$.

The function and terminal sets are very important to the search process, since they directly define the space of solutions reachable by GP. If these sets are too small or do not include some key functions or terminals, the space explored may not contain satisfactory solutions. On the other hand, if the function set contains too many elements, the size of the space searched may escalate, making the search harder [Banzhaf et al., 1998]. Another important property of the function set, called closure, states that each function must handle every possible input it might receive [Koza, 1992a]. The arithmetic division operator, for example, does not hold this property, since it may generate an output equal to zero, but cannot accept zero as a divisor. Depending on the programming environment, a division by zero may generate exceptions, errors or terminate the program. In order to overcome this problem, some alternatives were proposed as the protected division (usually represented by the symbol '%') [Koza, 1992a] and the Analytic Quotient (AQ) operator [Ni et al., 2013].

Protected division avoids dividing by zero by returning a constant (usually 1) when the divisor is zero. However, as presented by Keijzer [2003], protected operators may induce asymptotes in regions of the input space not covered by the training set. When the GP induced function is applied to unseen data, it may generate arbitrarily high or low values inside these regions. The solution presented by Ni et al. [2013] to overcome this pitfall is to replace the protected division by a continuous operator, called Analytic Quotient (AQ) operator, defined as

$$AQ(a, b) = \frac{a}{\sqrt{1 + b^2}} \ , \tag{2.1}$$

where a and b are respectively equivalent to the dividend and to the divisor in arithmetic division.

Algorithm 2: The `full` tree initialization algorithm

```
   Input: depth (d), maximum depth (D),
1          terminal set (T), function set (F)
2  Procedure full(d, D, T, F)
3      if d = D then
4          return random node from T;
5      else
6          n ← random node from F;
7          foreach possible input argument arg of n do
8              Fill arg with full(d + 1, D, T, F);
9          return n
```

2.2 Population Initialization

The first step of the evolutionary process is to generate an initial population. There are different methods for generating random individuals to compose the initial population. The most used and known of them were proposed by Koza [1992a], namely *full*, *grow* and *ramped half-and-half* (RHH).

The full method randomly selects functions from the function set and inserts them into the tree until the maximum tree depth[1] is reached. Then, the method randomly selects nodes from the terminal set and inserts them as tree leaves.

One problem related to the full method is that it only generates complete trees, reducing the range of possible trees. The grow method, on the other hand, may generate asymmetric trees. The method fills the tree by randomly choosing nodes from both the function and terminal sets until reaching the maximum depth or completing all the function nodes with terminals. When a function node reaches the maximum depth, the method completes it with randomly chosen terminals.

Algorithms 2 and 3 present common recursive procedures of the full and grow methods, respectively. Both implementations receive as input the terminal ($\mathcal{T}$) and function ($\mathcal{F}$) sets, the maximum allowed depth (D) and a depth control parameter (d), used by the recursion, which is set to 0 when the methods are called [Luke, 2000].

In order to generate trees with higher variability in size and shape, Koza [1992a] proposed the RHH method, a hybrid of the two aforementioned methods. Let D_{rhh} be the RHH maximum allowed depth, the method initializes 50% of the trees with grow and 50% with full method, equally distributed through different maximum depths varying from 2 to D_{rhh}.

[1] We adopt the depth definition from Poli et al. [2008], i.e., the depth of a node is the number of edges that must be transversed from the root of the tree to reach the node.

Algorithm 3: The `grow` tree initialization algorithm

```
   Input: depth (d), maximum depth (D),
1          terminal set (T), function set (F)
2  Procedure grow(d, D, T, F)
3      if d = D then
4          return random node from T;
5      else
6          n ← random node from T ∪ F;
7          if n ∈ F then
8              foreach possible input argument arg of n do
9                  Fill arg with grow(d + 1, D, T, F);
10         return n
```

2.3 Individual Evaluation

Genetic programming, as a metaphor for the natural selection, implements a mechanism based on the survival of the fittest. The better the solution represented by an individual, the more likely it will survive to compose the next generation.

The quality of the solution is a problem-dependent measure, defined by the fitness function [Banzhaf et al., 1998]. It may consider one desirable quality to be present at the solution, which comprehend single-objective problems, or more than one, which comprehend multi-objective problems. Multi-objective problems may demand considerable modifications to GP and are out of the scope of this thesis (see [Deb, 2001] for more details).

In canonical GP, the fitness of an individual is the only information available to measure the quality of the solutions and drive the search process. In single-objective problems, the fitness is usually a value proportional (or inversely proportional) to the performance of the program represented by the individual when applied to the problem, defined by a set of training cases. For example, the fitness can be defined in relation to the success or error rate in solving the problem, the amount of time spent or computational cost.

The mapping between genotype and fitness can be viewed in the fitness landscape, a d-dimensional plot where the height dimension represents the fitness and the other $d-1$ dimensions represent the genotypes [Langdon and Poli, 2002; Eiben and Smith, 2003]. If the genotypes can be visualized in two dimensions, the fitness landscape has the shape of a three-dimensional map, which may contain hills and valleys. Depending on the nature of the problem, i.e., minimization or maximization, GP searches the

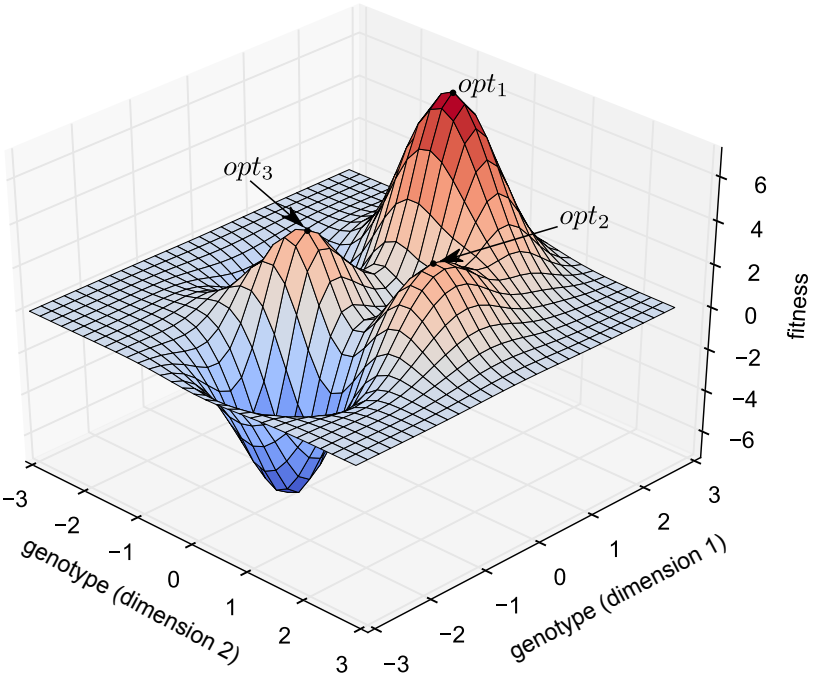


Figure 2.2: Example of fitness landscape in a 2D genotype space.

deepest valley or highest hill, respectively. Figure 2.2 presents a fitness landscape for a maximization problem. The point opt_1 represents the global optimum—i.e., the point with highest fitness in all the landscape—and the points opt_2 and opt_3 represent local optima—i.e., points with fitness higher than the fitness of every neighbour point.

2.4 Selection

GP usually adopts a probabilistic approach to select potentially good solutions to generate new offspring and compose the subsequent populations. The probability of selecting a specific individual is proportional to its suitability to solve the problem, measured by the fitness function. The higher the quality of the individual, the higher the probability that it becomes a parent and propagates its genotype [Eiben and Smith, 2003].

The genetic operators, like mutation and crossover, have the potential to generate a number of distinct offspring greater than the population size. The necessity of restricting the number of offspring to be accommodated in the surviving population results in a competition between the individuals, performed through the selection process. The degree to which the better individuals are favoured is regulated by the selective pressure [Banzhaf et al., 1998].

Tournament selection is one of the most commonly applied methods for selecting individuals in GP. Instead of performing a competition within the whole population, the tournament method randomly selects, with uniform probability, a subset composed of t individuals from the population and the competition occurs within this subset. Only the best individual from this subset is selected as parent. The parameter t defines the tournament size and allows to adjust the selective pressure imposed by the method, i.e., the larger the tournament size, the higher the pressure and, consequently, the faster the population converges [Banzhaf et al., 1998; Blickle, 2000; Poli et al., 2008].

Different EC selection mechanisms can be used by GP instead of the tournament method [Poli et al., 2008]. For instance, Banzhaf et al. [1998] describes the fitness-proportional, truncation (or (μ, λ) selection) and ranking selection methods.

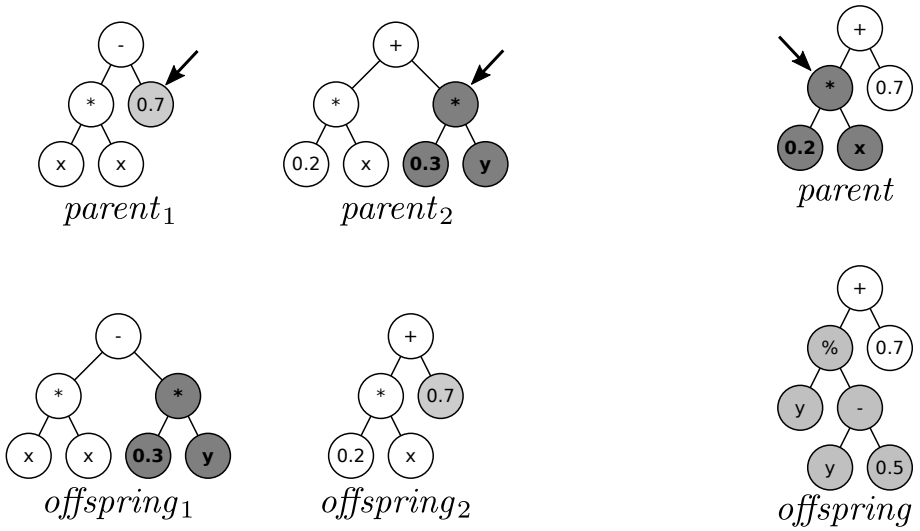
2.5 Genetic Operators

The quality of the solutions generated during the initialization procedure is, in general, very low. In order to transform and possibly improve these solutions, GP employs different genetic operators to modify them during the evolutionary process. There are several operators presented in the literature, including the permutation, editing and encapsulation operators [Koza, 1992a]. However, usually GP adopts only three of these operators, namely crossover, mutation and reproduction [Banzhaf et al., 1998].

The canonical crossover operator creates two offspring by interchanging the genetic material of two selected parents. A subtree of each parent is randomly selected and swapped between them, resulting in two (possibly) different individuals. The traditional GP mutation operator, on the other hand, creates only one offspring by picking a random subtree of a selected parent and replacing it with a new randomly generated subtree. Figure 2.3 describes graphically both operators.

The third genetic operator commonly applied in GP, the reproduction operator, is different from the others in the sense that it does not perform any modifications on the selected parents. Reproduction generates one offspring by copying the parent to the next population with no change.

Genetic operators are usually applied with user defined probabilities in a mutual exclusive strategy. Typically, crossover is applied with high probability, often around 90%, and mutation is applied with low probability, often around 1%. The remainder probability, i.e, to complete 100%, corresponds to the probability of applying the reproduction operator [Poli et al., 2008]. There are other parameters that control different aspects of the genetic operators and GP in general, as for instance the maximum al-



(a) Crossover operator

(b) Mutation operator

Figure 2.3: Example of application of crossover and mutation operators. The arrows point to the subtrees selected for crossover/mutation.

lowed depth for the resulting offspring and the probabilities of choosing a terminal or a function node while applying crossover and mutation operators. Nevertheless, many of them depend of the GP implementation and will not be discussed in this chapter.

In the next chapter we present a very promising research topic in GP, involving the use of semantic information within the evolutionary process.

Chapter 3

Semantics in Genetic Programming

The individuals evolved by GP represent computer programs, i.e., they are symbolic structures designed to attain a desired behaviour, such that given a specific input, they produce a certain output [Krawiec and Pawlak, 2013b]. Differently from other EC techniques, such as genetic algorithms, where the mapping genotype-phenotype is usually straightforward, in GP the structure of the individual (the program syntax) is separated from its effect (the program semantics) by multiple levels of abstraction. Consequently, even a slight modification on the structure of the individual program may lead to enormous changes on its semantics and, consequently, on its fitness. This makes GP fitness landscapes usually chaotic [Galván-López et al., 2010; Krawiec, 2011].

In order to overcome this problem, in the last few years a variety of works has emerged attempting to introduce semantic awareness to the evolutionary process (see [Vanneschi et al., 2014a] for a comprehensive survey of these studies). These works have as a common principle to analyse the output values generated by the programs (or fragments of the programs) represented by the individuals [Krawiec and Pawlak, 2013a], which is considered as their semantics. The semantic information can be explored in an attempt to increase GP locality, once semantics is directly related to the phenotype of the individuals. A formal definition of semantics, adopted in this thesis, is presented in the next section.

3.1 Definition

Syntax and semantics are concepts common to the study of programming languages. The syntax refers to the form of the expressions, statements, and program units of the language, and the semantics refers to the meaning of those expressions, statements, and program units. Despite the fact that syntax and semantics are usually separated

for discussion and study, they are closely related and, in well-designed languages, the semantics should follow straight from the syntax [Sebesta, 2012].

In the context of GP, these terms have similar meaning, i.e., syntax refers to the structure of the program represented by an individual and semantics describes the behaviour of this structure. However, for the purpose of this thesis, we need a more precise definition of semantics, since there are different ways of computing it.

In this thesis we adopt the formal definition of semantics used in the work from Vanneschi et al. [2014a], defined in the context of supervised learning. Supervised learning can be formally defined as follows [Murphy, 2012]:

Definition 1. *Given a finite set of input-output pairs representing the training cases defined as* $T = \{(\mathbf{x_1}, y_1), (\mathbf{x_2}, y_2), \ldots, (\mathbf{x_n}, y_n)\}$*, where* $(\mathbf{x_i}, y_i) \in \mathcal{X} \times \mathcal{Y}$ *(*$i = 1, 2, \ldots, n$*),* $\mathcal{X}$ *is the input space and* $\mathcal{Y}$ *is the output space, supervised learning consists in inducing a model* $p : \mathcal{X} \rightarrow \mathcal{Y}$ *that maps inputs to outputs, such as* $\forall(\mathbf{x_i}, y_i) \in T : p(\mathbf{x_i}) = y_i$*.*

In this thesis we are interested in a particular type of supervised learning called symbolic regression, where $\mathcal{X} \subseteq \mathbb{R}^d$, $\mathcal{Y} \subseteq \mathbb{R}$ and the model p is induced by GP[1]. From Definition 1 follows the definition of semantics[2]:

Definition 2. *Let* $in = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n)$ *and* $out = [y_1, y_2, \ldots, y_n]$ *be the input tuple and output vector, respectively, associated with the set of training cases* T*. Let* p *be the (program) function represented by an individual. The semantics of* p *is given by the vector of outputs it produces when applied to* in*, defined as* $s(p) = [p(\mathbf{x}_1), p(\mathbf{x}_2), \ldots, p(\mathbf{x}_n)]$*.*

Notice that this definition does not cover all the possible uses of semantics presented so far in the literature. For instance, Beadle and Johnson [2008, 2009a,b] adopted a canonical representation of the behaviour of Boolean programs, known as Reduced Ordered Binary Decision Diagrams (ROBDD) [Bryant, 1986], to check for semantic equivalence between individuals. This approach was extended to the symbolic regression domain by Beadle [2009], using ordered lists of powers, variables and coefficients. Johnson [2007], on the other hand, uses model checking—a technique for confirming that a program satisfies a given specification—to represent the semantics of the evolved individuals and quantify their fitness.

However, the definition adopted in this thesis has a few advantages [Vanneschi et al., 2014a]: (i) the semantics of an individual can be represented as a point in an n-dimensional metric space $\mathcal{S}$ (called semantic space) [Krawiec and Pawlak, 2013b],

[1]The models evolved by GP are interchangeably referred as functions, programs and individuals.

[2]The notation $s(p)$ for the semantics of a program p is extensible to the semantics of a population of programs $P = \{p_1, p_2, \ldots, p_{|P|}\}$, expressed by the set $s(P) = \{s(p_1), s(p_2), \ldots, s(p_{|P|})\}$.

(ii) when the fitness of an individual is calculated by running it with all the fitness cases, its semantics is automatically computed, and (iii) this definition is closely bound to the fitness function, which commonly measures the divergence between the target output and the output generated by the individual.

In the next section we present the taxonomy adopted in this thesis to classify methods that incorporate semantic awareness into GP, hereafter referred to as semantic GP methods.

3.2 Taxonomy

Different authors have proposed distinct methodologies to organize semantic GP methods. Nguyen [2011] and Nguyen et al. [2013] separate the approaches for representing, extracting, and using semantics in GP into three categories. *Grammar-based* methods comprise the approaches that incorporate semantic information into grammar-based GP, including attribute and logic grammars; *formal methods* incorporate information about the specification, development and verification of programs into GP; and *GP s-tree representations* include methods for extracting semantics from expression trees.

Vanneschi et al. [2014a] complement the work from Nguyen and collaborators and analyse the state-of-the art in the field, organizing the existing methods into three distinct categories. The first category comprises methods that work with *diversity*, mostly at the population level, including semantic diversity; the second class covers the *indirect* methods that operate on the syntax of the individuals and rely the survival criteria on semantic information; and the third category comprises the *direct* methods, that act straight in the individual semantics through the use of precise genetic operators.

Pawlak [2014] proposes another taxonomy, limited to the semantic crossover operators for tree-represented GP, and divides the related works into two categories: one including methods that use program semantics for providing *effective* changes, and other comprising the operators that use program semantics for exploiting *geometric* properties of the search space.

This thesis adopts a taxonomy inspired by the classification proposed by Pawlak [2015] to organize semantic GP methods into two categories: one including works that exploit *geometric* properties of the semantic space and one covering methods that operate on semantic diversity and locality, referred in this thesis as *non-geometric* methods. Given that this thesis focuses on semantic GP methods presenting geometric properties on the semantic space, the taxonomy presented by Pawlak [2015] allows us to isolate the *geometric* methods from the remainder semantic GP literature. The

next sections present an overview of both categories—*geometric* and *non-geometric* methods.

3.2.1 Non-Geometric Semantic Methods

The category of non-geometric methods encompasses the semantic GP methods that do not make use of geometric properties of the semantic space. It includes the diversity and indirect methods described in the survey of Vanneschi et al. [2014a], with exception of those that work directly in the geometry of the landscape (presented in section 5.3 of their survey).

One of the first efforts in the field of semantic GP refers to the works promoting semantic diversity at the population level. In order to analyse the effect of increasing semantic diversity in the initial population on the search process, Beadle and Johnson [2009a] and Beadle [2009] present the Semantically Driven Initialization (SDI) algorithm, a population initialization method that adopts a domain-dependent canonical representation for the behaviour of programs to generate individuals semantically different. Experimental analysis showed the effectiveness of SDI in comparison to classical syntactic initialization methods. Jackson [2010a], on the other hand, adopts a trial and error approach to achieve semantically different individuals in the initial population and analyses the effects of diversity under fitness, semantic and structural viewpoints. The experimental results indicate semantic diversity in the initial population improved solution-finding performance.

Jackson [2010b] extends his earlier work and analyses the effects of preserving the semantic diversity throughout the evolution. The author uses a trial and error crossover operator to generate offspring semantically different from their parents and increase the semantic diversity along the generations. The results point out that the new operator, allied to semantic diversity in the initial population, can improve even further the results from his early work. Beadle and Johnson [2008] and Beadle [2009] also consider the semantic information during crossover, but apply the same canonical representation used in SDI to check for semantic equivalence in the context of Semantically Driven Crossover (SDC). The same strategy is adopted in the context of the Semantically Driven Mutation (SDM) operator [Beadle and Johnson, 2009b]. Results presented in the respective publications indicate that SDC and SDM can increase the performance (in terms of error) of GP on the adopted test beds.

In this same direction, Castelli et al. [2013c] propose a semantic-based algorithm that keeps a distribution of different semantics to drive GP to search in areas of the semantic space where previous good solutions were found. The method outperformed

standard GP and bacterial GP [Botzheim et al., 2007] in the test bed adopted. However, the individuals generated by the method presented statistically larger sizes than the individuals generated by the other two GP variants.

Another line of research in semantic GP involves the role of locality, i.e., how well neighbouring genotypes correspond to neighbouring phenotypes [Galván-López et al., 2010]. Inspired by Semantic Aware Crossover (SAC) [Nguyen et al., 2009c], Nguyen et al. [2010] present the Semantic Similarity based Crossover (SSC) in an attempt to increase the locality in the canonical tree-swapping crossover. Before exchanging the two selected subtrees, SSC verifies their similarity by comparing them on a set of random points in the domain of interest. If the mean of the sum of the differences between the outputs of these subtrees on the set of random points is within an interval, then the subtrees are considered semantically similar and are exchanged. The method was tested in polynomial functions and the empirical results showed that SSC helps improving the performance of GP in comparison with SAC and the canonical GP, both in terms of percentage of successful runs and average of best fitness over a number of runs.

Nguyen et al. [2013] extend their earlier work [Nguyen et al., 2010], further exploring the role of semantic locality in GP crossover and proposing improvements to the SSC operator. Experimental analysis showed that semantic locality is more beneficial in GP crossover than syntactic locality and is an important property to be explored by GP crossover operators.

Also inspired by SAC, Nguyen et al. [2009a] propose a mutation operator based on semantic similarity, called Semantic Similarity based Mutation (SSM). Experimental results demonstrate that SSM improved locality when compared to the standard mutation operator, reaching superior performance.

Although non-geometric semantic methods have shown improvements in relation to semantic-blind methods, they still rely on syntactic manipulations unaware of the semantics to attain the desired semantic properties.

3.2.2 Geometric Semantic Methods

Moraglio and collaborators [Moraglio and Poli, 2004; Moraglio, 2007, 2011] defined mutation and crossover operators within a geometric framework. Let $\mathcal{P}$ denote the solution set comprising all the possible candidate solutions to a given search problem and let $d : \mathcal{P} \times \mathcal{P} \rightarrow \mathbb{R}^*$ be a distance function on $\mathcal{P}$. A closed ball is defined as the set $B(x; r) = \{z \in \mathcal{P} \mid d(x, z) \leq r\}$, where $x \in \mathcal{P}$ and $r \in \mathbb{R}^+$ is the radius of the ball. A line segment (or closed interval) is the set of the form $[x; y] = \{z \in \mathcal{P} \mid$

$d(x,z) + d(z,y) = d(x,y)\}$, where $x, y \in \mathcal{P}$ are called extremes of the segment. The geometric crossover and mutation operators are defined as:

Definition 3. *A binary search operator $CX : \mathcal{P} \times \mathcal{P} \to \mathcal{P}$ is a geometric crossover if $\forall (p_1, p_2) \in \mathcal{P} \times \mathcal{P}$, $CX(p_1, p_2) \in [p_1; p_2]$.*

Definition 4. *A unary search operator $M : \mathcal{P} \to \mathcal{P}$ is a geometric ε-mutation operator if $\forall (p) \in \mathcal{P}$, $M(p) \in B(p; \varepsilon)$.*

The term *geometric semantic* in the sense of geometric operators is commonly used to refer to GP works related to Definitions 3 and 4, as for instance operators that approximate the geometric characteristic [Krawiec and Lichocki, 2009; Krawiec and Pawlak, 2012a, 2013a,b; Pawlak et al., 2014; Pawlak, 2014] or that present the exact behaviour of geometric operators [Moraglio et al., 2012]. However, in this thesis the term *geometric semantic* is less restrictive, covering works not related to geometric operators, as for instance the works from Ruberto et al. [2014] and Castelli et al. [2015f] that derive the concept of error space from the semantic space, to define geometric relations between the individuals evolved by GP.

One of the first geometric semantic methods was the Approximately Geometric Semantic Crossover (SX and SX+) proposed by Krawiec and Lichocki [2009]. Given two parents, p_1 and p_2, the SX operator applies k times a syntactic crossover operator—e.g. the tree-swapping crossover—generating a pool C of new offspring, and returns the offspring that minimizes the expression:

$$\arg\min_{c \in C} \; d(s(p_1), s(c)) + d(s(p_2), s(c)) \;\; , \tag{3.1}$$

where $d(a, b)$ is a distance metric between a and b and $s(\cdot)$ follows from Definition 2. The SX+ is a SX variation with a penalty term that rewards offspring equidistant or close-to-equidistant from both parents. The minimization performed by SX+ is given by:

$$\arg\min_{c \in C} \; d(s(p_1), s(c)) + d(s(p_2), s(c)) + \underbrace{\mid d(s(p_1), s(c)) - d(s(p_2), s(c)) \mid}_{\text{penalty term}} \;\; . \tag{3.2}$$

Pawlak [2014] extends SX+ by removing the offspring that are semantically equal to any of the parents, i.e., the set C from Equation 3.2 is replaced by its subset O, computed by Equation 3.3. To the best of our knowledge this is the first attempt

to hybridize a geometric approach (SX+) and a non-geometric one (the removal of offspring semantically equivalent to their parents).

$$O = \begin{cases} O' \ , & \text{if } O' \neq \emptyset \\ C \ , & \text{otherwise} \end{cases} \tag{3.3}$$
$$\text{where } O' = \{o : o \in C, s(o) \neq s(p_1), s(o) \neq s(p_1)\}$$

Moraglio et al. [2012] proposed the first geometric search operators for GP, in the context of the Geometric Semantic Genetic Programming (GSGP), with the introduction of geometric semantic crossover and mutation operators. Given the contribution of GSGP to the field and, particularly, to this thesis, it will be explained deeply in Section 3.3.

The Locally Geometric Semantic Crossover (LGX) [Krawiec and Pawlak, 2012a, 2013b] recombines homologous subtrees in the parents by finding semantically intermediate programs from a library, expecting that these semantic changes propagate towards the tree root and induce an approximately geometric comportment in the offspring. Given two previous selected parents, p_1 and p_2, the method finds the common region of both parents—i.e., the set of loci occurring in both individuals—and randomly chooses a homologous locus inside the common region. Then, it selects the subtrees rooted in the drawn locus in p_1 and p_2—p'_1 and p'_2, respectively—and computes the midpoint between p'_1 and p'_2 in the semantic space according to the problem domain. Equation 3.4 presents the formula to compute the midpoint s_m for the regression domain.

$$s_m = \frac{s(p'_1) + s(p'_2)}{2} \tag{3.4}$$

The midpoint s_m is the semantics of a hypothetical offspring p, perfectly geometric regarding p_1 and p_2. Then, LGX searches in a library of previously prepared program trees for the program that minimizes the semantic distance to s_m, replaces the previous selected subtrees by the found procedure and returns the resulting offspring. Figure 3.1 depicts an example of the functioning of LGX.

Krawiec and Pawlak [2012b] present a deep analysis regarding the LGX, covering aspects related to search progress, size of the procedure library, importance of homology, impact on the individual tree size, generalization and time complexity. The empirical study indicated that the homologous and semantic awareness features present in LGX can improve GP search convergence and reduce the error in the test set.

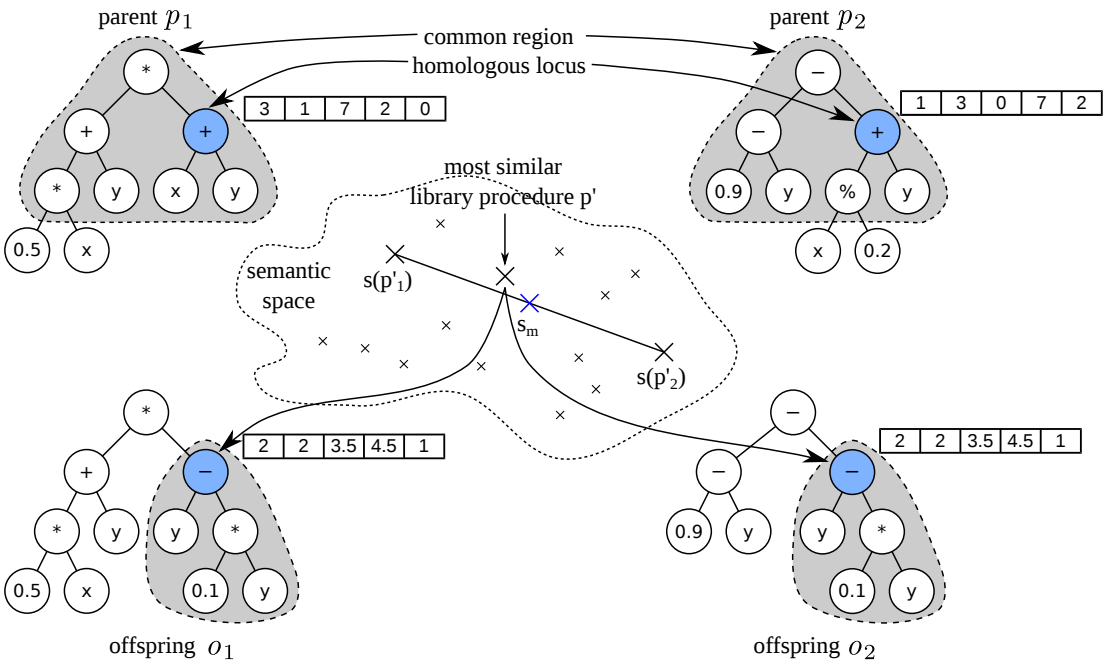


Figure 3.1: Functioning principle of LGX. The vectors represent the subtrees' semantics before and after the application of LGX and the blue nodes represent the roots of the subtrees (p'_1 and p'_2). Adapted from [Krawiec and Pawlak, 2013b].

The Approximately Geometric Crossover (AGX) [Krawiec and Pawlak, 2013a; Pawlak et al., 2014] also employs a library of program trees to approximate the semantics, but in a different context. The method replaces subtrees in the parents by programs from the library in a way that the semantics of the offspring lays in the middle of the segment connecting the parents' semantics. Given two parents, p_1 and p_2, and their respectively selected subtrees, p'_1 and p'_2, the method uses the concept of semantic backpropagation to compute the expected semantics of p'_1 and p'_2 such that the semantics of the respective root node corresponds to the midpoint between p_1 and p_2 in the semantic space. The midpoint is computed by the same equation (3.4) used in the LGX w.r.t. the parents' semantics.

Pawlak et al. [2015] present a formal and empirical analysis regarding multiple geometric crossover operators, namely SX+, LGX, AGX and the geometric semantic crossover from GSGP. The study regards performance, measured by the best-of-generation fitness, generalization, computational cost, bloat, degree of geometricity[3] and diversity, considering a test bed composed of nine datasets in the symbolic regression domain and nine in the Boolean domain. The main conclusion of the study is that

[3]Pawlak et al. [2015] define geometric offspring with respect to parents p_1 and p_2, as the individual o resulting from an application of a crossover operator such that $d(p_1, p_2) = d(p_1, o) + d(p_2, o)$.

there is no better operator for all cases, i.e., each operator has its own advantages and drawbacks.

Ruberto et al. [2014] explores the geometry of the semantic space through the concept of error vector. Let $s(p)$ be the semantics of the individual p and *out* the output vector (Definition 2). The error vector is represented by a point in the n-dimensional space, called error space, given by the translation $t_e(p) = s(p) - out$. This notion is used to introduce the concept of optimally aligned individuals in the error space—i.e., given a number of dimensions $\mu = 1, 2, ..., n$, where n is the size of the training set, μ individuals are optimally aligned in the error space if they belong to the same μ-dimensional hyperplane intersecting the origin of the error space. The authors show that if μ individuals are optimally aligned, we can obtain an equation analytically to express the desired output vector *out*. In this context, they present GP based methods to find optimally aligned individuals in two and three dimensions, called ESAGP-1 (Error Space Alignment GP) and ESAGP-2, respectively. Experimental results suggest that searching for optimally aligned individuals (in two and three dimensions) is easier than directly searching for a globally optimal solution.

Castelli et al. [2015f] extends the ESAGP-1 by means of the Pair Optimization GP (POGP). Unlike the original method, which represents individuals as simple expressions and computes the fitness by the angle between the error vector of an individual and a particular point called attractor, POGP represents the individuals as pairs of expressions, and calculates the fitness as the angle between the error vectors of these two expressions. Although POGP is presented in a preliminary study, experimental results indicate that the method deserves more attention in future studies.

3.3 Geometric Semantic Genetic Programming

The GSGP proposed by Moraglio et al. [2012] introduces a new class of genetic operators in GP that, acting on the syntax of the parent programs, produce offspring that are guaranteed to respect some semantic criterion by construction. These operators guarantee that the semantic fitness landscape explored by GP is conic, a property with positive effects on the search process. Moraglio [2011] presents formal evidence that indicates evolutionary algorithms with geometric operators can optimise cone landscapes with good results for virtually any metric.

Since the method was proposed it has been successfully applied in different domains, e.g., modelling of the behaviour of different pharmacokinetics parameters [Vanneschi et al., 2013, 2014b], prediction of high performance concrete strength

[Castelli et al., 2013b], multiclass classification involving land cover/land use applications [Castelli et al., 2013a], prediction of energy performance of residential buildings [Castelli et al., 2015c], forecasting energy consumption [Castelli et al., 2015b,e], prediction of the Unified Parkinson's Disease Rating Scale (UPDRS) assessment [Castelli et al., 2014b], prediction of burned areas resulting from forest fires [Castelli et al., 2015d] and application in maritime awareness [Vanneschi et al., 2015].

GSGP operators are defined based on the concept of topological operators [Moraglio and Poli, 2004] and can be directly implemented for different domains following a simple formal recipe [Moraglio et al., 2012]. The next section presents the mathematical formalism behind the method and implementations of GSGP operators for the arithmetic domain.

3.3.1 Implementation of the Operators

Let the fitness function f be the distance d between the semantics of an individual p and the target output of the associated programming task, as presented by Equation 3.5. Given two functions, $p_1, p_2 \in P$, the semantic distance between them $d_s : P \times P \to \mathbb{R}^*$ is the distance between their corresponding output vectors measured by a metric d, i.e., $d_s(p_1, p_2) = d(s(p_1), s(p_2))$ [Moraglio and Mambrini, 2013]. Moraglio et al. [2012] derived specific forms of geometric operators from Definitions 3 and 4 for the space of functions P, endowed with the semantic distance d_s, called geometric semantic crossover and mutation operators.

$$f(p) = d(s(p), out) \tag{3.5}$$

Moraglio et al. [2012] define these operators for the Boolean and program domains with fitness functions based on the Hamming distance and for the arithmetic domain with fitness function based on the Euclidean and Manhattan distances. Due to the scope of this thesis, here we present only the definitions for the arithmetic domain, which corresponds to the symbolic regression problem. The other definitions can be found in the original work [Moraglio et al., 2012].

Let $\mathcal{P}$ be the solution set comprising all the possible (GP representable) candidate solutions to a problem in the arithmetic domain, the geometric semantic operators are defined as follows:

Definition 5. *Given two parent functions, $p_1, p_2 \in \mathcal{P}$, the Geometric Semantic Crossover for fitness function (Eq. 3.5) based on Euclidean distance, $GSXE : \mathcal{P} \times \mathcal{P} \to$*

$\mathcal{P}$, returns the offspring arithmetic function

$$o(\mathbf{x}) = r \cdot p_1(\mathbf{x}) + (1 - r) \cdot p_2(\mathbf{x}) \ , \tag{3.6}$$

where r is a random real constant in $[0, 1]$; and the Geometric Semantic Crossover for fitness function based on Manhattan distance, $GSXM : \mathcal{P} \times \mathcal{P} \to \mathcal{P}$, returns the offspring arithmetic function

$$o(\mathbf{x}) = r(\mathbf{x}) \cdot p_1(\mathbf{x}) + (1 - r(\mathbf{x})) \cdot p_2(\mathbf{x}) \ , \tag{3.7}$$

where $r(\mathbf{x})$ is a real function randomly generated, with codomain $[0, 1]$.

Definition 6. *Given a parent function, $p \in \mathcal{P}$, the Geometric Semantic Mutation $GSM : \mathcal{P} \times \mathbb{R}^+ \to \mathcal{P}$ with mutation step $ms \in \mathbb{R}^+$ returns the real function*

$$o = p(\mathbf{x}) + ms \cdot (r_1(\mathbf{x}) - r_2(\mathbf{x})) \ , \tag{3.8}$$

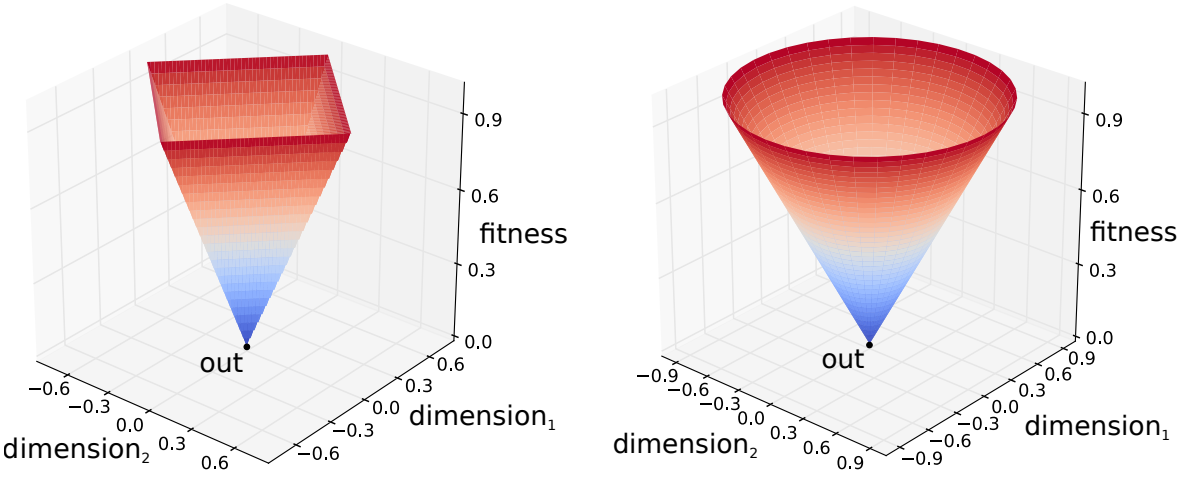
where $r_1(\mathbf{x})$ and $r_2(\mathbf{x})$ are real functions randomly generated.

The functions $r(\mathbf{x})$—from the GSXM operator—and $r_1(\mathbf{x})$ and $r_2(\mathbf{x})$—from the GSM operator—are randomly generated from the function and terminal sets available for the GSGP. Usually, these functions are created with population initialization methods—e.g., grow and full methods [Koza, 1992a]—at each application of these operators.

The original definition of GSM does not make any assumptions about the codomain of $r_1(\mathbf{x})$ and $r_2(\mathbf{x})$. However, the lack of bounds may generate individuals extremely distant from the parent, with negative impact on the performance of GSGP, regarding the test error [Castelli et al., 2015a; Gonçalves et al., 2015]. Thus, following the work of Castelli et al. [2015a], we assume that $r_1(\mathbf{x})$ and $r_2(\mathbf{x})$ have codomain $[0, 1]$.

3.3.2 Geometric Implications

The fitness at any point in the semantic space is equivalent to the semantic distance between the point and the target output vector (*out*), as presented by Eq. 3.5. Thus, the fitness function spanning the semantic space has a conic shape and, consequently, is unimodal (considering that the fitness function has only one global minimum, at *out*) and does not feature plateaus. This statement holds for any semantic space, any data type associated with the target output vector and any metric [Pawlak et al., 2015].



(a) Manhattan distance (b) Euclidean distance

Figure 3.2: Conic shapes of fitness functions under the Euclidean and Manhattan distances. In this example, the desired output *out* corresponds to the origin.

Figure 3.2 presents fitness functions spanning a two dimensional semantic space under the Euclidean and Manhattan distances.

However, notice that the semantic space $\mathcal{S}$ is not the space directly searched, since the geometric semantic operators manipulate programs, through the operations presented in Definitions 5 and 6. The semantic space is searched only implicitly [Pawlak et al., 2015]. Nevertheless, these operators hold well defined geometric properties in $\mathcal{S}$:

- The offspring generated by $GSM(p, ms)$ is placed inside the closed ball $B(p; \varepsilon)$, centred in $s(p)$, with radius ε and expected value equal to the semantics of the parent function[4]. The radius ε is defined as the smallest real for which this condition holds true [Moraglio et al., 2012].

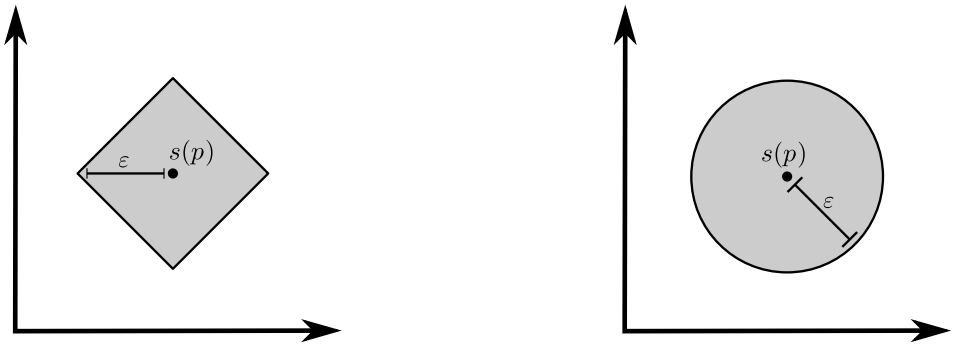
  - Given the restriction that $r_1(\mathbf{x}), r_2(\mathbf{x}) \in [0, 1]$, we can define the value of ε that ensures $GSM(p, ms)$ is inside $B(p; \varepsilon)$, according to the metric used in the semantic space:

$$\varepsilon = \left(\sum_{i=1}^{n} ms^{\gamma} \right)^{1/\gamma} \tag{3.9}$$

    where n is the dimensionality of $\mathcal{S}$ and γ is the order of the Minkowski distance[5]—e.g., $\gamma = 1$ for Manhattan and $\gamma = 2$ for Euclidean. Figure

[4] The statement about the expected value of $GSM(p, ms)$ holds if the random real functions employed by the operator—$r_1(\mathbf{x})$ and $r_2(\mathbf{x})$ in Eq. 3.8—have the same expected values. However, this is not always possible in practice.

[5] The Minkowski distance corresponds to a class of distance functions, being a generalization of both Manhattan and Euclidean distances [Basilevsky, 2005].



(a) (b)

Figure 3.3: Geometric representation of the geometric semantic mutation operator in two-dimensional semantic spaces defined by (a) Manhattan and (b) Euclidean metrics. The resulting offspring is placed inside the closed ball $B(p;\varepsilon)$, where $\varepsilon = 2 \times ms$—for Manhattan metric—and $\varepsilon = \sqrt{2} \times ms$—for Euclidean distance.

3.3 depicts $B(p;\varepsilon)$ for two-dimensional semantic spaces for Manhattan and Euclidean metrics.

- Let $o_E(x) = GSXE(p_1, p_2)$ and $o_M(x) = GSXM(p_1, p_2)$:
  - By construction, the semantics of o_E corresponds to a convex combination[6] of the semantics of p_1 and p_2 and is located on the line segment connecting $s(p_1)$ to $s(p_2)$ in $\mathcal{S}$ (see Figure 3.4b). Consequently, the fitness of o_E—$f(o_E)$—is upper bounded by $max(f(p_1), f(p_2))$ [Pawlak et al., 2015]. This property holds for the fitness calculated regarding both the training set and the test set.
  - The semantics of o_M is located inside a n-dimensional hyperrectangle[7] with vertices in $s(p_1)$ and $s(p_2)$ (see Figure 3.4a). Unlike o_E, $max(f(p_1), f(p_2))$ is not an upper bound for $f(o_M)$.

3.3.3 GSGP Limitations

In this section we present some limitations regarding GSGP that restrain its capability of finding the optimal solution to the problem, even with the conic fitness landscape induced by the geometric operators.

[6] A convex combination of points $v_1, v_2, \ldots, v_n \in \mathbb{R}^n$ is a combination in the form $\alpha_1 \cdot v_1 + \alpha_2 \cdot v_2 + \ldots + \alpha_n \cdot v_n$ such that $\alpha_1 + \alpha_2 + ... + \alpha_n = 1$ and $\alpha_i \geq 0$ for $i = 1, 2, \ldots, n$ [Lay, 2012].

[7] The hyperrectangle is the equivalent in the Euclidean space of the line segment in the Manhattan space.

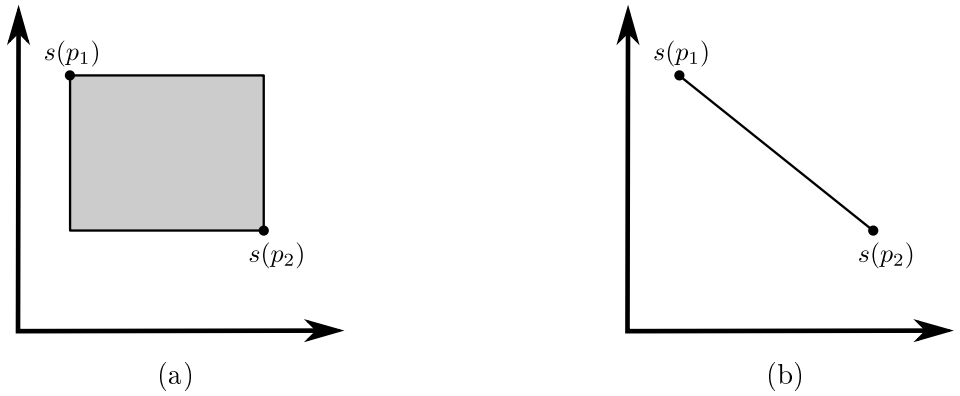


Figure 3.4: Geometric representation of the geometric semantic crossover operators in a two-dimensional semantic space. The offspring resulting from the geometric semantic crossover for fitness function based on Manhattan and Euclidean distances are placed, respectively, (a) inside the hyperrectangle with vertices in $s(p_1)$ and $s(p_2)$ and (b) on the line segment connecting the semantics of the parents.

3.3.3.1 Individual Exponential Growth

The geometric semantic mutation and crossover operators proposed by Moraglio et al. [2012] experience, respectively, linear and exponential growth in the size of the individuals over the generations. By construction, the offspring resulting from GSGP crossover operator are composed of the combination of two parents (p_1, p_2 in Definition 5). Consequently, its offspring have size as big as the sum of the sizes of their parents, leading to the exponential growth. On the other hand, the offspring resulting from GSGP mutation are composed of only one parent (p in Definition 6) combined with two random functions (r_1, r_2 in Definition 6), leading to linear growth of the offspring proportional to the sum of the sizes of the two random functions.

Equations 3.10, 3.11 and 3.12 present the expected number of nodes in a program tree in a GSGP that applies only *GSXE*, *GSXM* and *GSM*, respectively, to generate all offspring [Pawlak, 2015]. The expected number of nodes in the individuals of the initial population is given by $E[P_0]$, $g > 0$ denotes the current generation (the initial population corresponds to generation 0 and is not computed by the equations), $E[r_x]$ is the expected number of nodes in the random functions generated by the geometric semantic operators and a, b and c are the number of additional nodes (constant) used to implement *GSXE*, *GSXM* and *GSM*, respectively.

$$E[GSXE_g] = 2^g \cdot E[P_0] + (2^g - 1) \cdot a \tag{3.10}$$

$$E[GSXM_g] = 2^g \cdot E[P_0] + (2^g - 1) \cdot (E[r_x] + b) \tag{3.11}$$

$$E[GSM_g] = E[P_0] + g \cdot (2 \cdot E[r_x] + c) \tag{3.12}$$

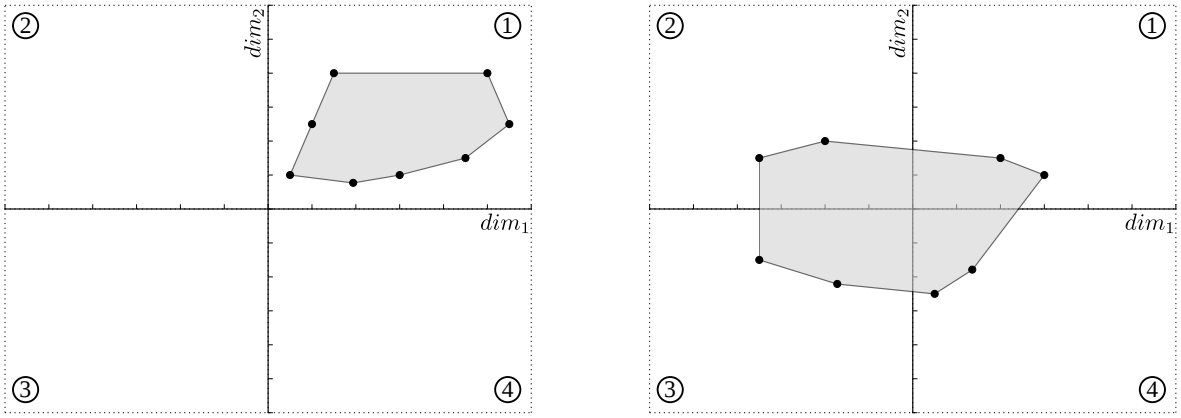
The exponential growth of the individuals in the population due to the use of these operators (particularly the geometric semantic crossover) makes GSGP unmanageable in practice: after a few generations the size of the individuals reaches intractable levels, demanding excessive memory and computational power. Thus, the literature of GSGP based methods, including this thesis, usually does not concern with interpretability of the resulting solutions.

Moraglio et al. [2012] suggest the syntactic simplification of the expressions without changing the computed function, which can be done at any moment and in any amount. However, this process increases the computational cost inherent to GSGP and only partially solves the problem of size growth.

Vanneschi et al. [2013, 2014b] propose a new implementation of these operators for symbolic regression to reduce memory consumption and computational time, here called EGSGP (Efficient GSGP). In this implementation, the individuals of the initial population and the random functions generated by GSGP operators, both represented as trees, are stored in memory, such that the subsequent individuals are composed of pointers to these structures. The semantics of each individual is also stored in memory and updated every time an operator is applied to the population, which reduces computational effort to calculate the fitness. However, the solutions are never truly built during the evolution, i.e., if we need to analyse the resulting solutions or apply them to new data, it is necessary to reconstruct the expressions, which still exhibit exponential size.

Moraglio [2014] presents a Python implementation of the GSGP where higher-order functions and memoization are used instead of pointers and data structures, delegating the control to the compiler. The solutions are represented directly as compiled Python and present exponential size when decompiled.

Zhu et al. [2013] present a mechanism based on the biological concept of devolution to reduce the complexity of the solutions generated by the Adapted Geometric Semantic (AGS) operators. However, although AGS operators are based on GSGP, no considerations about their geometric properties are made by the authors.



(a) P_c: population concentrated into a single quadrant.

(b) P_s: population encompasses solutions in all quadrants.

Figure 3.5: Different distributions of a population in two-dimensional semantic space. The desired output *out* is located on the origin of the space and the circled numbers indicate the quadrants.

3.3.3.2 Initial Population

By definition, GSXE and GSXM are geometric crossover operators [Moraglio et al., 2012]. This property leads us to the following theorem regarding the convex hull of the population[8]:

Theorem 1. *Let P_g be the population at generation g. For a GSGP, where the only search operator available is one of the geometric semantic crossover operators (GSXM or GSXE), we have $\mathcal{C}(s(P_{g+1})) \subseteq \mathcal{C}(s(P_g)) \subseteq \ldots \subseteq \mathcal{C}(s(P_1)) \subseteq \mathcal{C}(s(P_0))$.*

Theorem 1 is a particular case of Theorem 3 defined and proved by Moraglio [2011], and has an important implication regarding the geometric semantic crossover operators. Given a population P and the desired semantic output *out* in $\mathcal{S}$, the offspring resulting from the application of GSXM or GSXE to any pair of individuals in P can reach *out* if and only if $out \in \mathcal{C}(s(P))$. Consequently, if GSGP has no other search operators (only one of the geometric semantic crossover operators), the output vector *out* is reachable if and only if $out \in \mathcal{C}(s(P_0))$, i.e., if *out* is located inside the convex hull of the initial population. Thus, the initialization method adopted by the GSGP has direct impact on the search performance, even if other genetic operators are considered during the evolution.

[8]Let P be a population of individuals, we adopt the notation $\mathcal{C}(s(P))$ to denote the convex hull of the set composed of the semantics of the individuals of P, i.e., $s(P)$.

Figure 3.5 illustrates this situation for a two-dimensional Euclidean[9] semantic space. Let $out = [0, 0]$ be the desired output vector defined by the training cases. Now consider two different populations P_c and P_s, where the individuals from P_c are concentrated in the first quadrant and, consequently, $\mathcal{C}(s(P_c))$ cannot reach the origin (out). On the other hand, the set $s(P_s)$ is distributed along the four quadrants and $\mathcal{C}(s(P_s))$ embraces the desired vector. In the first scenario, GSGP needs the mutation operator to expand the convex hull to reach the solution. In the second scenario, as it is already inside the convex hull, the desired vector can be found using the crossover operator alone or it can be calculated analytically[10] with no need to use GSGP.

Pawlak [2015] presents the Competent Initialization (CI) method in order to improve the initial convex hull. The algorithm adopts the same strategy of the SDI method [Beadle and Johnson, 2009a] to generate random individuals in a trial-and-error strategy. However, the new produced individual is added to the population only if it is not in the convex hull of the population generated so far. The main concern about this method is the waste of resources, once the individuals are randomly created and discarded when they are already in the population's convex hull.

Pawlak and Krawiec [2016], on the other hand, introduce the Semantic Geometric Initialization (SGI) method to generate the initial population taking into account the location of the target output, such that out is guaranteed to be inside the convex hull of the generated population. However, although SGI can achieve very small error in the training set, the solutions found by the method are prone to overfitting, presenting high error in the test set.

[9]The metric adopted in the space impacts on the convex hull shape. For instance, in Manhattan semantic spaces, the convex hull is always an n-dimensional hyperrectangle.

[10]The coefficients of the convex combination can be found by means of Gaussian elimination [Gentle, 1998].

Chapter 4

The Role of the Geometric Semantic Operators

The canonical genetic operators presented in Chapter 2 are one of the fundamental forces to promote the evolution of the solutions in GP [Eiben and Smith, 2003]. They have direct impact on the way the search space is explored and, consequently, on the quality of the generated solutions. Given their importance, several works in the literature investigate different aspects of these operators. For instance, there are a variety of studies that examine in details the mechanics of the subtree crossover [D'haeseleer, 1994; Luke and Spector, 1998; Langdon, 2000] and mutation [Angeline, 1996; Luke and Spector, 1998] operators.

The operators employed in the Geometric Semantic GP (GSGP) [Moraglio et al., 2012] also have a crucial role on the evolutionary process, searching the space of the underlying semantics of the programs. These operators respect some geometric properties that can be explored in order to improve the search process. This chapter investigates the role of the geometric semantic crossover and mutation operators in the GSGP search process, and is organized as follows. Section 4.1 presents a brief review of the studies covering these properties and other aspects related to the geometric semantic operators. In Section 4.2 we present a further investigation regarding the impact of the geometric semantic crossover operator on the search process and propose two geometric semantic crossover operators that optimally combine the parent individuals by minimizing the training error.

4.1 Geometric Semantic Operators in the Literature

Given the successful application of GSGP in different domains [Vanneschi et al., 2013, 2014b, 2015; Castelli et al., 2013a,b, 2014b, 2015b,c,d,e], several studies were performed to cover different aspects regarding the geometric semantic operators.

Castelli et al. [2014a] present a preliminary study where each individual has its own probability of crossover and mutation. The value of these probabilities is dynamically tuned during the generations, according to the quality of the offspring generated.

Regarding the Efficient GSGP (EGSGP) [Vanneschi et al., 2013, 2014b], reviewed in Section 3.3.3.1, Castelli et al. [2015a] present a C++ framework for the EGSGP implementation, with additional details regarding the geometric semantic crossover and mutation operators adopted. Following Definitions 5 and 6, the framework adopts a logistic wrapper for the random real functions to constrain their outputs to the interval $[0, 1]$, such that r, r_1 and r_2 are replaced by $logis(r)$, $logis(r_1)$ and $logis(r_2)$, respectively, and the logistic wrapper is given by:

$$logis(x) = \frac{1}{1 + e^{-x}} \ . \tag{4.1}$$

However, according to Dick [2015], there are some limitations on the use of the logistic wrapper to bound the outputs of the random trees generated within GSGP operators. For instance, the logistic function (Equation 4.1) approximates to 0 and 1 for $x \leq -5$ and $x \geq 5$, respectively—as presented in Figure 4.1—limiting the range of possible outputs. The author adopts interval arithmetic [Keijzer, 2003] to incorporate basic knowledge regarding the intervals of the problem's inputs and generate random trees for GSGP capable of producing a wider range of outputs. An experimental study comparing the new approach with GSGP, with and without the logistic wrapper, showed the interval arithmetic outperforms both baselines when tested on a range of problems.

Gonçalves et al. [2015] also analyse the effect of the logistic wrapper in the GSGP search process, but with emphasis on its impact on the geometric semantic mutation operator. The authors compare the influence of the mutation wrapped and not wrapped by the logistic function on GSGP performance with and without the crossover operator. They also propose two variants of the mutation operator for symbolic regression,

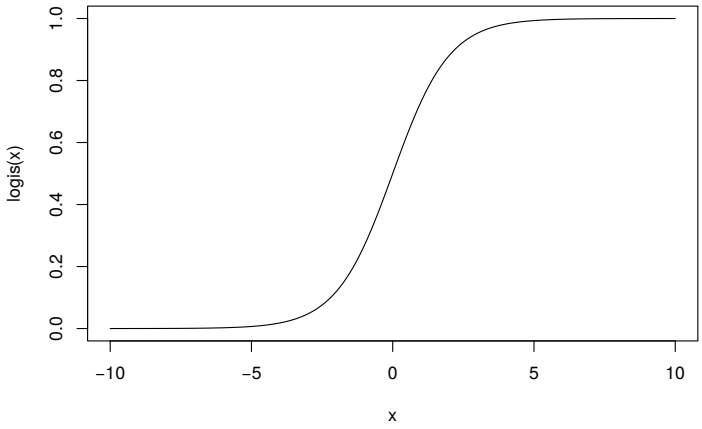


Figure 4.1: Logistic wrapper output in the interval $[-10, 10]$.

called Adaptive Mutation (AM) and Doubly Adaptive Mutation (DAM), presented in Equations 4.2 and 4.3, respectively.

$$AM(p) = p + ms \cdot (r_1 - r_2) \tag{4.2}$$

$$DAM(p) = pw \cdot p + ms \cdot (r_1 - r_2) \tag{4.3}$$

The AM operator solves the expression $ms \cdot (r_1 - r_2) = (out - p)$ and the DAM solves $pw \cdot p + ms \cdot (r_1 - r_2) = out$, where r_1, r_2 and p correspond to the semantics of the two random trees generated by the operator and the parent individual, respectively, calculated w.r.t. the training cases, out is the target output vector defined by the training set and $ms, pw \in \mathbb{R}$ are calculated deterministically and optimally by the application of the Moore-Penrose pseudoinverse.

Both AM and DAM operators are implemented within the Semantic Stochastic Hill Climber[1] (SSHC) [Moraglio et al., 2012] and compared with the canonical GP and different configurations of the GSGP, established to investigate the effect of these operators on two datasets. Among the conclusions, Gonçalves et al. [2015] state that (1) the mutation variants wrapped by the logistic function generalize well, while those without the wrapper overfit the training data, (2) AM and DAM quickly overfit the data, but achieve a competitive generalization in the first generation on the adopted test bed and (3) the geometric semantic crossover operator is unnecessary to the search, since it does not significantly improve search performance.

[1]SSHC is basically a stochastic hill climber that employs geometric semantic mutation to explore the neighbourhood.

In the next section we also present a study to analyse if the impact of the geometric semantic crossover operator is relevant to the search performed by the GSGP. However, our results indicate the operator is relevant to the search, contrary to the statement presented by Gonçalves et al. [2015]. We also present two geometric semantic crossover operators based on the principle of optimally calculating the weight of the individuals in the convex combination.

4.2 A Study of the Geometric Semantic Crossover

In this section we consider aspects related to Issue 2 presented in Section 1.1. As stated in Section 3.3.1, the fitness landscape induced by geometric semantic operators is unimodal, which may indicate that methods based on local decisions are sufficient for achieving good solutions and, hence, the geometric semantic crossover and mutation operators might have similar effects. In this section we present an experimental study to verify this hypothesis by comparing the GSGP with and without the crossover operator.

By construction (see Definition 5), the geometric semantic crossover for symbolic regression (arithmetic domain) with fitness function based on Euclidean distance is a convex combination of the parents p_1 and p_2. Equation 4.4 presents an alternative notation for the GSXE operator—presented in Eq. 3.6—with the restrictions $\beta_1, \beta_2 \in [0, 1]$ and $\beta_2 = 1 - \beta_1$.

$$GSXE(p_1(\mathbf{x}), p_2(\mathbf{x})) = \beta_1 \cdot p_1(\mathbf{x}) + \beta_2 \cdot p_2(\mathbf{x}) \tag{4.4}$$

However, the value of β_1 is randomly defined in the interval $[0, 1]$ and does not reflect the contribution of each parent in the training error. In our preliminary study [Albinati et al., 2014] and further analysis [Albinati et al., 2015], we present a formal basis to optimally and deterministically find the values of β_1 and β_2 that minimize the training error generated by the offspring produced by the crossover, resulting in two new operators, the Optimized Convex Euclidean-Based Geometric Semantic Crossover (GSXE-C) and the Optimized Non-Convex Euclidean-Based Geometric Semantic Crossover (GSXE-L), presented in the next sections. We also present an empirical analysis of the impact of these operators on the search process.

4.2.1 Optimized Convex Euclidean-Based Geometric Semantic Crossover

The GSXE-C operator was created by finding the value of β_1 in Equation 4.4 that leads to the minimum training error under the restrictions $\beta_2 = 1 - \beta_1$ and $\beta_1 \in [0, 1]$. As we show in this section, this new operator has an interesting property: it is non-degenerative, strengthening the convex property regarding the error. While the convex property states that the error of the function being generated will never be larger than the worst of its parents [Moraglio et al., 2012], we can now state that the error of the function being generated will never be larger than the best of its parents when considering training error, as presented below.

Let $o(\mathbf{x}) = GSXE(p_1(\mathbf{x}), p_2(\mathbf{x}))$ be the offspring resulting from the geometric semantic crossover from Equation 4.4 applied to parents p_1, p_2. Replacing β_2 by $1-\beta_1$, the sum of squared errors (SSE) of o can be expressed in terms of β_1 as[2]

$$SSE(\beta_1) = \sum_{i=1}^{n} [y_i - \beta_1 \cdot p_1(\mathbf{x}_i) - (1 - \beta_1) \cdot p_2(\mathbf{x}_i)]^2 \ , \tag{4.5}$$

where the training data is represented as the set of pairs $T = \{(\mathbf{x}_i, y_i)\}_{i=1}^{n}$.

Since SSE(β_1) is continuous, we can calculate the derivative of Equation 4.5 and equal it to zero, finding

$$\beta_1^* = \frac{\sum_{i=1}^{n} [y_i - p_2(\mathbf{x}_i)][p_1(\mathbf{x}_i) - p_2(\mathbf{x}_i)]}{\sum_{i=1}^{n} [p_1(\mathbf{x}_i) - p_2(\mathbf{x}_2)]^2} \tag{4.6}$$

that minimizes the error of $o(\mathbf{x})$. Note that this optimization step can be done in $O(n)$, the same time required for computing the semantics of the offspring. Therefore, the optimization of coefficients does not change the asymptotic complexity of GSGP.

Given that the value found by Equation 4.6 is not necessarily in $[0, 1]$, we employ Equation 4.7 to force β_1 to be in the interval.

$$\beta_1^{**} = max(min(1, \beta_1^*), 0) \tag{4.7}$$

Proposition 1. *The argument β_1^{**} as expressed in Equation 4.7 minimizes the error function (Equation 4.5) while respecting the interval constraint.*

Proof. Since $\lim_{\beta_1 \to c} SSE(\beta_1) = SSE(c), \forall c \in \mathbb{R}$, $SSE(\beta_1)$ is a continuous function in $\mathbb{R}$, we can compute its derivate with respect to β_1 given by

[2] In order to simplify the proof presented in this section, we consider that $s(p_1) \neq s(p_2)$. The case where $s(p_1) = s(p_2)$ is irrelevant, since $SSE(\beta_1) = \sum_{i=1}^{n} [y_i - p_1(\mathbf{x}_i)], \forall \beta_1 \in \mathbb{R}$.

$$\frac{\delta SSE}{\delta \beta_1} = 2 \cdot \sum_{i=1}^{n} [y_i - p_2(\mathbf{x}_i)][p_2(\mathbf{x}_i) - p_1(\mathbf{x}_i)] + \beta_1 \cdot [p_2(\mathbf{x}_i) - p_1(\mathbf{x}_i)]^2 \ .$$

By making the derivative equal to zero, we find a (local) minimum or maximum point:

$$\beta_1^* = \frac{\sum_{i=1}^{n} [y_i - p_2(\mathbf{x}_i)][p_1(\mathbf{x}_i) - p_2(\mathbf{x}_i)]}{\sum_{i=1}^{n} [p_1(\mathbf{x}_i) - p_2(\mathbf{x}_i)]^2} \ .$$

In order to show that β_1^* is a minimization point we apply the second derivative test [Stewart, 2008]. We compute the second derivative of $SSE(\beta_1)$ with respect to β_1, leading to

$$\begin{aligned} \frac{\delta^2 SSE(\beta_1)}{\delta \beta_1^2} &= \sum_{i=1}^{n} [p_2(\mathbf{x}_i) - p_1(\mathbf{x}_i)]^2 \\ &\geq 0 \ . \end{aligned}$$

Given that $s(p_1) \neq s(p_2)$, as stated, the second derivative obtained is always positive. Thus, β_1^* is a minimization point and $SSE(\beta_1)$ is convex. Suppose now that $\beta_1^* > 1$. Then, $\beta_1^{**} = 1$ and it minimizes $SSE(\beta_1)$ while being in the interval $[0, 1]$, since $SSE(\beta_1)$ is convex and β_1^{**} is the closest point to β_1^* in the interval. An analogue reasoning implies if $\beta_1^* < 0$, $\beta_1^{**} = 0$ minimizes the error function while being in the required interval. Finally, if $0 \leq \beta_1^* \leq 1$, then $\beta_1^{**} = \beta_1^*$ and it is also a minimization point in the required interval. Thus, we prove that β_1^{**} minimizes $SSE(\beta_1)$ while respecting the interval constraint. □

As β_1 is optimized in the closed interval $[0, 1]$, if $0 < \beta_1^{**} < 1$, then $SSE(\beta_1^{**}) \leq SSE(1)$ and $SSE(\beta_1^{**}) \leq SSE(0)$. Otherwise, β_1^{**} would be 1 or 0 and the best of the two functions used in crossover would be simply replicated. This shows that the error of the function being generated will never be larger than the largest error amongst its parents. Notice that the fitness of the resulting individual is still upper bounded by $max(f(p_1), f(p_2))$—both in the training and test sets—since this modified version of the crossover is still a convex combination of functions.

4.2.2 Optimized Non-Convex Euclidean-Based Geometric Semantic Crossover

The convex geometric semantic crossover operator can be very constrained: the fact that it can only generate solutions semantically intermediate to the functions being used for crossover implies that the performance can be strongly determined by the initial population. In order to build a more flexible operator, we propose the following non-convex crossover operator, based on linear combinations. However, this means increasing the risk of overfitting, as we now do not have any guarantee that the test error is upper bounded by the error of the worst of the parents.

First, we consider the case where the vectors defined by $s(p_1)$ and $s(p_2)$ are linearly independent and different from $\mathbf{0}$. Given the geometric semantic crossover from Equation 3.6, without the restrictions regarding β_1 and β_2, i.e., β_1 and β_2 can be outside the interval $[0, 1]$ and β_2 not necessarily equals $1 - \beta_1$, we can express the error of the offspring function $o(\mathbf{x})$ generated through this operator in terms of β_1 and β_2 as

$$SSE(\beta_1, \beta_2) = \sum_{i=1}^{n} [y_i - \beta_1 \cdot p_1(\mathbf{x}_i) - \beta_2 \cdot p_2(\mathbf{x}_i)]^2 \ . \tag{4.8}$$

Since $SSE(\beta_1, \beta_2)$ is continuous in $\mathbb{R}^2$, we can use the same strategy presented in Section 4.2.1 to find β_1^* and β_2^* that minimizes Equation 4.8. Let F be an n-by-2 matrix, where $F_{ij} = p_j(\mathbf{x}_i)$, and let Y be a column vector of length n containing the target values of each training instance. Then

$$\begin{pmatrix} \beta_1^* \\ \beta_2^* \end{pmatrix} = (F^t F)^{-1} F^t Y \ . \tag{4.9}$$

Proposition 2. *The arguments β_1^* and β_2^*, as expressed in Equation 4.9, minimize the error function (Equation 4.8).*

Proof. Since $\lim_{(\beta_1,\beta_2)\to(c_1,c_2)} SSE(\beta_1, \beta_2) = SSE(c_1, c_2)$ for an arbitrary pair $(c_1, c_2) \in \mathbb{R}^2$, $SSE(\beta_1, \beta_2)$ is continuous in $\mathbb{R}^2$ and we can compute its derivative with respect to β_1 and β_2

$$\begin{aligned} \frac{\delta SSE}{\delta \beta_1} &= \sum_{i=1}^{n} -2[y_i \cdot p_1(\mathbf{x}_i) - \beta_1 \cdot p_1(\mathbf{x}_i)^2 - \beta_2 \cdot p_1(\mathbf{x}_i) \cdot p_2(\mathbf{x}_i)] \\ \frac{\delta SSE}{\delta \beta_2} &= \sum_{i=1}^{n} -2[y_i \cdot p_2(\mathbf{x}_i) - \beta_1 \cdot p_1(\mathbf{x}_i) \cdot p_2(\mathbf{x}_i) - \beta_2 \cdot p_2(\mathbf{x}_i)^2] \ . \end{aligned}$$

Letting F be a n-by-2 matrix, where $F_{ij} = p_j(\mathbf{x}_i)$, and letting Y be a column vector of length n containing the target values for each training instance, by making the derivatives above equal to zero, we arrive at the following matrix formulation[3]:

$$(F^tF)\begin{pmatrix}\beta_1^*\\ \beta_2^*\end{pmatrix} = F^tY \Rightarrow \begin{pmatrix}\beta_1^*\\ \beta_2^*\end{pmatrix} = (F^tF)^{-1}F^tY \ .$$

Therefore, we only need to show that (β_1^*, β_2^*) consists of a minimization (and not maximization) point. For that, we will apply the second derivative test [Stewart, 2008] to the error function (Equation 4.8). Replacing the second derivatives

$$\begin{aligned}\frac{\delta^2 SSE}{\delta\beta_1^2} &= 2\sum_{i=1}^{n} p_1(\mathbf{x}_i)^2\\ \frac{\delta^2 SSE}{\delta\beta_2^2} &= 2\sum_{i=1}^{n} p_2(\mathbf{x}_i)^2\\ \frac{\delta^2 SSE}{\delta\beta_1\delta\beta_2} &= 2\sum_{i=1}^{n} p_1(\mathbf{x}_i)\cdot p_2(\mathbf{x}_i)\end{aligned}$$

in the determinant of the Hessian matrix $H(\beta_1, \beta_2)$ of SSE, we have

$$\begin{aligned}D &= \frac{\delta^2 SSE}{\delta\beta_1^2}(\beta_1^*, \beta_2^*)\cdot\frac{\delta^2 SSE}{\delta\beta_2^2}(\beta_1^*, \beta_2^*) - \left[\frac{\delta^2 SSE}{\delta\beta_1\delta\beta_2}(\beta_1^*, \beta_2^*)\right]^2\\ &= 4\sum_{i=1}^{n} p_1(\mathbf{x}_i)^2\sum_{i=1}^{n} p_2(\mathbf{x}_i)^2 - 4\left(\sum_{i=1}^{n} p_1(\mathbf{x}_i)\cdot p_2(\mathbf{x}_i)\right)^2 \ .\end{aligned}$$

Following from the Cauchy-Schwarz inequality [Banerjee and Roy, 2014] we have

$$\sum_{i=1}^{n} p_1(\mathbf{x}_i)^2\sum_{i=1}^{n} p_2(\mathbf{x}_i)^2 \geq \left(\sum_{i=1}^{n} p_1(\mathbf{x}_i)\cdot p_2(\mathbf{x}_i)\right)^2 \ .$$

Thus,

$$D \geq 0 \ .$$

$D = 0$ when the Cauchy-Schwarz is an equality. This occurs when at least one of the parents' semantics ($s(p_1)$ and $s(p_2)$) is equal to $\mathbf{0}$ or when $s(p_1)$ and $s(p_2)$ are

[3]This case consider that $s(p_1)$ and $s(p_2)$ are linear independent. Thus, the Gram matrix F^tF has non-zero determinant and is invertible [Lay, 2012].

linearly dependent[4] (see [Banerjee and Roy, 2014], p. 180 for the proof). Thus, $D > 0$, which leads us to conclude that (β_1^*, β_2^*) is indeed a minimization point.

□

There are three cases not considered in the proof above. First, if $s(p_1) = s(p_2) = \mathbf{0}$, then $GSXE(p_1(\mathbf{x}), p_2(\mathbf{x})) = \mathbf{0}$, for any pair $(\beta_1, \beta_2) \in \mathbb{R}^2$ and no minimization is necessary. Second, if the semantic vectors $s(p_1)$ and $s(p_2)$ are linear dependent, Equation 4.4 can be rewritten as

$$\begin{aligned} GSXE(p_1(\mathbf{x}), p_2(\mathbf{x})) &= \beta_1 \cdot p_1(\mathbf{x}) + \beta_2 \cdot (\lambda \cdot p_1(\mathbf{x})) && (4.10) \\ &= \beta_3 \cdot p_1(\mathbf{x}) \ , && (4.11) \end{aligned}$$

where $\beta_3 = \beta_1 + \lambda \cdot \beta_2$.

The third case occurs when $s(p_1) \neq \mathbf{0}$ and $s(p_2) = \mathbf{0}$ or vice-versa. In this circumstance, Equation 4.4 can also be rewritten as Equation 4.11. Notice that when $s(p_1) = \mathbf{0}$ and $s(p_2) \neq \mathbf{0}$, we just have to change $p_1(\mathbf{x})$ by $p_2(\mathbf{x})$ in the equation without loss of generality.

Both the second and the third cases can be optimized in relation to β_3. The error of the offspring function $o(\mathbf{x})$ generated by Equation 4.11 can be expressed in terms of β_3 as

$$SSE(\beta_3) = \sum_{i=1}^{n} [y_i - \beta_3 \cdot p_1(\mathbf{x}_i)]^2 \ . \tag{4.12}$$

Since $SSE(\beta_3)$ is continuous in $\mathbb{R}^2$, we compute the derivative of Equation 4.12 and equal it to zero, leading to

$$\beta_3^* = \frac{\sum_{i=1}^{n} y_i \cdot p_1(\mathbf{x}_i)}{\sum_{i=1}^{n} p_1(\mathbf{x}_i)^2} \ , \tag{4.13}$$

which minimizes the error of $o(\mathbf{x})$.

Proposition 3. *The argument β_3^*, as expressed in Equation 4.13, minimizes the error function (Equation 4.12).*

Proof. Since $\lim_{(\beta_3)\to(c)} SSE(\beta_3) = SSE(c)\ \forall c \in \mathbb{R}$, $SSE(\beta_3)$ is continuous in $\mathbb{R}$ and we can compute its derivative with relation to β_3 as

[4] As stated at the beginning of this section, these two cases are not considered in this proof.

$$\frac{\delta SSE}{\delta \beta_3} = -2 \cdot \sum_{i=1}^{n} [y_i \cdot p_1(\mathbf{x}_i) - \beta_3 \cdot p_1(\mathbf{x}_i)^2] \ .$$

In order to show that β_3^* is a minimization point we employ the second derivative test. The second derivative of $SSE(\beta_3)$ with relation to β_3 is given by

$$\begin{aligned} \frac{\delta^2 SSE(\beta_3)}{\delta \beta_3^2} &= 2 \cdot \sum_{i=1}^{n} p_1(\mathbf{x}_i)^2 \\ &\geq 0 \ . \end{aligned}$$

Since $p_1(\mathbf{x}) \neq 0$ for at least one $\mathbf{x}_i \in in$, the second derivative is always positive. Thus, β_3^* is a minimization point. □

Notice that, besides the operator proposed in this section does not corresponds to a convex combination, it maintains the non-degenerative property regarding training error, since we are optimizing parameters over a set that includes $(\beta_1 = 1, \beta_2 = 0)$ and $(\beta_1 = 0, \beta_2 = 1)$.

4.2.3 Experimental Results

The experiments reported in this section were performed to evaluate the role of geometric semantic crossover in a diversified collection of real-world and synthetic datasets. The first experiment was designed to show that, different from canonical crossover operators, geometric semantic crossover operators have nothing to do with macro mutation [Angeline, 1997]—as they guarantee their offspring will be semantically intermediate to their parents—and that, when combined with the geometric semantic mutation operator, they outperform strictly mutation-based methods. The second experiment compares variations of convex semantic crossover operators using different distances and optimized coefficients.

In order to analyse and evaluate the impact of the methods and conjectures proposed in this thesis, including the ones presented in this chapter, we adopted a test bed composed of datasets selected from different sources, including the UCI machine learning repository [Bache and Lichman, 2014], GP benchmarks [McDermott et al., 2012; White et al., 2013] and works involving GSGP [Castelli et al., 2013b; Vanneschi et al., 2013], as presented in Table 4.1. The column *No. of instances* indicates the total number of examples in real datasets and the size of the training and test sets

Table 4.1: Main characteristics of the datasets used in the experiments.

Abbr.	Dataset	No. of Input Variables	No. of instances Train.	Test	Nature	Source	Experimental Strategy
air	Airfoil Self-noise	5	1503		Real	1	10 × 5-CV
con	Concrete Comp. Str.	8	1030		Real	2	10 × 5-CV
cpu	Computer Hardware	7	209		Real	1	10 × 5-CV
eneC	Energy Effic. (cooling)	8	768		Real	1	10 × 5-CV
eneH	Energy Effic. (heating)	8	768		Real	1	10 × 5-CV
for	Forest Fires	10	517		Real	1	10 × 5-CV
bio	Human Oral Bioav.	241	359		Real	3	10 × 5-CV
ppb	PPB	626	131		Real	3	10 × 5-CV
tow	Tower	25	4999		Real	5	10 × 5-CV
wineR	Wine Quality (Red)	11	1599		Real	1	10 × 5-CV
wineW	Wine Quality (White)	11	4898		Real	1	10 × 5-CV
yac	Yacht Hydrodynamics	6	768		Real	1	10 × 5-CV
kei5	Keijzer-5	3	1000	10000	Synt.	4	10 × 5-ND
kei6	Keijzer-6	1	50	120	Synt.	4,5	50 × D
kei7	Keijzer-7	1	100	991	Synt.	4	50 × D
kor1	Korns-1	5	10000	10000	Synt.	4	10 × 5-ND
kor2	Korns-2	5	10000	10000	Synt.	4	10 × 5-ND
kor12	Korns-12	5	10000	10000	Synt.	4,5	10 × 5-ND
vla1	Vladislavleva-1	2	100	2025	Synt.	4	10 × 5-ND
vla4	Vladislavleva-4	5	1024	5000	Synt.	4,5	10 × 5-ND

Sources: (1) Bache and Lichman [2014], (2) Castelli et al. [2013b],
(3) Vanneschi et al. [2013], (4) McDermott et al. [2012], (5) White et al. [2013]

in synthetic ones. All the categorical attributes were removed to maintain all the attributes in the real domain.

We defined different strategies for the experiments according to the nature and source of the datasets, as detailed in the last column of Table 4.1. For real datasets, we randomly partitioned the data into 5 disjoint sets of the same size and executed the methods 10 times with a 5-fold cross-validations (10 × 5-CV). For the synthetic ones, we used two different strategies according to the way the dataset was defined in its original work: non-deterministic datasets were resampled 5 times and the experiments were repeated ten times for each sampling (10 × 5-ND); experiments with datasets deterministically sampled were repeated fifty times with the same data folds (50 × D). Training and test sets were sampled with the same strategy. The only exception was the *Vladislavleva-1* dataset, where the training set was sampled following the ten 5-ND strategy (10 × 5-ND) and the test set was deterministically sampled one time only, following the original experiment [McDermott et al., 2012]. At the end, all methods were executed 50 times.

We adopted the paired Wilcoxon test to analyse the statistical differences regarding the results of the experiments performed with this test bed [Demšar, 2006]. We employed the Wilcoxon test provided by the *stats* package, from the R language, with

continuity correction and exact p-value computation [R Core Team, 2015]. For all these tests we considered a confidence level of 95%.

In order to reduce the importance of the outliers in the experimental analysis, we adopted the median and the Interquartile Range (IQR) to summarize the results, instead of the mean and standard deviation, respectively. The IQR is the difference between the upper and lower quartiles, i.e., $IQR = Q_3 - Q_1$, where Q_1 and Q_3 are the first and third quartiles, respectively.

For all methods, a preliminary parameter study was performed, and we defined the population size equal to 1,000 individuals, evolved for 2,000 generations to ensure convergence. The operator set included the arithmetic operators $\{+, -, \times\}$ and the AQ function as an alternative to the division, and the terminal set included ephemeral random constants [Koza, 1992a] defined in $[-1, 1]$ and all the input variables from the respective datasets. The tournament size was defined as 10 and the probabilities of applying the crossover and mutation operators were both defined as 0.5[5].

We employed the logistic wrapper [Vanneschi et al., 2013, 2014b; Castelli et al., 2015a] to restrain the output of the random functions used by geometric semantic operators—namely mutation (GSM) and crossover for fitness functions based on Manhattan distance (GSXM). All methods use GSM as the mutation operator, with a mutation step equal to 10% of the standard deviation of the training data output.

For each algorithm, the following variations of semantic crossover were tested:

- **GSXE:** Euclidean-based geometric semantic crossover with random crossover coefficients;
- **GSXM:** Manhattan-based geometric semantic crossover with random crossover (function) coefficients;
- **GSXE-C:** Optimized convex Euclidean-based geometric semantic crossover operator;
- **GSXE-L:** Optimized non-convex Euclidean-based geometric semantic crossover operator;
- **GSGP-Mut:** GSGP with crossover rate equal to 0—i.e., GSM is the only search operator and it is always applied.

Table 4.2 shows the median results of Root Mean Squared Error (RMSE) followed by the Interquartile Range (IQR) obtained by both configurations on the 20

[5]The relatively high mutation rate is commonly adopted in GSGP in order to be able to effectively explore the search space [Vanneschi et al., 2013, 2014b].

Table 4.2: Median test RMSE (and respective IQR) obtained after 2,000 generations for each dataset, considering 50 replications. Results statically better are marked in bold.

Dataset	GSGP-Mut		GSXM	
	median	IQR	median	IQR
air	**6.54**	0.83	8.42	0.76
bio	32.70	1.29	**30.74**	2.33
con	5.45	0.60	**5.39**	0.64
cpu	34.51	9.81	**30.92**	15.19
eneC	**1.46**	0.19	1.51	0.15
eneH	0.97	0.17	0.96	0.18
for	55.12	42.12	**51.63**	48.17
kei5	**0.04**	0.01	0.05	0.01
kei6	0.30	0.17	0.40	0.34
kei7	0.03	0.01	**0.02**	0.01
kor1	112.21	11.03	**86.53**	7.29
kor2	638.51	3028.40	**632.48**	3064.42
kor12	1.02	0.01	**1.02**	0.00
ppb	29.33	7.29	**28.74**	5.29
tow	**21.83**	1.17	21.92	1.27
vla1	0.06	0.04	**0.04**	0.03
vla4	0.06	0.01	**0.05**	0.00
wineR	0.63	0.05	**0.62**	0.04
wineW	0.70	0.01	**0.70**	0.01
yac	**2.18**	0.62	2.52	0.85

datasets used as benchmarks. The results indicate that GSXM performs better than GSGP-Mut—GSXM was statistically better than GSGP-Mut in 12 datasets, while being statistically worse in five.

These results indicate that the geometric semantic crossover operator in GSGP is indeed beneficial and has a different effect from using mutation-only based methods, despite the simplicity of the fitness landscape. A possible explanation for this fact is the difference in the way the geometric semantic crossover and mutation operators search the space. The former limits the range of possible offspring to the convex hull described by the population's semantics, ensuring an error bounded by this convex hull. The latter, on the other hand, generates individuals that do not hold this property, which can degrade the population's mean error.

Notice that our conclusion about the geometric semantic crossover operator being beneficial to GSGP is contrary to the one presented by Gonçalves et al. [2015]. A possible explanation for this fact is the reduced number of datasets adopted in their experimental analysis. Table 4.2 presents three datasets where the results obtained by the GSGP-Mut and GSXM have no statistical difference. Analysing only these datasets, leads us to the same conclusion presented by Gonçalves et al. [2015]. However, when all the 20 datasets are analysed, we draw a different conclusion, i.e., the geometric semantic crossover is beneficial to the search process.

Table 4.3: Median training RMSEs (and IQR) obtained after 2,000 generations for each dataset, considering 50 replications.

Dataset	GSXE		GSXM		GSXE-C		GSXE-L	
	median	IQR	median	IQR	median	IQR	median	IQR
air	8.76	0.58	7.89	0.53	7.81	0.73	1.73	0.06
bio	10.59	0.57	9.89	0.65	10.05	0.69	7.47	0.99
con	4.02	0.13	3.65	0.14	3.94	0.14	3.61	0.16
cpu	7.38	1.10	6.13	0.66	7.11	0.88	5.43	0.98
eneC	1.38	0.09	1.26	0.07	1.38	0.06	1.17	0.09
eneH	0.86	0.10	0.80	0.11	0.86	0.11	0.60	0.14
for	35.70	4.83	30.74	4.63	34.99	4.25	26.24	3.46
kei5	0.05	0.00	0.04	0.00	0.05	0.00	0.01	0.01
kei6	0.01	0.01	0.01	0.00	0.01	0.01	0.00	0.00
kei7	0.02	0.01	0.02	0.01	0.02	0.01	0.00	0.00
kor1	83.49	5.95	74.80	6.10	80.22	7.08	0.00	0.00
kor2	786.93	858.27	737.88	830.60	778.46	854.49	661.29	829.07
kor12	0.97	0.01	0.92	0.01	0.97	0.01	0.95	0.01
ppb	1.39	0.28	0.92	0.27	1.24	0.25	0.63	0.23
tow	21.19	0.54	20.44	0.61	21.03	0.54	18.86	0.66
vla1	0.01	0.00	0.01	0.00	0.01	0.00	0.01	0.00
vla4	0.04	0.00	0.04	0.00	0.04	0.00	0.04	0.00
wineR	0.51	0.01	0.49	0.01	0.51	0.01	0.45	0.01
wineW	0.65	0.00	0.64	0.00	0.65	0.00	0.63	0.00
yac	2.22	0.25	2.12	0.26	2.14	0.16	0.87	0.15

Given that geometric semantic crossover is indeed necessary, we now turn our attention to the impact of the crossover operator distance function, as well as the performance of the two optimized crossover operators proposed, i.e., GSXE-C and GSXE-L. Therefore, we compare GSXE, GSXM, GSXE-C and GSXE-L using the datasets listed in Table 4.1.

Tables 4.3 and 4.4 show the final training and test error, respectively, obtained by each operator on each dataset, while Table 4.5 shows the number of datasets where the operator positioned in the line is statistically better than the operator positioned in the column in both training and test sets, according to the Wilcoxon test.

Regarding the training data, GSXE-L is statistically better than all the other operators, including GSXE-C. This behaviour can be explained by the fact that the optimization performed by the GSXE-L ensures the coefficients computed minimize the sum of squared error (SSE), which is proportional to the RMSE[6]. This optimization can achieve smaller errors than the one performed by GSXE-C since the latter is more restrictive than the former—while GSXE-C guarantees the resulting offspring are a convex combination of their parents, GSXE-L does not have this restriction. GSXE performs poorly compared to other operators regarding the training data and GSXM is consistently better than both GSXE and GSXE-C.

[6] $RMSE = \sqrt{SSE/n}$, where n is the size of the training set.

Table 4.4: Median test RMSEs (and IQR) obtained after 2000 generations for each datasets, considering 50 replications.

Dataset	GSXE		GSXM		GSXE-C		GSXE-L	
	median	IQR	median	IQR	median	IQR	median	IQR
air	9.232	1.401	8.417	0.757	8.401	1.170	2.144	0.185
bio	31.833	2.336	30.736	2.326	31.530	1.814	42.431	18.566
con	5.269	0.762	5.394	0.642	5.261	0.630	5.518	0.592
cpu	30.446	13.094	30.917	15.185	32.137	13.040	59.652	239.326
eneC	1.565	0.172	1.515	0.147	1.571	0.190	1.416	0.201
eneH	1.009	0.161	0.956	0.185	1.017	0.179	0.771	0.203
for	51.792	47.307	51.632	48.166	52.835	47.315	99.678	46.663
kei5	0.051	0.005	0.049	0.005	0.049	0.006	0.017	0.006
kei6	0.448	0.373	0.398	0.339	0.414	0.400	0.057	0.173
kei7	0.021	0.008	0.018	0.010	0.018	0.008	0.054	0.099
kor1	86.670	6.586	86.526	7.287	83.878	9.535	0.000	0.000
kor2	638.019	3055.120	632.479	3064.421	638.397	3056.030	907.735	3427.487
kor12	1.015	0.007	1.017	0.004	1.017	0.011	1.053	0.007
ppb	28.975	6.093	28.740	5.290	27.761	6.591	35.504	15.640
tow	22.404	0.937	21.920	1.272	22.155	1.152	21.574	2.860
vla1	0.055	0.027	0.044	0.030	0.053	0.035	0.128	0.092
vla4	0.052	0.002	0.052	0.003	0.052	0.003	0.081	0.012
wineR	0.622	0.051	0.620	0.040	0.622	0.046	0.663	0.073
wineW	0.697	0.016	0.696	0.014	0.699	0.015	0.720	0.023
yac	2.533	0.865	2.522	0.847	2.439	0.673	1.203	0.315

Table 4.5: Number of datasets where the operator presented in the row was statistically better than the operator presented in the column according to the Wilcoxon test considering training and test error.

	Training Error				Test Error			
	GSXE	GSXM	GSXE-C	GSXE-L	GSXE	GSXM	GSXE-C	GSXE-L
GSXE	—	0	1	1	—	2	4	12
GSXM	20	—	17	2	13	—	11	12
GSXE-C	16	0	—	1	6	2	—	12
GSXE-L	19	17	19	—	7	7	7	—

Regarding the test data, we observe that the performance of GSXE-L deteriorates in relation to its performance in the training data, with results statistically worse than those obtained by the other operators in more than half of the datasets of the test bed. We attribute these results to overfitting as a side effect of not ensuring the convex property, since we eliminate any bounds on the test error. GSXE and GSXE-C present similar performance in the test data, indicating that the convex optimization has no statistically significant effect on the error. GSXM, on the other hand, presents the best performance among the operators, indicating that GSGP performs better in the Manhattan metric semantic space than in the Euclidean. Overall, on the test set the optimization of the coefficients has no significant impact on the crossover behaviour when the convexity is maintained, and has negative impact when the optimization is not convex.

4.2.4 Conclusions

In this section we investigated the importance of the crossover operator in GSGP and the effects of the use of different distance functions when defining the crossover operator along with the performance of two new optimized crossover operators, GSXE-C and GSXE-L.

A first experimental analysis indicated the crossover operator is important to GSGP search process, contradicting the results from Gonçalves et al. [2015]. A second empirical analysis indicated that the use of the Manhattan distance function instead of Euclidean for geometric semantic crossover has positive impact on the training and test errors, even when using our proposed versions of the Euclidean based crossover with optimized coefficients. The use of linear combination instead of convex combination led to the best results in the training and to the worst results in the test data. A possible explanation to this behaviour is the the lack of any property restricting the error in the test—in comparison to the convex combination.

Chapter 5

Sequential Symbolic Regression

As introduced by Issue 3 in Section 1.1 and presented in Section 3.3.3.1, one of the main drawbacks of the GSGP method is the exponential growth of the individuals in the population along the generations, which limits its application in real life problems.

EGSGP (efficient GSGP) [Vanneschi et al., 2013, 2014b; Castelli et al., 2015a] ensures a linear time and space complexity with relation to the population size $|P|$ and number of generations g_{max}, given by $O(|P|g_{max})$. However individuals are not explicitly built during the search process and an "unwind" operation is needed to build the syntax of any individual, since the use of pointers mitigates the problem of representing large individuals in memory but does not prevent the exponential growth of their representation. Therefore, when an individual is reconstructed it still presents an exponential size complexity. The excessive size increases the computational time to evaluate new input data and makes it difficult to interpret the solutions.

In this chapter we present a first attempt to maintain the size of the solutions in a manageable level while applying the geometric semantic crossover operator, with the Sequential Symbolic Regression (SSR). The method is inspired by the divide and conquer strategy, where the original problem is decomposed in (potentially) simpler subproblems that are tackled independently and their solutions are combined to form the solution to the original problem. In the next section we present the inspirational background for SSR.

5.1 Background

Since the introduction of genetic programming [Koza, 1992a], researchers have been interested in exploring the regularities and modularity of the problem space. One of the main motivations is to identify these regularities to decompose the problem at

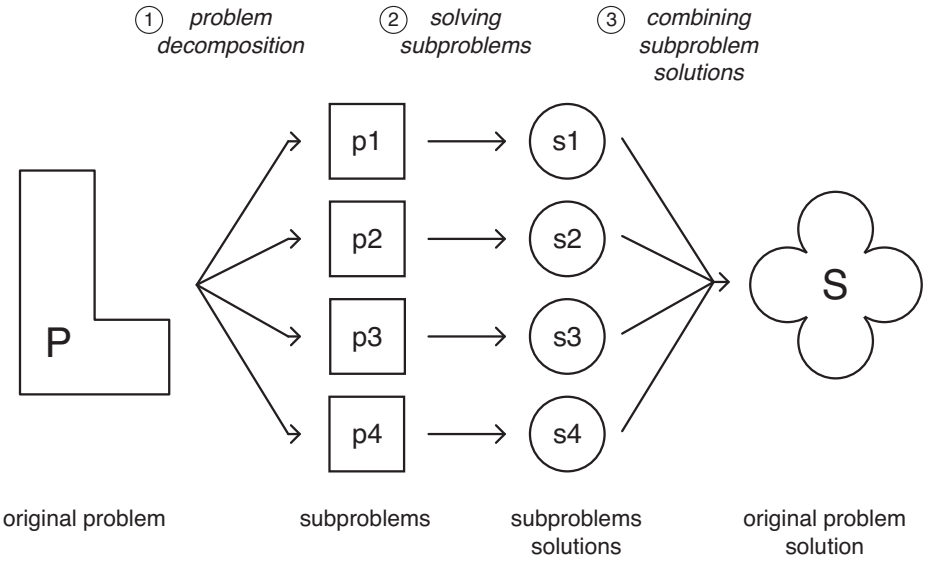


Figure 5.1: The hierarchical problem-solving process: the original problem P is decomposed in a set of subproblems (step 1); the goal is then to solve each of the subproblems (step 2); finally, the solution S to the original problem P is created by using the solutions to the subproblems (step 3). Figure adapted from [Koza, 1992a].

hand into more tractable subproblems—finding solutions to subproblems should be easier than finding a solution to the original problem, and these subsolutions can be used to create the solution to the whole problem. This process is illustrated in Figure 5.1. In the next sections we present two approaches adopted in the GP literature to decompose the problem into (possibly) simpler sub-problems: modularization and iterative adjustment of the error.

5.1.1 Modularization

The Automatic Defined Functions (ADFs) proposed by Koza [1992a,b, 1994] was one of the first ideas to address the automated problem decomposition. ADFs impose a syntactical structure to individuals—an individual genotype is divided into a *result-producing branch* and several *function-defining branches*. The motivation is that function definitions potentially exploit the regularities of the problem space and these definitions can be used from the result-producing branch. On the one hand, Koza argues that by allowing the definition and use of functions, the problem is decomposed into sub-problems. On the other hand, the modular structure (syntax) of individuals is manually defined and, therefore, the decomposition process is not autonomous—the number of ADFs and their parameters are controlled by user-defined values. Additionally, even if func-

tions actually represent solutions to sub-problems, they are being evolved at the same time as the complete solution. There is a pressure to solve all parts of the problem at once—the definition of the functions and the correct use of those functions.

A popular idea to explore problem space regularities focuses on defining modules based on the genetic material of individuals. Several works involve the random selection of subtrees to create modules: Koza [1992a] proposed the use of a subtree *encapsulation* operator, which consists of randomly selecting a subtree from an individual to create a terminal primitive that encapsulated the subtree; Angeline and Pollack [1992, 1994] proposed the Genetic Library Builder (GLiB) system, which employs mutation operators that randomly select subtrees to create modules (*compress* operator) that can be later expanded (*expand* operator). Similar *compress* and *expand* operators to create and expand modules were more recently proposed by Walker and Miller [2008] in the context of Embedded Cartesian Genetic Programming (ECGP), with the extension of the use of module-altering operators (*module point* mutation, *add-input*, *add-output*, *remove-input* and *remove-output* operators); Spector et al. [2011a,b, 2012] proposed the use of the concept of 'tags' to label fragments of code that can be later reused by referencing the same label—while this is similar to the use of a *compress* operator, it provides the flexibility of partial name matches (a label will match the closest matching tag).

5.1.2 Iterative Adjustment of the Error

The second divide and conquer strategy applied in GP is related to ensemble learning, in particular to *gradient boosting* [Breiman, 2001; Friedman, 2001; Friedman and Friedman, 2002]. Ensembles are methods that generate several weak models that are combined to generate a stronger classification or regression model [Mendes-Moreira et al., 2012]. The main principle behind gradient boosting is to learn new models with highest correlation to the negative gradient of the loss (error) function associated to the final ensemble output. Given a training set $T = \{(\mathbf{x}_i, y_i)\}_{i=1}^{n}$ defined by an unknown function $f^*(\mathbf{x})$, the method approximates $f^*(\mathbf{x})$ by the incremental model

$$f(\mathbf{x}) = \sum_{m=0}^{M} \beta_m h(\mathbf{x}; \mathbf{a}_m) \ , \tag{5.1}$$

where $h(\mathbf{x}; \mathbf{a}_m)$ is the base (weak) learner—usually a simple parameterized function characterized by a parameter vector $\mathbf{a}_m$, such as a decision tree—and β_m is the corresponding coefficient. The method starts with an initial approximation, induced directly from T, and incrementally expands the model. The function $h(\mathbf{x}; \mathbf{a}_m)$ is fitted by ad-

justing the parameter $\mathbf{a}_m$ using least-squares and the optimal value of β_m is determined by a line search[1].

When the loss function adopted is the quadratic error, the optimization process is similar to searching by a model that approximates the residuals of the previous model found, similar to the idea presented in SSR.

However, even before the theoretical foundation given by gradient boosting, Lee [1999] proposed a form of recursively adjusting a symbolic regression function within the context of time series forecasting by Genetic Recursive Regression (GRR). GRR assumes real world time series are composed of a deterministic (f_D) and a stochastic component (f_S), such as

$$f = f_D + f_S \ . \tag{5.2}$$

A traditional GP expresses the deterministic part while the time series' residual, $f_S = f - f_D$, is again expanded into two parts.

GRR assigns numerical coefficients to each function induced during the process and calculates them by the least squares method with respect to the training dataset. Panyaworayan and Wuetschner [2002] presented a similar approach to GRR, also applied to time series forecasting. However, a genetic algorithm is used to adjust the values of the numerical coefficients instead of the least squares method.

5.2 The Proposed Method

In order to explore the geometric properties of the geometric semantic crossover operator while maintaining the size of the individuals in a manageable level, we proposed the Sequential Symbolic Regression (SSR) method [Oliveira et al., 2015, 2016b].

SSR sequentially transforms the original problem, according to the partial solutions generated, into potentially simpler ones. When the symbolic regression finds a suboptimal function $f(\mathbf{x})$, the expected behaviour of a complementary function $f'(\mathbf{x})$ can be determined so that, when combined with $f(\mathbf{x})$, the error is minimized. Hence, if we can find a good approximation for $f'(\mathbf{x})$, the resulting function can still be improved. The method employs a GP to search for function approximations and the geometric semantic crossover operator to find the expected semantics of the complementary functions and combine them.

[1]Line search is an optimization approach to find a local minimum of an univariable objective function [Sun and Yuan, 2006].

Unlike GSGP, SSR applies the geometric semantic crossover as a concatenation operator in such a way that the size of the resulting solution is proportional to the average size of the functions induced by the GP. Let the expected size of GP induced individuals be $E[f]$, the resulting solution size is given by $it \cdot E[f]$, where it is the number of SSR iterations, i.e., the number of independent GP executions. However, despite using the geometric semantic crossover operator, SSR is not a geometric semantic method.

The majority of the error metrics adopted for symbolic regression, such as the Mean Squared Error (MSE), the Mean Absolute Error (MAE), the Root Mean Squared Error (RMSE) or the Coefficient of Determination (R^2) use the sum of the absolute or squared residuals—the difference between the current output and the function output—to compute their respective values. These metrics share a common property: the error quantified by the metric decreases when the absolute residuals are minimized. A residual $e(f, T)$ corresponds to the deviation of the fitted function f from the observed outputs in T. For each pair $(\mathbf{x}_i, y_i)$ in T, the residual is defined as

$$e_i(f, T) = y_i - \hat{y}_i = y_i - f(\mathbf{x}_i) \ . \tag{5.3}$$

The optimal solution to a regression problem is a function f^* that perfectly fits the input data, i.e., where $e_i(f^*, T) = y_i - f^*(\mathbf{x}_i) = 0$, for $i = 1, 2, ..., n$. However, often the regression model is just an approximation of f^*, not reaching a zero error or a predefined minimum error. Whenever the symbolic regression finds a suboptimal function f—i.e., $e_i(f, T) \neq 0$ for at least one observation i—the model can still be improved by adding a term that approximates the residuals.

SSR takes advantage of this property and employs the geometric semantic crossover operator for fitness function based on Euclidean distance (GSXE) [Moraglio et al., 2012] to subsequently combine suboptimal functions and approximations to their residuals. The main idea behind GSXE is to create a convex combination of two solutions to the problem being tackled. Considering the semantic space where these solutions lie, the geometric semantic crossover guarantees that the quality of the new solution generated through this combination will never be worse than the worst of its parents. We use this operator to combine an initial approximation function f evolved over T with a new function f', generated over the residuals of f in T.

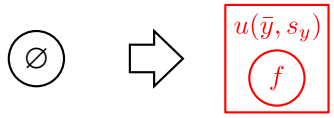
Algorithm 4 presents the high-level pseudocode of the SSR procedure. The method starts with an empty solution tree S, which is incrementally constructed over the iterations and returned as the final regression model. At the beginning of the k-th iteration, the desired outputs are normalized (line 8), as detailed in Section 5.2.2. The

Algorithm 4: Sequential Symbolic Regression procedure

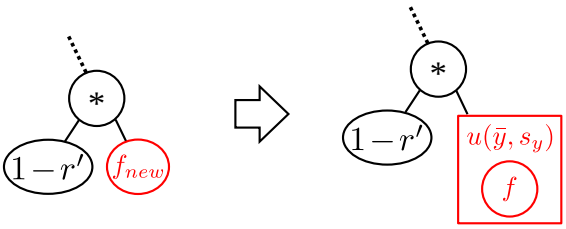
```
   Input: training points (T), number of iterations (it_max),
1          GP parameters (param)
2  Procedure SSR(T, it_max, param)
3      T_x ← (x_1, x_2, ..., x_n), for x_i ∈ ℝ^d ;                    // Inputs
4      T_y ← (y_1, y_2, ..., y_n), for y_i ∈ ℝ ;                      // Outputs
5      S ← ∅ ;                          // Solution iteratively constructed
6      k ← 1 ;
7      while TRUE do
8          (T_y', ȳ, s_y) ← normalize(T_y) ;                  // Normalization
9          f ← runGP(T_x, T_y') ;          // Run GP with normalized data
10         if isLastIteration(f, k) = TRUE then
11             if k = 1 then                                  // First scenario
12                 S ← ȳ + s_y · f;
13             else                                           // Second scenario
14                 f_new ← ȳ + s_y · f;
15             return S
16         r ← random();
17         if k = 1 then                                      // Third scenario
18             S ← ȳ + s_y · [r · f + (1 − r) · f_new];
19         else                                               // Fourth scenario
20             f_new ← ȳ + s_y · [r · f + (1 − r) · f_new];
21         T_y ← adjustOutputs(f, r, T_y');
22         k ← k + 1;
```

procedure returns the normalized output set $T_{y'} = \{y'_i\}_{i=1}^{n}$ and the mean and standard deviation calculated during the process.

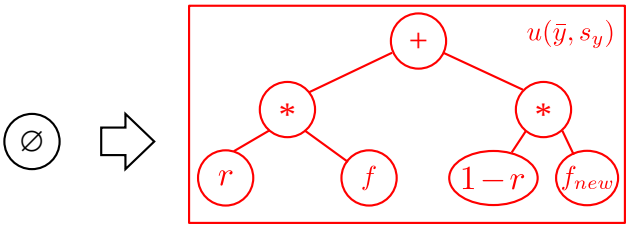
The original inputs and the normalized outputs define the training set $T' = \{(\mathbf{x}_i, y'_i)\}_{i=1}^{n}$, used by a canonical GP to induce a function f (line 9). Following the GP execution, function *isLastIteration* checks whether k is equal to it_{max}—i.e., whether the maximum number of iterations is reached—or if the function f corresponds to the optimal solution—i.e., has RMSE equal to zero. When any of the previous mentioned situations occur, f is added to the solution tree S and the sequential procedure stops, returning S. Otherwise, f is added to the solution tree S using the geometric semantic crossover, and the outputs are updated to consider the residuals of the current solution—procedure *adjustOutputs*—which will be considered as the desired outputs in the next SSR—and consequently, next GP—iteration. The next sections detail all components of SSR.



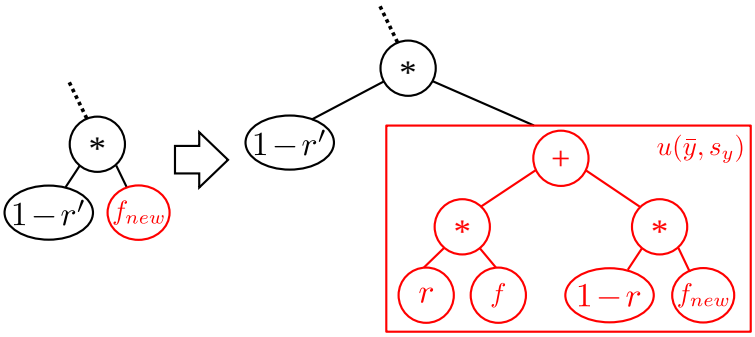
(a) SSR is in the last iteration and $k = 1$.



(b) SSR is in the last iteration and $k \neq 1$.



(c) SSR is not in the last iteration and $k = 1$.



(d) SSR is not in the last iteration and $k \neq 1$.

Figure 5.2: *AddFunction* scenarios. The box $u(\bar{y}, s_y)$ indicates the denormalization process regarding f, r' represents the value of r defined in the previous iteration and nodes in red denote what is modified in the iteration. Scenarios (b) and (d) present only the rightmost subtree of S (symbolized by a dashed line).

5.2.1 Adding functions to the current solution

At each iteration k of SSR, the method adds the (denormalized) function found by GP—f—to the overall solution S. The way f is added to S depends on whether the current iteration is or is not the first or the last iteration, generating four scenarios: SSR is in the last iteration and (1) $k = 1$ or (2) $k \neq 1$, SSR is not in the last iteration and (3) $k = 1$ or (4) $k \neq 1$. These scenarios are depicted in Figure 5.2.

Algorithm 4 presents these four scenarios. The first scenario occurs when SSR finds the optimal solution in the first iteration or when $it_{max} = 1$. In this case (line 12), the method returns the equivalent to the output of a canonical GP with the normalization/denormalization stage presented in Section 5.2.2. The second scenario takes place in the last iteration of the method—i.e., when the maximum number of iterations is reached or the optimal solution is found—when there was at least one other iteration (line 14). In this circumstance, the incomplete function generated in the previous iteration has its pointer f_{new} pointing to the generated function f, resulting in a fully functional solution S, returned to the user.

The other two scenarios employ the geometric semantic crossover operator to append function f to S. The main idea behind this operator is to create a convex combination of two solutions, guaranteeing that the fitness of the new solution generated (in our case, RMSE) will never be worse than the worst of its parents. However, instead of applying the geometric semantic crossover to two parents, SSR applies it to one known parent—f—and to a pointer to be fulfilled with the second parent—f_{new}—in the next iteration, as presented by:

$$r \cdot f + (1 - r) \cdot f_{new} \ , \tag{5.4}$$

where r is random constant in $[0, 1)$.

Notice that the third and fourth scenarios occur only when the optimal solution is not found, i.e., when the procedure *isLastIteration* returns `FALSE`. Thus, the method can search in the next iteration by a function to be pointed by f_{new}, such that zero error is achieved. The difference between the third scenario (line 18) and the fourth (line 20) is that the former takes place in the first iteration, when S is empty—S is replaced by the output of the geometric semantic crossover—and the latter occurs in the subsequent iterations—S is already defined and the current pointer f_{new} starts to point to the output of the geometric semantic crossover operator.

When S is modified by the third or fourth scenarios, SSR has to compute a new output set used to search for the function placed in f_{new}. The method *adjustOutputs* replaces $f(\mathbf{x}_i)$ in Eq. 5.3 by Eq. 5.4, equals the error $e_i(f, T)$ to zero and isolates the pointer f_{new} in the left-hand side, leading to:

$$y_i^{(new)} = \frac{y_i - r \cdot f(\mathbf{x}_i)}{1 - r} \ , \tag{5.5}$$

where $y_i^{(new)}$ is the output expected from the function placed in f_{new} and y_i is in the output set used to find the function f by the *runGP* procedure.

5.2.2 Normalization and Denormalization Procedures

The normalization and denormalization processes are not present in the early SSR version [Oliveira et al., 2015], appearing only in our second work [Oliveira et al., 2016b]. They were introduced to avoid computations leading to anomalous behaviour, caused by approximations and *double* overflow. Let $y_i^{(0)}$ be the i-th original desired output defined by the training set. We can generate any desired output for the k-th iteration of SSR as:

$$y_i^{(k)} = \frac{y_i^{(0)} - r_1 f_1(\mathbf{x}_i) - \sum_{j=2}^{k} \left[r_j f_j(\mathbf{x}_i) \prod_{m=1}^{j-1} (1 - r_m) \right]}{\prod_{j=1}^{k} (1 - r_j)} \ , \tag{5.6}$$

for $k \geq 2$. Let $R = \{r_1, r_2, \ldots, r_k\}$ be equally distributed in intervals defined in $\left[\frac{m}{10}, \frac{m+1}{10}\right)$ for $m = 0, 1, \ldots, 9$ (given that the random values of r_m are uniformly distributed, this is a viable proposition). We can calculate an upper bound for the denominator of (5.6) as

$$\prod_{j=1}^{k} (1 - r_j) \leq \prod_{m=1}^{10} \left(\frac{m}{10}\right)^{\frac{k}{10}} \ . \tag{5.7}$$

When k is sufficiently big this limit potentially leads to double overflow, e.g., for $k = 1000$, this upper bound is approximately $9.47 \cdot 10^{-345}$. This is equivalent to multiplying the numerator by a number as big as $1.06 \cdot 10^{344}$. Even the single-precision 64-bit IEEE 754 floating point standard (used in Java and C++ *double* primitives) cannot handle such a big number, causing an overflow error. The normalization prevents this kind of problem by changing the function output range.

The new values of outputs are normalized as

$$y'_i = \frac{y_i - \bar{y}}{s_y} \ , \tag{5.8}$$

where y'_i is the normalized value, and $\bar{y}$ and s_y are the mean and standard deviation of T_y, respectively.

In order to obtain values in the original range, the function f learned from the normalized data needs to be denormalized through $\bar{y} + s_y \cdot f(\mathbf{x}_i)$—in the last iteration of the algorithm (lines 12 and 14 in Alg. 4)—or through $\bar{y} + s_y \cdot [r \cdot f + (1 - r) \cdot f_{new}]$—in the other iterations (lines 18 and 20 in Alg. 4).

5.2.3 Mitigating Overfitting

Genetic Programming and other machine learning techniques are usually trained by maximizing their accuracy (fitness in the evolutionary algorithm context) in some training data. However, the accuracy of a learned model is measured in relation to unseen data and not to the training set. Overfitting occurs when the model performs well on the training data but very poorly on unseen instances.

In order to reduce overfitting in SSR we adopted a different training sample strategy within GP. Instead of using the entire training set T at each GP generation, we adopt our own variation of the Random Sample Technique (RST) [Liu and Khoshgoftaar, 2004] that randomly partitions the training set into k equal size disjoint subsets, $T_1, T_2, \ldots, T_k$, alternating among them at each s generations. Only the last generation of the GP uses the whole training set T to select the best individual, which is the individual that will be added to the solution tree S.

5.3 Experimental Results

In this section we present an experimental analysis of SSR in the same test bed adopted in Section 4.2.3. The results are compared with a canonical GP [Banzhaf et al., 1998; Koza, 1992a], the GSGP with the GSXM operator from Section 4.2.3 and a modified version of the Genetic Recursive Regression (GRR) [Lee, 1999] for symbolic regression.

In contrast with the original GRR, developed for forecasting real world chaotic time series [Lee, 1999], the version adopted in this section was adapted for symbolic regression, and renamed Genetic Recursive Symbolic Regression (GRSR). GRSR does not use the parallel architecture with multiple populations or the derived terminal set adopted by Lee [1999], and adopts the RMSE as the fitness measure.

The main differences of the GRSR regarding SSR are: (i) the resulting solution S' combines the functions $f_1, f_2, \ldots, f_k$ found by GP through the Equation 5.9, where the coefficients $\alpha_0, \alpha_1, \ldots, \alpha_k$ are obtained by the ordinary least squares method with respect to the training data set; (ii) there is no need for normalization, once the function coefficients are adjusted by ordinary least squares; and (iii) its stopping criteria is based on the RMSE of S in relation to the original training set, while SSR uses the RMSE of the last generated function in relation to the normalized data.

$$S' = \alpha_0 + \alpha_1 \cdot f_1 + \alpha_2 \cdot f_2 + \cdots + \alpha_k \cdot f_k \tag{5.9}$$

In order to make a fair comparison, we included our version of RST presented in Section 5.2.3 within the RSS and GRSR. Preliminary experiments showed that this

Table 5.1: Parameters used by each algorithm during the experiments.

Parameter	**SSR / GRSR**	**GSGP**	**GP**
Crossover probability	0.9	0.5	0.9
Mutation Probability	0.1	0.5	0.1
Tournament Size	7	10	7
Population Size	1000	1000	1000
Number of generations	200	2000	2000
Number of iterations	10	-	-
k (RST)	5	-	-
s (RST)	10	-	-

strategy is prejudicial to GP and GSGP, and hence they do not use RST. For all the methods, we adopted a function set containing the operators $\{+, -, \times, AQ\}$ and a terminal set containing ephemeral random constants [Koza, 1992a] defined in $[-1, 1]$ and all the input variables from the respective datasets. Table 5.1 presents the other parameters adopted in each algorithm, respecting a budget of 2 million evaluations, selected from a preliminary parameter study. Note that the number of generations of SSR is smaller because it runs for 10 iterations, making the number of evaluations the same as for the other methods.

Tables 5.2 and 5.3 present the median and IQR of the test RMSE and resulting function size, respectively, according to the 50 executions of each method. In order to analyse the statistical difference of the results, we adopted the less conservative variant of the Friedman test proposed by Iman and Davenport [1980], here called adjusted Friedman test. We performed one adjusted Friedman test under the null hypothesis that the performances of the methods—measured by their median RMSE—are equal; and one under the null hypothesis that the median sizes of the resulting expressions (functions) generated by the methods are equal. The *p-values* reported by both tests are presented in Table 5.4, and implicate in discarding both null hypothesis with a confidence level of 95%.

As the null hypotheses were discarded, two Finner post-hoc tests [García et al., 2010] were carried out to verify statistical differences among SSR and the other methods in relation to the RMSE and function size. The Adjusted *P-Values* (APV) resulting from these tests are presented in the last three columns of Table 5.4.

Overall, the results show that SSR performs better than the GP in terms of RMSE but generates bigger solutions; and there is no statistical evidence that the SSR and GSGP performance are different but there is strong evidence that SSR induced functions are smaller than those generated by GSGP. Regarding the GRSR, there is no

Table 5.2: Median test RMSE (and respective IQR) for each algorithm according to the experimental strategy presented in Table 4.1.

Dataset	SSR		GRSR		GSGP		GP	
	Med.	IQR	Med.	IQR	Med.	IQR	Med.	IQR
air	3.06	0.39	4.15	0.43	15.12	0.95	9.60	10.43
bio	31.21	3.38	30.99	3.98	31.77	0.89	38.23	8.81
con	7.02	0.62	7.35	0.54	4.89	0.26	9.59	1.60
cpu	55.26	30.27	37.84	24.01	42.74	4.84	45.89	37.99
eneC	2.38	0.45	2.95	0.63	1.68	0.11	3.43	0.26
eneH	1.83	0.66	2.43	0.53	1.09	0.14	3.05	0.56
for	71.23	66.86	68.89	57.64	84.59	2.60	37.64	63.87
kei5	0.01	0.01	0.01	0.01	0.06	0.00	0.03	0.02
kei6	0.19	0.19	0.06	0.07	0.17	0.21	0.34	0.24
kei7	0.03	0.02	0.00	0.00	0.01	0.01	0.07	0.05
kor1	8.47	26.28	681.91	1395.34	89.48	7.58	0.05	0.41
kor2	807.35	3336.73	2164.20	2865.46	399.10	12.08	1000.95	2959.69
kor12	1.01	0.01	4.21	1.23E11	1.01	0.00	1.00	0.00
ppb	29.40	7.51	29.86	5.43	27.78	3.11	37.07	9.11
tow	34.91	3.70	38.77	4.30	23.16	0.66	47.73	6.77
vla1	0.09	0.06	0.08	0.05	0.03	0.02	0.17	0.07
vla4	0.11	0.02	1.23	3.44E10	0.05	0.00	0.17	0.01
wineR	0.64	0.03	0.64	0.03	0.63	0.01	0.67	0.05
wineW	0.73	0.02	0.74	0.02	0.70	0.00	0.76	0.03
yac	1.88	0.62	1.31	0.54	2.48	0.31	4.61	1.61

Table 5.3: Median and IQR of the resulting function size for each algorithm according to the experimental strategy presented in Table 4.1.

Dataset	SSR		GRSR		GSGP		GP	
	Med.	IQR	Med.	IQR	Med.	IQR	Med.	IQR
air	641.0	54.0	883.0	78.5	5.1E135	2.2E137	99.0	19.5
bio	252.0	49.5	495.0	64.5	9.6E166	2.0E170	77.0	23.5
con	516.0	75.0	543.0	42.5	3.6E247	1.6E250	77.0	23.0
cpu	412.0	41.0	410.0	40.5	1.8E259	5.5E259	85.0	16.0
eneC	554.0	73.5	598.0	49.0	1.1E241	1.3E242	80.0	19.5
eneH	557.0	87.0	608.0	73.0	2.6E242	9.4E243	79.0	21.0
for	374.0	59.0	414.0	61.0	6.1E219	5.4E220	82.0	23.0
kei5	642.0	64.5	632.0	85.5	1.1E234	9.8E235	57.0	17.5
kei6	557.0	66.5	628.0	69.5	1.1E271	4.2E271	67.0	27.0
kei7	587.0	90.5	601.0	61.5	1.5E265	2.0E266	71.0	21.5
kor1	595.0	151.0	827.0	292.0	4.6E275	3.7E281	71.0	12.0
kor2	405.0	41.5	457.0	50.5	2.4E281	3.0E281	100.0	16.0
kor12	384.0	42.5	494.0	382.5	4.9E284	4.5E286	5.0	48.0
ppb	283.0	97.5	410.0	57.0	1.8E241	1.3E242	65.0	27.5
tow	619.0	58.5	621.0	60.5	2.9E207	3.0E208	66.0	20.0
vla1	561.0	75.0	578.0	54.5	8.9E245	6.2E247	77.0	17.0
vla4	626.0	53.5	482.0	165.0	1.1E239	1.5E240	59.0	24.0
wineR	437.0	44.5	496.0	59.0	1.8E233	1.7E234	66.0	16.0
wineW	540.0	55.0	559.0	40.0	5.5E231	2.7E232	67.0	14.0
yac	595.0	74.5	698.0	66.0	1.7E220	3.9E221	85.0	25.5

statistical difference in size or performance in relation to SSR. These results indicate SSR is a better option than the canonical GP when the main goal is to obtain more

Table 5.4: P-values obtained by the statistical analysis of the algorithms' performance and resulting function size. The symbol ▲(▼) indicates the method in the column is statistically better (worse) than the SSR with 95% confidence.

With reference to	Adjusted Friedman p-value	Finner APV GP	GSGP	GRSR
RMSE	**0.0068**	0.0300 ▼	0.6883	0.6883
Function size	**0.0000**	0.0073 ▲	0.0000 ▼	0.0864

accurate functions and the size of the solutions is not relevant. Furthermore, SSR is capable of producing solutions with similar performance to those produced by GSGP, but up to two hundred orders of magnitude smaller, since the size of GSGP individuals grows exponentially in relation to the number of generations.

5.4 Conclusions

In this chapter we presented the Sequential Symbolic Regression (SSR), a new strategy to perform symbolic regression by iteratively learning solutions from a transformed set of problems. The method uses the geometric semantic crossover operator to concatenate the solutions obtained, maintaining the size of the solutions in a treatable level.

Experiments were run in a test bed composed of 20 datasets, and results compared with a canonical GP, GSGP and GRSR. When median RMSE was used as the performance metric, the results showed SSR performs significantly better than the GP and it is equivalent to GSGP and GRSR. When comparing the size of the resulting functions produced by the algorithms, SSR solutions are significantly smaller than those generated by GSGP and larger than the solutions produced by GP. These results indicate that SSR is a good alternative to GP when the performance is more important than the size of the functions; and to GSGP, where the RMSE obtained is similar but SSR generates smaller solutions, restraining the exponential grown caused by the indiscriminate use of geometric semantic operators by GSGP.

Chapter 6

Geometric Dispersion Operators

One of the issues presented in Section 1.1 regarding GSGP states that, when using only the geometric semantic crossover operator, the set of possible individuals generated during evolution is delimited by the convex hull of the current population's semantics [Pawlak, 2015]. Hence, if the target output is not within the convex hull, the algorithm will not be able to reach it.

The geometric semantic mutation operator appears as the solution to this problem, as it can expand the semantic search space and, consequently, the population's convex hull. However, depending on how far from the target solution the individuals in the initial population are, GSGP may take a prohibitive amount of time to reach the relevant region of the search space where the target output is.

In this context, this chapter proposes a new class of geometric semantic operators, named geometric dispersion (GD) operators. GD operators take as input a single individual and move it in the semantic space aiming to better distribute the population around the target output. By doing that GD increases the chances of generating a convex hull from the population semantics that contains the target output vector. In contrast with GSM, which randomly moves the individual within a hypersphere around it, the geometric dispersion operators deterministically select a region of the semantic space where the individual should be moved to.

6.1 Convex Hull and GSGP

Previous works on GSGP have proposed different approaches to take advantage of the properties of the geometric semantic space to improve search. However, to the best of our knowledge, so far only two have investigated ways to increase the area covered by

the population's convex hull, specially focusing on the coverage of the initial population, as discussed in this section.

Pawlak [2015] proposes the Competent Initialization (CI) method, which aims to increase the convex hull of the initial population. The algorithm adopts a generalized version of the Semantically Driven Initialization (SDI) method [Beadle and Johnson, 2009a], initially proposed for non-geometric spaces, to generate individuals semantically distinct. SDI picks a node randomly from the function set to combine individuals already in the population. If the resulting program has semantics different from the other members of the population, it is accepted. Otherwise, the method makes a new attempt of generating an individual. The process continues until a semantically distinct individual is created, in a trial-and-error strategy. CI, on the other hand, accepts the semantically distinct individual only if it is not in the current convex hull. The main drawback of this method is the possible waste of resources, since individuals are randomly created, evaluated and discarded when they are semantically similar to an existing member of the population or when it is already in the population's convex hull.

The Semantic Geometric Initialization (SGI) [Pawlak and Krawiec, 2016], on the other hand, generates a set S of semantics, such that the desired output is guaranteed to belong to the convex hull of S. These semantics are generated by adding or subtracting an offset to *out* in different combinations of dimensions of the semantic space. Then, for each semantics $s \in S$, the method generates an individual whose semantics is equal to s. The synthesis of these individuals is domain-dependent and the authors present methods to generate individuals for symbolic regression problems—by polynomial interpolation—and for boolean problems—by a boolean formula. The experimental analysis indicates that SGI can achieve training error significantly smaller than the ramped half-and-half (RHH) method in symbolic regression and boolean problems. However, the test error achieved by SGI is significantly higher than the error achieved with RHH, which indicates that SGI is very susceptible to overfitting.

6.2 Geometric Dispersion Operators

In this section we present a generic geometric dispersion (GD) framework along with two implementations, the multiplicative geometric dispersion (MGD) [Oliveira et al., 2016c] and the additive geometric dispersion (AGD) [Oliveira et al., 2016d] operators.

6.2.1 A Framework for Geometric Dispersion Operators

This section presents a general framework for geometric dispersion (GD)[1] operators aiming to redistribute the population around the desired output vector *out* in the semantic space. These operators move a given individual to the region of the semantic space around *out* with the lowest concentration of individuals in order to, hopefully, modify the convex hull of the population to contain *out*.

Let GT (greater than) and LT (less than) be n-element arrays, where n is the number of dimensions of the semantic space and the i-th element corresponds to the number of individuals p in the current population P where $s(p)[i] > out[i]$ and $s(p)[i] < out[i]$, respectively. If $LT[i] < GT[i]$, the population is unbalanced with more individuals in the right side of the desired output vector in the i-th dimension of the semantic space. Otherwise, if $LT[i] > GT[i]$, the unbalance occurs in the opposite side, i.e., the population is concentrated on the left side of *out* in the dimension i.

GD operators adopt a greedy strategy to redistribute the population around the desired output vector by moving individuals to balance the distribution of the population on both sides of *out*, in each dimension of S—i.e., the operator moves the individuals in order to approximate the values of $GT[i]$ and $LT[i]$ for $i = 1, 2, \ldots, n$.

When we know the region of the semantic space around *out* where we want to have individuals shifted to, different methods can be used to move the individual p. GD operators do that by applying a constant m to p through a mathematical operation $\oplus$, in the form $m \oplus p$. The movement performed by the GD operator depends directly of the chosen operation for $\oplus$. Thus, the value of m must be chosen such that the displacement of p benefits the largest number of dimensions.

The process of finding such value is equivalent to finding m that solves the inequality system:

$$\begin{cases} m \oplus s(p)[1] \lesseqgtr_1 out[1] \\ m \oplus s(p)[2] \lesseqgtr_2 out[2] \\ \ldots \\ m \oplus s(p)[k] \lesseqgtr_k out[k] \end{cases} \tag{6.1}$$

where '$\lesseqgtr_i$' is an inequality operator ('$<$' or '$>$') chosen according to the asymmetry of the population. If $LT[i] < GT[i]$, the individual should be moved to the left side of *out*

[1] Oliveira et al. [2016c] presents an operator called Geometric Dispersion (GD) operator. However, this operator is a particular case of the framework presented in this chapter. Hence, hereafter the operator of Oliveira et al. [2016c] is referred as Multiplicative Geometric Dispersion (MGD) operator in contrast to the GD framework.

and '$\lesseqgtr_i$' is replaced by '$<$'. Otherwise, if $LT[i] > GT[i]$, the individual should be moved to the opposite side of *out* and '$\lesseqgtr_i$' is replaced by '$>$'. Note that when $GT[i] = LT[i]$, no inequality is added to the system. Therefore the number of inequalities in Eq. 6.1 is smaller or equal to the number of dimensions of the semantic space, i.e., $k \leq n$.

However, due to the large number of inequalities in the system, usually it does not admit feasible solutions. Thus, instead of finding a value for m that satisfies all inequalities, the operator finds one that maximizes the number of satisfied inequalities. In [Oliveira et al., 2016c], we present algorithms to construct the system of inequalities and find the value of m that satisfies the largest number of inequalities when the mathematical operation adopted is the multiplication, i.e., $\oplus$ is replaced by $\times$ (times). In another work [Oliveira et al., 2016d], we extend these algorithms and present a general framework, called Geometric Dispersion (GD), to move individuals through the semantic space in order to distribute the population around *out*.

GD is independent of the arithmetic operation adopted in the inequalities. The only requirements are that the operation is binary and allows inverse. Let $\oplus$ and $\ominus$ be a binary operation and its inverse, respectively, m can be isolated in the left side of the system of inequalities of Eq. 6.1 as showed in Eq. 6.2, such that the operator can find the value that satisfies the largest number of inequalities.

$$\begin{cases} m \lesseqgtr_1 out[1] \ominus s(p)[1] \\ m \lesseqgtr_2 out[2] \ominus s(p)[2] \\ ... \\ m \lesseqgtr_k out[k] \ominus s(p)[k] \end{cases} \tag{6.2}$$

Given the arrays GT and LT, Algorithm 5 checks each dimension i for an unbalanced distribution, i.e., where $GT[i] \neq LT[i]$. When these values differ the method add a new inequality to the system, represented by a bound value that should be satisfied.

The sign '$\lesseqgtr_i$' of the inequalities is defined according to the distribution of the population in the verified dimension (lines 6-9). The next step of the method is to isolate m in the left side of the inequality and store the value of the right side in *bound* (lines 10-11). There are a few considerations in this step, according to the arithmetic operation used in the inequalities. E.g., if GD uses multiplication ($\oplus$ is replaced by $\times$), as presented by Oliveira et al. [2016c], the method must check for division by zero and negative value on the left side of the inequality. When a division by zero is found, the algorithm ignores the inequality. When the left side is negative, both sides of the inequality are multiplied by -1, inverting the inequality sign.

Algorithm 5: GD procedure to build the system of inequalities

```
   Input: Individual program (p), desired output (out),
1          population distribution (GT, LT)
2  Procedure buildInequalities(p, out, GT, LT)
3      B ← {};
4      for i ← 1 to |s(p)| do                          // Calculate the bounds
5          if GT[i] ≠ LT[i] then
6              if GT[i] > LT[i] then
7                  sign ← 'lessThan' ;                 // '≦_i' is replaced by '<'
8              else
9                  sign ← 'greaterThan' ;              // '≦_i' is replaced by '>'
               // m ≦_i out[i] ⊖ s(p)[i]
10             Isolate m in the left side of the inequality;
11             bound ← right side value ;              // bound ← out[i] ⊖ s(p)[i]
12             if sign = 'lessThan' then
13                 Add bound to B as upper bound;
14             else
15                 Add bound to B as lower bound;
16     return B
```

The sign of the inequalities is used to define the type of bound (lines 12-15). If the sign is '<', the value of m should be smaller than *bound* (it is an upper bound). Otherwise, m should be greater than *bound* (it is a lower bound). The bounds and their types are used to compute the value of m in Algorithm 6.

Algorithm 6 first sorts B by value in ascending order. The auxiliary variables *maxSatisfied*, *index* and *cSatisfied* store the number of inequalities satisfied by the best bound for m found so far, its index and the number of inequalities satisfied by the bound examined in the current iteration, respectively. The method starts by considering the interval before the first bound, i.e., $(-\infty, B[1].value)$. If a value from this interval is picked for m, all the upper bounds are satisfied, i.e., $maxSatisfied = n_{ub}$ (line 3). It then iterates over B counting the number of upper and lower bounds satisfied by each interval $(B[i], \infty)$ (lines 6-14). If the examined value corresponds to an upper bound, we decrement the *cSatisfied* counter, since the interval in the right side of the bound does not satisfy it. On the other hand, if the examined value corresponds to a lower bound, the right side interval satisfies the bound and *cSatisfied* is incremented.

After finding the best interval for m, the procedure assigns an actual value for m (lines 15-25). If the best interval corresponds to $(-\infty, B[1])$ or to $(B[|B|], \infty)$, m

Algorithm 6: GD procedure to find m

Input: Set of bounds for m (B), shift control variable (*shiftOne*)

```
1  Procedure findM(B, shiftOne)
2      n_ub ← number of upper bounds in B;
3      maxSatisfied ← cSatisfied ← n_ub;
4      index ← 0;
5      Sort B by value in ascending order;
6      for i ← 1 to |B| do                    // Find the best interval for m
7          bound ← B[i];
8          if bound is a lower bound then
9              cSatisfied ← cSatisfied + 1;
10             if cSatisfied > maxSatisfied then
11                 maxSatisfied ← cSatisfied;
12                 index ← i;
13         else
14             cSatisfied ← cSatisfied − 1;
15     if index = 0 then                                  // Calculate m
16         if B is empty then                        // No need to move p
17             m ← 1;
18         else
19             m ← getLeftExtreme(B, shifOne);
20     else
21         if index = |B| then
22             m ← getRightExtreme(B, shifOne);
23         else
24             δ ← B[index + 1].value − B[index].value;
               // rnd() returns a random value in (0, 1)
25             m ← B[index].value + δ · rnd();
26     return m
```

takes the output of *getLeftExtreme* and *getRightExtreme*, respectively. Otherwise, the method selects a random value in the interval $(B[index], B[index + 1])$.

Algorithms 7 and 8 present the procedures *getRightExtreme* and *getLeftExtreme*, respectively. The control variable *shiftOne* indicates if the methods should shift values in the extreme of the interval by one or shift the values by a random number proportional to the closest interval defined in B.

The value of m returned by Algorithm 6 is then used to move the individual p in the semantic space. GD is applied during the evolution, at every generation, right before other genetic semantic operators (crossover and mutation).

Algorithm 7: *getRightExtreme* procedure

Input: Set of bounds for m (B), control variable *shiftOne*

```
1 Procedure getRightExtreme(B, shiftOne)
2     if shifOne=TRUE or |B| < 2 then
3         return B[|B|] + 1
4     else
5         δ ← B[|B|].value − B[|B| − 1].value;
6         return B[|B|] + δ · rnd()
```

Algorithm 8: *getLeftExtreme* procedure

Input: Set of bounds for m (B), control variable *shiftOne*

```
1 Procedure getLeftExtreme(B, shiftOne)
2     if shifOne=TRUE or |B| < 2 then
3         return B[1] − 1
4     else
5         δ ← B[2].value − B[1].value;
6         return B[1] − δ · rnd()
```

One thing we realized during preliminary experiments was that moving individuals around the semantic spaces in the initial generations has a high positive impact in search. However, this behaviour is not maintained during the whole evolutionary process, leading to negative or no effects in later generations. For this reason, we proposed to dynamically adjust the probability *pgd* of applying GD as follows:

$$pgd = pgd_0 \cdot exp\left(\frac{-\alpha \cdot g}{g_{max}}\right) \quad , \tag{6.3}$$

where pgd_0 is the base probability, α is the decay rate, g is the current generation index and g_{max} is the total number of generations. Equation 6.3 ensures the probability of applying the operator decays exponentially with the generations.

6.2.2 Multiplicative Geometric Dispersion

The first version of the geometric dispersion operator proposed in [Oliveira et al., 2016c], here called Multiplicative Geometric Dispersion (MGD), is an implementation of the GD framework where the constant m is multiplied by the semantics of the individual p. MGD manipulates inequality systems as given by Eq. 6.1 and isolates m in the left side as presented by Eq. 6.2, where $\oplus$ and $\ominus$ are replaced by $\times$ and $\div$, respectively.

The multiplicative operation applied to the individual p is geometrically equivalent to moving it through the line crossing both $s(p)$ and the origin of $\mathcal{S}$.

As discussed above, when isolating m in the i-th dimension, MGD must consider two special cases: (1) if $s(p)[i] = 0$, isolating m implies in division by zero and the operator ignores the inequality and (2) if $s(p)[i] < 0$, the inequality is multiplied by -1 and the inequality sign is inverted.

6.2.3 Additive Geometric Dispersion

In addition to MGD, Oliveira et al. [2016d] presents a geometric dispersion operator based on addition. The Additive GD (AGD) moves a given individual p through the line $L = \{s(p) + t : t \in \mathbb{R}\}$, with $L \subset \mathcal{S}$, in order to redistribute the population around *out*. The inequalities used by AGD are in the form $m + s(p)[i] \lesseqgtr_i out[i]$, which result in $m \lesseqgtr_i out[i] - s(p)[i]$, where i is the dimension analysed.

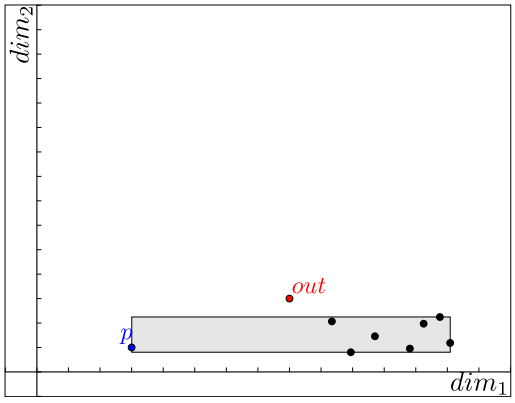
The use of different mathematical operations within the GD operators allows them to explore different regions of $\mathcal{S}$. Figure 6.1 and 6.2 present examples in Manhattan and Euclidean semantic spaces, respectively, where AGD can improve the distribution of the individuals around *out*, but MGD cannot. Figures 6.3 and 6.4, on the other hand, present the opposite—they present scenarios in Manhattan and Euclidean semantic spaces, respectively, where MGD can improve the distribution around the desired output vector and AGD cannot.

6.3 Experimental Analysis

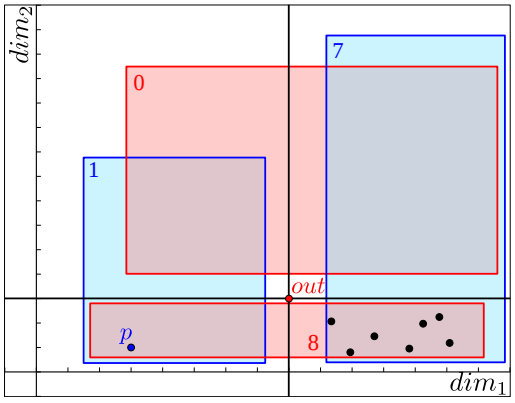
In this section we present an experimental analysis of the geometric dispersion operators within the GSGP in a diversified collection of real-world and synthetic datasets. We analysed four different versions of GD operators within GSGP:

- **GSGP+M**: GSGP with MGD and *shiftOne* = `TRUE`;
- **GSGP+MR**: GSGP with MGD and *shiftOne* = `FALSE`;
- **GSGP+A**: GSGP with AGD and *shiftOne* = `TRUE`; and
- **GSGP+AR**: GSGP with AGD and *shiftOne* = `FALSE`.

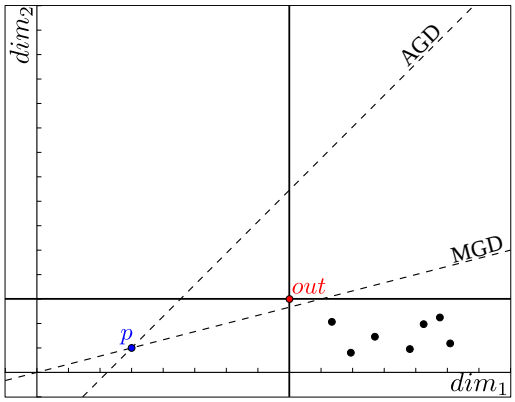
The experimental test bed is composed by a collection of sixteen datasets selected from the test bed presented in Section 4.2.3 in Table 4.1—the selected collection is presented in Table 6.1. All executions used a population of 1,000 individuals evolved



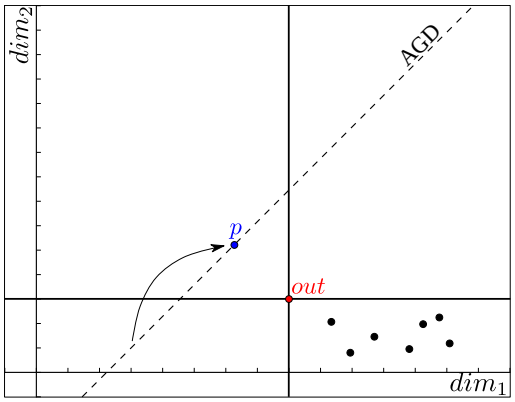
(a) Population and desired output vector. The polygon depicts the population's convex hull.



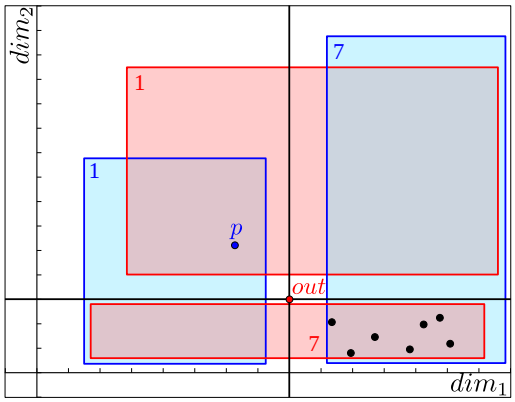
(b) Distribution of individuals around *out* regarding dim_1 (blue) and dim_2 (red). The numbers indicate the frequency of individuals on each side of *out* for each dimension.



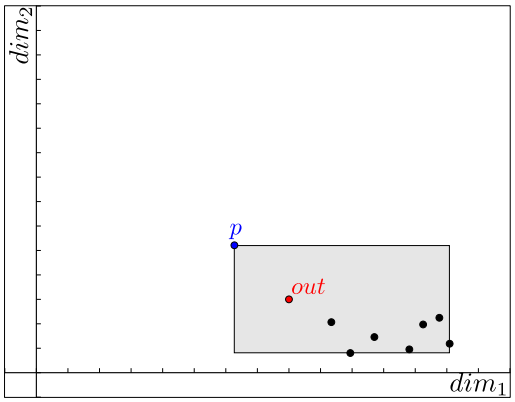
(c) The lines represent the regions reachable by AGD and MGD operators when applied to the individual p.



(d) AGD moves the individual p along the line to a new position.

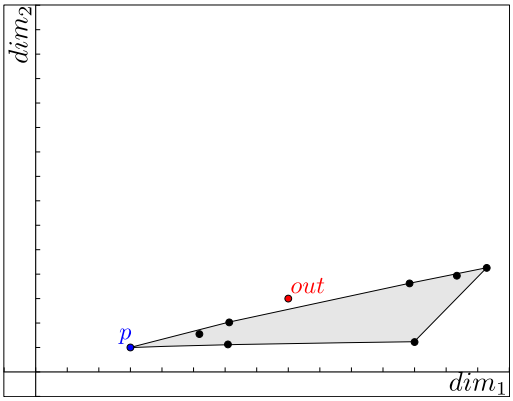


(e) The movement improves the individuals' distribution in dim_2.

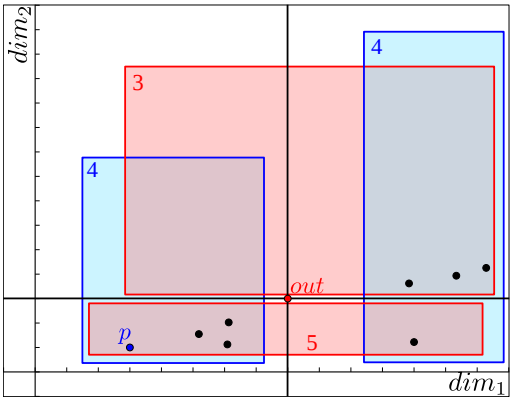


(f) The resulting population's convex hull embraces the target output.

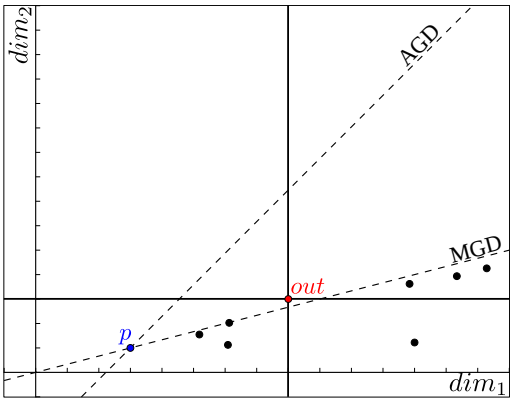
Figure 6.1: Example where the AGD operator can improve the population's distribution around *out* but MGD cannot, in a two-dimensional Manhattan semantic space.



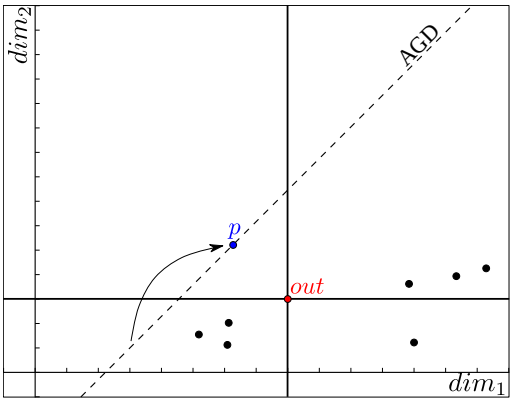
(a) Population and desired output vector. The polygon depicts the population's convex hull.



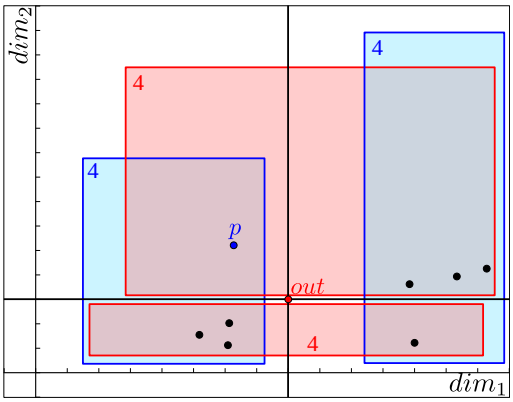
(b) Distribution of individuals around *out* regarding dim_1 (blue) and dim_2 (red). The numbers indicate the frequency of individuals on each side of *out* for each dimension.



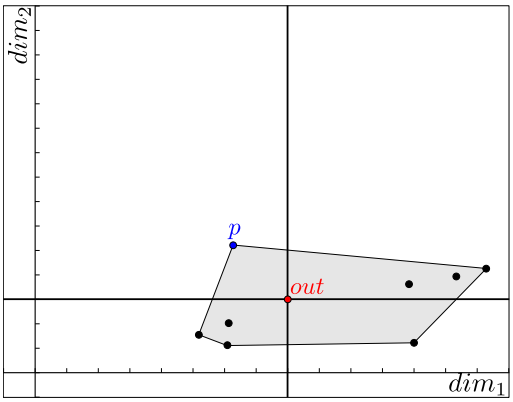
(c) The lines represent the regions reachable by AGD and MGD operators when applied to the individual p.



(d) AGD moves the individual p along the line to a new position.

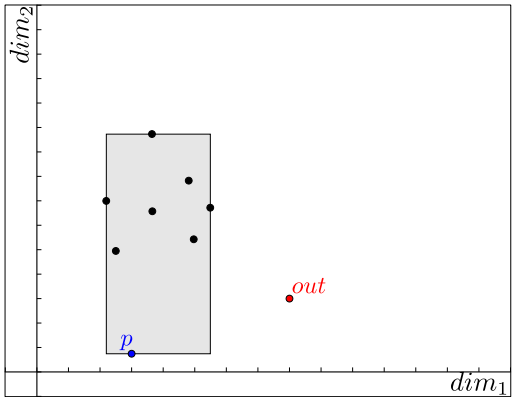


(e) The movement improves the individuals' distribution in dim_2.

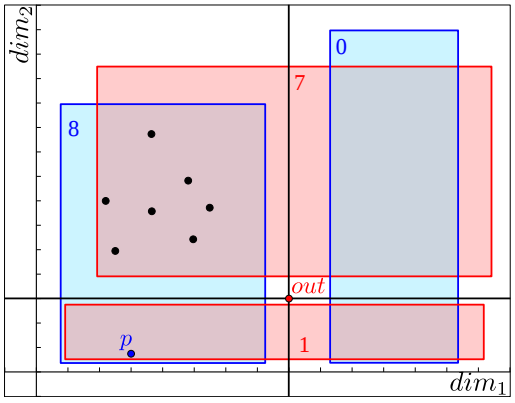


(f) The resulting population's convex hull embraces the target output.

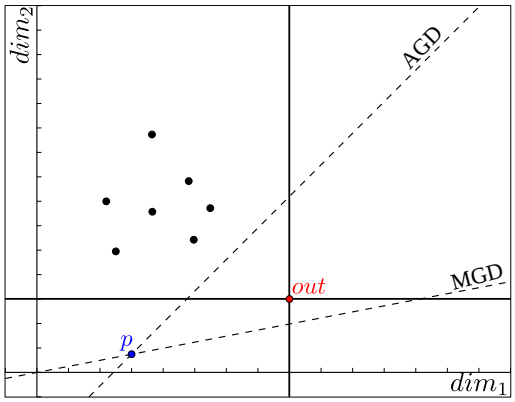
Figure 6.2: Example where the AGD operator can improve the population's distribution around *out* but MGD cannot, in a two-dimensional Euclidean semantic space.



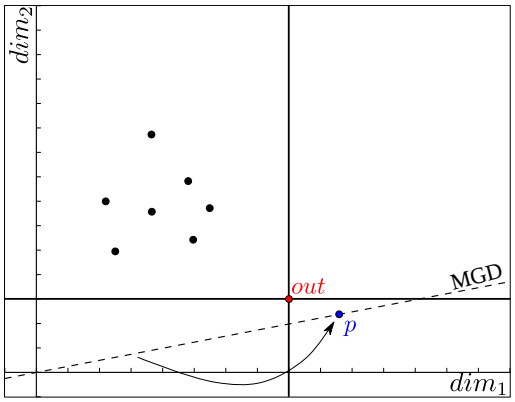
(a) Population and desired output vector. The polygon depicts the population's convex hull.



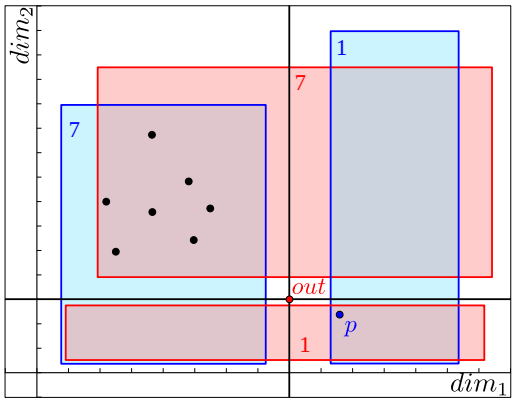
(b) Distribution of individuals around *out* regarding dim_1 (blue) and dim_2 (red). The numbers indicate the frequency of individuals on each side of *out* for each dimension.



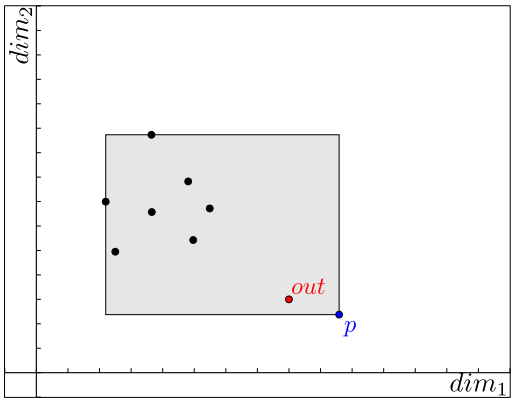
(c) The lines represent the regions reachable by AGD and MGD operators when applied to the individual p.



(d) MGD moves the individual p along the line to a new position.

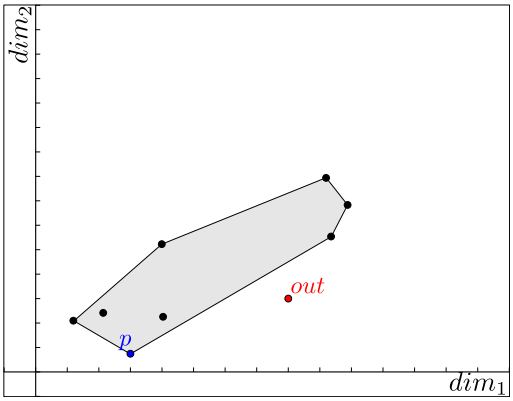


(e) The movement improves the individuals' distribution in dim_1.

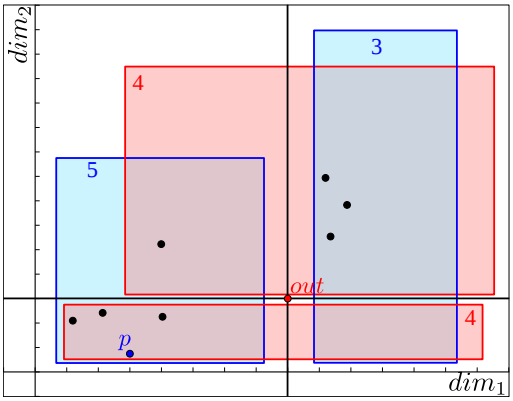


(f) The resulting population's convex hull embraces the target output.

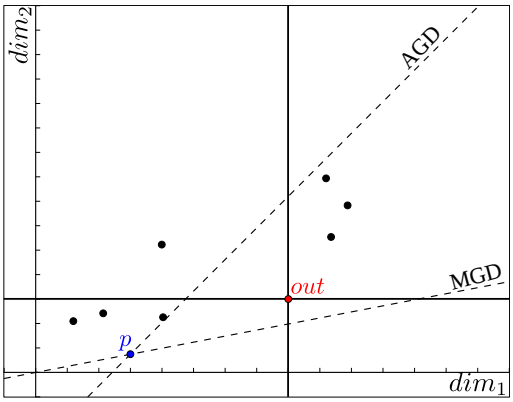
Figure 6.3: Example where the MGD operator can improve the population's distribution around *out* but AGD cannot, in a two-dimensional Manhattan semantic space.



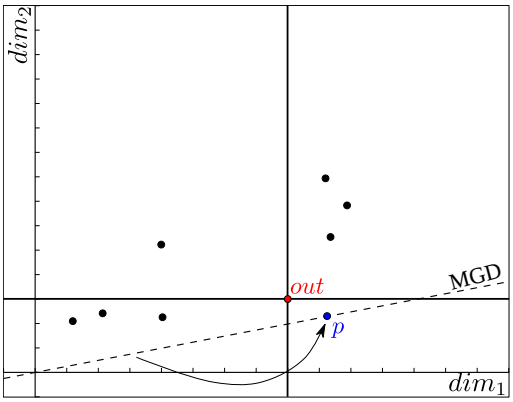
(a) Population and desired output vector. The polygon depicts the population's convex hull.



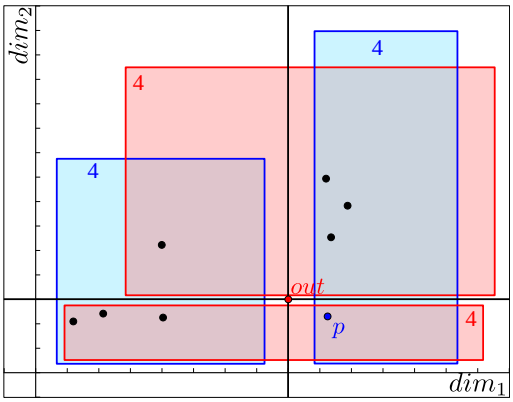
(b) Distribution of individuals around *out* regarding dim_1 (blue) and dim_2 (red). The numbers indicate the frequency of individuals on each side of *out* for each dimension.



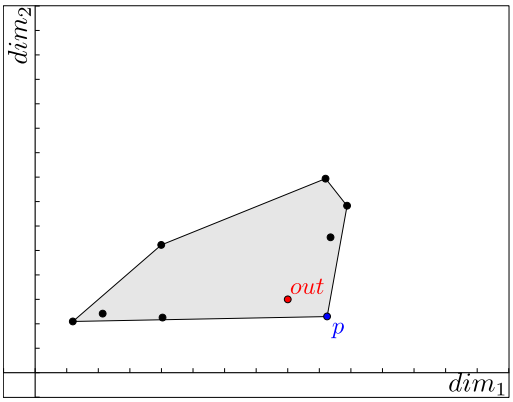
(c) The lines represent the regions reachable by AGD and MGD operators when applied to the individual p.



(d) MGD moves the individual p along the line to a new position.



(e) The movement balances the individuals' distribution in dim_1.



(f) The resulting population's convex hull embraces the target output.

Figure 6.4: Example where the MGD operator can improve the population's distribution around *out* but AGD cannot, in a two-dimensional Euclidean semantic space.

for 2,000 generations with tournament selection of size 10. The grow method was adopted to generate the random functions inside the geometric semantic crossover and mutation operators, and the ramped half-and-half method used to generate the initial population, both with maximum individual depth equal to 6. As before, the function set includes three binary arithmetic operators $(+, -, \times)$ and the Analytic Quotient (AQ) [Ni et al., 2013] as an alternative to the arithmetic division. The terminal set includes the variables of the problem and constant values randomly picked from the interval $[-1, 1]$. GSGP employed the geometric semantic crossover for fitness function based on Manhattan distance[2] (GSXM) and mutation (GSM) operators, as presented in [Castelli et al., 2015a], both with probability 0.5. The mutation step was set to 10% of the standard deviation of the training data output.

6.3.1 Parameter Tuning

As we are interested in understanding the impact of the GD operators, we fixed the other GSGP parameters and focused on looking at the results as we varied the probability values of the GD operator. The experiments of this section employ the GSGP+M to evaluate the effect of different configurations on the search.

We tested the application of MGD with different initial probabilities, and here we show the best results, which were obtained by setting pgd_0 to 0.2, 0.4 and 0.6. For these three values, we varied the parameter α from Equation 6.3, which defines the decay rate of pgd_0 along the generations. Notice that when α equals 0, the decay function is not used, and pgd_0 becomes constant along the generations. Figure 6.5 shows the decay curves for the different combinations of pgd_0 and α adopted in our experiments.

Table 6.1 presents the results of Root Mean Squared Error (RMSE) for these combinations of parameters in the training set. The results show that lower values for pgd_0 $(0.2, 0.4)$, allied with higher α values (10) tend to reduce the training RMSE. This fact indicates the application of the GD operator with low probability only in the early generations can reduce the training RMSE in relation to other combination of values.

6.3.2 Experimental Results

After the parameter tuning, the use of GD operators was evaluated considering three metrics: (i) the effect on the RMSE compared to GSGP without GD operators; (ii) the distribution of the individuals along the dimensions of the semantic space; and (iii) the execution time. We start by discussing the error results. Table 6.2 and 6.3 present the

[2] As the experimental analysis of the Chapter 4 showed, GSXM presents better results than GSXE.

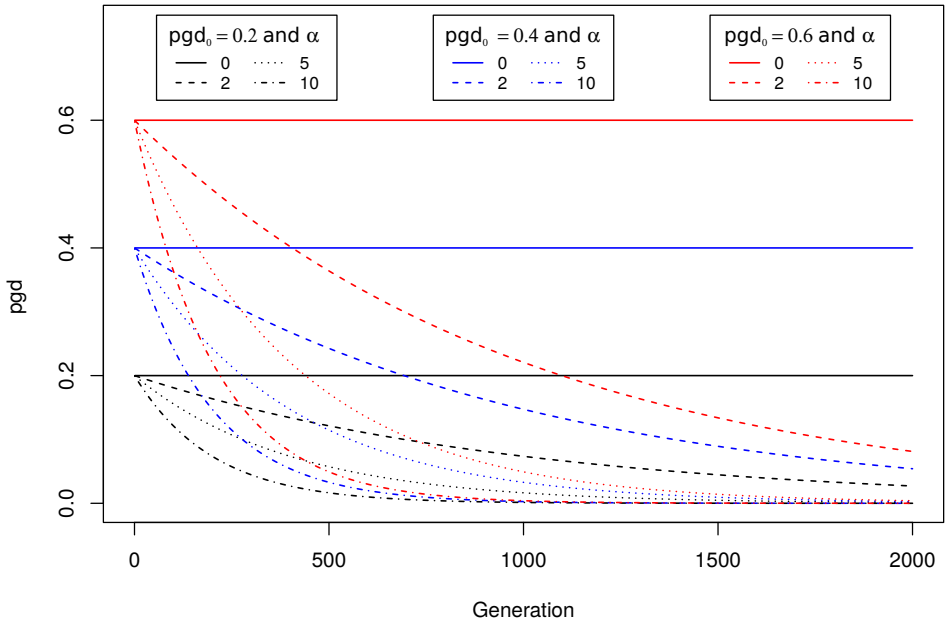


Figure 6.5: Value of *pgd* over the generations for the different combinations of pgd_0 and α used during the tuning stage.

Table 6.1: Median training RMSE of the GSGP+M with different values of α and pgd_0 for the adopted test bed. The smallest RMSE for each dataset is presented in bold.

Dataset	$pgd_0 = 0.2$				$pgd_0 = 0.4$				$pgd_0 = 0.6$			
	α				α				α			
	0	2	5	10	0	2	5	10	0	2	5	10
air	1.518	1.475	**1.452**	1.460	1.612	1.491	1.457	1.457	1.844	1.531	1.472	1.465
bio	1.303	1.103	1.032	1.046	1.854	1.172	1.059	**1.024**	3.138	1.382	1.089	1.035
con	2.819	2.771	2.750	2.746	2.937	2.782	2.750	2.746	3.097	2.829	2.788	**2.732**
cpu	1.151	0.983	1.020	1.036	1.290	1.104	1.008	**0.969**	1.487	1.269	1.093	1.086
eneC	1.063	1.020	0.997	1.000	1.120	1.052	1.004	**0.985**	1.244	1.077	1.030	1.009
eneH	0.758	0.722	0.733	**0.703**	0.819	0.771	0.744	0.730	0.933	0.801	0.745	0.737
for	3.461	3.407	3.182	3.225	3.968	3.511	3.315	**3.162**	4.472	3.589	3.311	3.242
kei5	0.048	0.046	0.045	0.045	0.043	0.045	0.044	0.043	**0.026**	0.026	0.031	0.035
kei6	**0.007**	0.007	0.007	0.007	0.009	0.009	0.010	0.009	0.010	0.010	0.009	0.010
kei7	0.017	0.017	**0.016**	0.017	0.020	0.018	0.019	0.019	0.018	0.019	0.017	0.017
ppb	0.004	0.003	**0.003**	0.003	0.007	0.004	0.003	0.003	0.005	0.006	0.004	0.003
tow	19.150	18.672	18.574	**18.366**	20.500	19.101	18.781	18.605	22.127	19.774	18.941	18.688
vla1	0.012	0.012	0.012	0.012	0.013	0.012	0.012	0.012	0.015	0.012	0.012	**0.012**
vla4	0.040	0.039	0.038	**0.038**	0.043	0.040	0.039	0.038	0.048	0.040	0.039	0.039
wineR	0.340	0.326	0.322	**0.319**	0.373	0.338	0.327	0.322	0.423	0.350	0.330	0.323
wineW	0.547	0.539	0.535	**0.533**	0.567	0.545	0.538	0.535	0.594	0.553	0.541	0.536

median training and test RMSE and respective IQR (Interquartile Range), according to 50 executions. The results regarding GSGP with GD operators were obtained with the values of α and pgd_0 generating the smallest median training RMSE in each of the datasets, presented in bold in Table 6.1.

Table 6.4 summarizes the statistical differences regarding the results presented in Tables 6.2 and 6.3. The table presents the number of datasets where the method in

Table 6.2: Training RMSE (median and IQR) obtained by the algorithms for each dataset.

Dataset	GSGP		GSGP+A		GSGP+AR		GSGP+M		GSGP+MR	
	Med.	IQR	Med.	IQR	Med.	IQR	Med.	IQR	Med.	IQR
air	7.885	0.527	1.873	0.057	1.900	0.052	1.886	0.041	1.890	0.051
bio	9.893	0.652	9.653	0.552	9.706	0.547	9.695	0.690	9.794	0.703
con	3.647	0.138	3.659	0.136	3.644	0.150	3.654	0.118	3.634	0.109
cpu	6.126	0.665	6.149	0.977	6.223	1.067	6.151	0.905	6.215	0.790
eneC	1.257	0.070	1.282	0.065	1.267	0.057	1.271	0.066	1.255	0.052
eneH	0.802	0.113	0.790	0.084	0.789	0.123	0.798	0.083	0.764	0.084
for	30.737	4.626	31.684	4.858	31.247	4.490	30.967	4.201	31.534	4.156
kei5	0.045	0.003	0.063	0.006	0.062	0.006	0.026	0.008	0.026	0.009
kei6	0.007	0.005	0.007	0.007	0.007	0.005	0.007	0.006	0.006	0.005
kei7	0.017	0.010	0.017	0.010	0.016	0.009	0.016	0.009	0.014	0.009
ppb	0.917	0.266	0.930	0.241	0.924	0.274	0.954	0.305	0.937	0.202
tow	20.436	0.610	20.558	0.704	20.587	0.610	20.405	0.621	20.472	0.404
vla1	0.012	0.002	0.012	0.002	0.012	0.001	0.012	0.002	0.012	0.002
vla4	0.038	0.001	0.038	0.002	0.038	0.002	0.038	0.001	0.038	0.002
wineR	0.493	0.011	0.494	0.012	0.494	0.010	0.494	0.011	0.495	0.010
wineW	0.641	0.003	0.642	0.004	0.642	0.004	0.642	0.003	0.641	0.003

the row is statistically better than the method in the column regarding the test RMSE, according to a Wilcoxon test with 95% confidence level. The results indicate the search performed by GSGP benefits from the dispersion provided by the dispersion operators, as pointed out by the score of GSGP in relation to GD configurations. Regarding the use of the shift one algorithm or the random method to compute the values of m in the extremes, there are no significant differences on the dispersion operators. Lastly the results indicate that overall the multiplicative version of the geometric dispersion operator performs better than the additive counterpart.

The second aspect we evaluate is whether GD actually improves the distribution of the population around the desired output *out* or not. For that, we propose a new measure, called Mean Dimension Distribution (MDD). For each dimension d of the training set, it is defined as

$$mdd(P,d) = abs\left(\frac{1}{|P|}\sum_{i=1}^{|P|} ge(s(p_i)[d], out[d]) - 0.5\right) \quad , \tag{6.4}$$

where P is the population, $|P|$ is the population size, p_i is its i-th individual, $ge(a,b)$ returns 1 if $a \geq b$ and 0 otherwise and $abs(a)$ returns the absolute value of a. MDD is defined in $[0, 0.5]$. A $mdd(P,d) = 0$ means that P is evenly distributed in dimension d—i.e., half the individuals are greater than or equal to and half are smaller than *out* in dimension d—while $mdd(P,d) = 0.5$ means the population is badly distributed—i.e.,

Table 6.3: Test RMSE (median and IQR) obtained by the algorithms for each dataset.

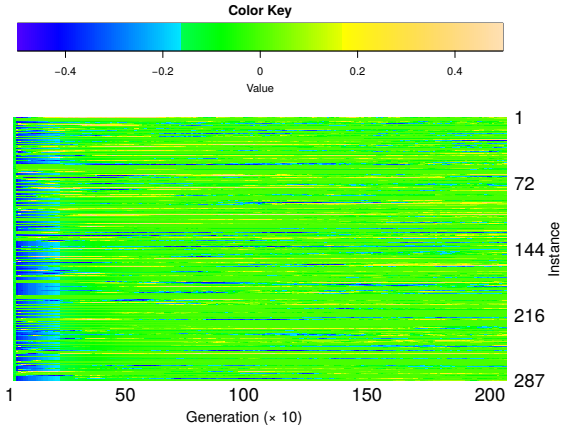
Dataset	GSGP		GSGP+A		GSGP+AR		GSGP+M		GSGP+MR	
	Med.	IQR	Med.	IQR	Med.	IQR	Med.	IQR	Med.	IQR
air	8.417	0.757	2.154	0.237	2.131	0.272	2.131	0.243	2.152	0.210
bio	30.736	2.326	31.139	4.170	30.682	4.189	30.860	4.426	30.619	3.773
con	5.394	0.642	5.285	0.624	5.244	0.575	5.144	0.635	5.054	0.489
cpu	30.917	15.185	32.804	13.166	32.400	16.182	30.837	14.563	32.027	16.790
eneC	1.515	0.147	1.553	0.151	1.489	0.180	1.531	0.159	1.486	0.194
eneH	0.956	0.185	0.919	0.157	0.928	0.136	0.971	0.128	0.933	0.169
for	51.632	48.166	52.026	48.626	51.483	50.444	50.227	48.373	50.590	48.762
kei5	0.049	0.005	0.066	0.006	0.065	0.007	0.028	0.009	0.027	0.010
kei6	0.398	0.339	0.275	0.228	0.293	0.203	0.281	0.282	0.250	0.166
kei7	0.018	0.010	0.019	0.010	0.018	0.010	0.017	0.009	0.015	0.008
ppb	28.740	5.290	27.337	5.031	28.139	5.630	28.568	6.170	27.969	5.849
tow	21.920	1.272	21.979	1.264	21.826	1.263	21.769	1.252	21.871	1.134
vla1	0.044	0.030	0.041	0.030	0.046	0.022	0.044	0.030	0.039	0.025
vla4	0.052	0.003	0.051	0.003	0.050	0.003	0.051	0.004	0.052	0.002
wineR	0.620	0.040	0.614	0.041	0.610	0.042	0.615	0.046	0.619	0.049
wineW	0.696	0.014	0.695	0.015	0.696	0.014	0.696	0.015	0.696	0.013

Table 6.4: Number of datasets where the method in the row obtained statistically smaller test RMSE in relation to the method in the column, according to a Wilcoxon test with 95% confidence level.

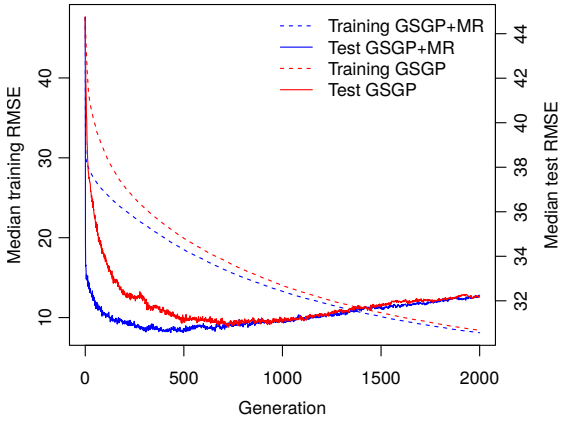
	GSGP	GSGP+A	GSGP+AR	GSGP+M	GSGP+MR	Total (wins)
GSGP	–	1	1	0	0	2
GSGP+A	3	–	1	2	0	6
GSGP+AR	5	1	–	1	4	11
GSGP+M	6	3	5	–	3	17
GSGP+MR	7	4	4	3	–	18
Total (losses)	21	9	11	6	7	

all individuals are greater than or equal to *out* and none is smaller than *out* in the dimension d or vice-versa.

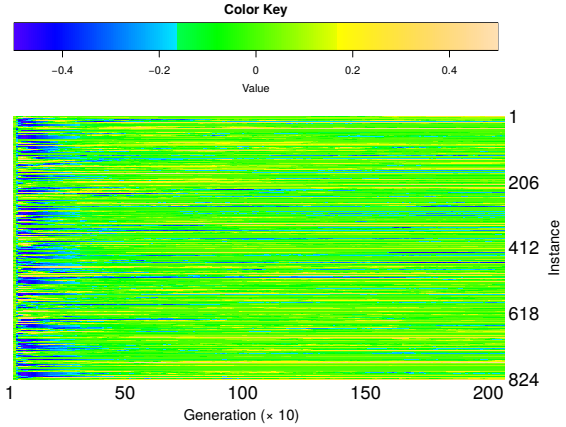
We used this measure to compare GSGP+M, which obtained one of the best results, with GSGP without the GD operator. Here, we show only the datasets where the differences were visually perceptible in our plots—namely bioavailability and concrete. Figures 6.6a and 6.6c show the analysis of the MDD throughout the generations using heatmaps for the bioavailability and concrete datasets, respectively. We reported the mean of the values resulting from ten replications of the first fold adopted in the test bed, using the best parameter combination from Section 6.3.1. The values in each cell correspond to the difference between the MDD obtained by GSGP+M and GSGP—i.e., values close to -0.5 indicate the dimension is better distributed in the GSGP+M, values close to 0.5 indicate the opposite and values close to 0 indicate similar behaviour.



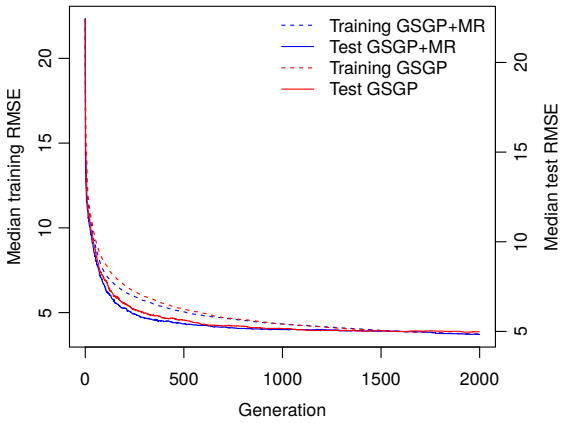
(a) MDD bioavailability heatmap.



(b) Median RMSE (bioavailability).



(c) MDD concrete heatmap.



(d) Median RMSE (concrete).

Figure 6.6: MDD heatmap and evolution of the RMSE over the generations for GSGP+M and GSGP in the datasets bioavailability and concrete.

The x axis corresponds to the number of generations (only values multiple of 10 are considered), and the y axis represents different training cases—i.e., all dimensions of the semantic space.

As expected, GSGP+M population is better distributed than the GSGP population in the initial generations (blue region in the left side). However, this behaviour does not persist along the evolution, given the decaying probability of GD. After 250 generations, approximately, both methods present a similar MDD for the three datasets, denoted by the high concentration of green in the remainder of the heatmap.

Figures 6.6b and 6.6d show the evolution of the fitness of the best individual along the generations in the training and test sets for GSGP and GSGP+M for these same datasets. Note that at the begining of the generations, when the probability of applying the GD operator is higher, the error in the training and test sets drops quicker

Table 6.5: Median elapsed time (in seconds) of the GSGP and GSGP+M. All differences are statistically significant according to a Wilcoxon test with a confidence level of 95%.

Dataset	GSGP+M	GSGP	Difference (%)
air	1708.3	1523.9	12.10
bio	421.0	387.6	8.62
con	1229.7	1195.0	2.90
cpu	327.4	313.4	4.48
eneC	852.9	801.1	6.47
eneH	826.8	782.3	5.70
for	585.5	555.8	5.34
kei5	10579.3	9287.9	13.90
kei6	234.0	224.5	4.23
kei7	1021.2	995.1	2.62
ppb	215.4	203.4	5.92
tow	5357.9	4997.3	7.22
vla1	1847.4	1870.3	-1.23
vla4	5344.2	5074.2	5.32
wineR	1597.2	1447.7	10.33
wineW	4690.9	4339.7	8.09

for GSGP+M than GSGP. However, as the operator becomes less applied, the curves start to approximate both in the training and test sets.

The third point to be analysed is the impact of the operator on the execution time of GSGP. Table 6.5 presents the time elapsed during the execution of GSGP+M and GSGP, considering both training and test stages. Except for vladislavleva-1, the use of GD increases the execution time of the GSGP from 2.62% to 13.9%, depending on the dataset. The trade-off between the improvement on RMSE and the increase on computational time needs to be further analysed, and more efficient versions of the operator can be designed.

6.4 Conclusions

This chapter presented a general framework to construct Geometric Dispersion (GD) operators for GSGP in the context of symbolic regression, followed by two concrete instantiations: the multiplicative geometric dispersion and additive geometric dispersion operators. These operators move the individuals in order to balance the population around the target output in each dimension of the semantic space, with the objective of expanding the convex hull defined by the population to include the desired output vector.

Experimental analysis was performed on a test bed composed by sixteen datasets to compare the effects of GD operators within GSGP: GSGP with additive GD, multiplicative GD and without GD operators were compared regarding the test RMSE. The results showed that GD operators can improve the search performance in terms of test

RMSE and that the multiplicative version presents advantage over the additive version regarding test RMSE. We also analysed the GD impact on the population distribution around the target output. The analysis showed that the operator promoted a better distribution of the population around the desired output in relation to GSGP in the initial generations, where the probability of applying it was higher. Besides that, we examined the impact of the use of GD operators on GSGP execution time. The results indicated that GD operators increase the computational time by a manageable amount varying according to the dataset.

Chapter 7

Reducing the Dimensionality of the Semantic Space

By definition, as presented in Chapter 3, the semantic space $\mathcal{S}$ (implicitly) searched by geometric semantic operators is n-dimensional, where n is the number of training instances we learn the function from. When applying any function—e.g., an individual—to the training set, the produced output corresponds to a point in the semantic space.

Thus, as presented by Issue 4 in the Section 1.1, the number of dimensions of the semantic space equals the number of training examples. In problems where this number is high—a common scenario in real-world applications—the search process can became harder as the number of dimensions increases, a problem well-known as the *curse of dimensionality* [Domingos, 2012]. As the number of dimensions of the problem increases, the volume of the search space also increases exponentially.

One of the simplest ways to deal with the curse of dimensionality is to reduce the number of dimensions of the search space [Domingos, 2012]. As in geometric semantic genetic programming each space dimension corresponds to a training instance, an alternative is to perform what in the machine learning literature is known as data instance selection. Data instance selection is a well-known problem within the context of data classification, but there is not extensive work regarding regression problems [Arnaiz-González et al., 2016]. Instance selection methods are strongly based on distances between training instances from both the set of input and output features.

This chapter evaluates the impact of reducing the number of dimensions of the semantic space in the context of geometric semantic genetic programming. Intuitively, a lower number of dimensions can make search more feasible, reducing the number of evaluations required to find a suitable solution while decreasing the chances of data overfitting.

Looking at how current instance selection methods work, we take advantage of previous knowledge and propose two approaches for instance selection: (i) apply current instance selection methods as a pre-process step before training points are given to GSGP; (ii) incorporate instance selection to the evolution of GSGP. In the first case, we use the methods Threshold Condensed Nearest Neighbor (TCNN) and Threshold Edited Nearest Neighbor (TENN) [Kordos and Blachnik, 2012] to select instances, which are then used as input for the GSGP. The second approach incorporates instance selection to the evolution of GSGP, through the proposed Probabilistic instance Selection based on the Error (PSE) method [Oliveira et al., 2016a].

7.1 Related Work

Instance selection methods are commonly used in the classification literature [Garcia et al., 2012], and play different roles in noisy and noise-free application scenarios. In noise-free scenarios, the idea is to remove points from the training set without degrading accuracy, such as improving storage and search time. In noisy application domains, the main idea is to remove outliers. In classification, these methods rely on the class labels of neighbour instances to determine the rejection/acceptance of an instance to the selected set. However, there are not many methods for instance selection in regression problems. A few works have extended well-known instance selection methods for classification to the context of regression [Arnaiz-González et al., 2016].

Guillen et al. [2010] introduced a method based on mutual information, inspired by feature selection methods that rely on this criterion. The method focuses on noise free scenarios, and has as its main objective to choose the best subset of instances to build a model. In this same direction, Rodríguez-Fdez et al. [2013] proposed the Class Conditional Instance Selection for Regression (CCISR). It extends the Class Conditional Instance Selection method for classification, which uses a class conditional nearest neighbour relation to guide the search process. Kordos and Blachnik [2012] proposed the Threshold Condensed Nearest Neighbour (TCNN) and Threshold Edited Nearest Neighbour (TENN) algorithms—regression versions of the ENN and CNN methods for classification, respectively. These algorithms will be discussed in the next section, as they are used for instance selection in the proposed method.

Recently, Arnaiz-González et al. [2016] compared different strategies for instance selection in regression: discretization techniques—which transform the continuous outputs of the problem into discrete variables and then apply the traditional version of instance selection methods for classification—TCNN and TENN. They also proposed

an ensemble method, namely bagging, to combine several instance selection algorithms. Each algorithm within the ensemble returns an array of binary votes (0 means the instance is not selected and 1 otherwise), and the relevance of an instance in the training set is considered proportional to the number of accumulated votes. The final instance selection is given by a threshold, which defines the percentage of votes an instance must have to be selected. As expected, the ensemble method presented the best results overall.

In our context, the use of an ensemble is not justifiable, as it is a time consuming task and would add too much time overhead to the search. For this reason, we adopted the TCNN and TENN, as the first assumes noise-free scenarios and the second focuses on outliers.

7.2 Strategies for Semantic Space Dimensionality Reduction

As the dimensionality of the semantic space in GSGP is defined by the number of instances given as input to a candidate regression function, by reducing the number of input instances we automatically reduce the number of dimensions of the semantic space, which in turn reduces the complexity of the search space. Intuitively, the smaller the complexity the smaller the number of possible convex combinations, which may help the speed of convergence to the optimum. In this context, the first strategy we propose to reduce the number of dimensions of the search space is executed before data is given as input to GSGP, and depends only on the characteristics of the dataset. The second strategy, in turn, takes into account the median absolute error of an instance during GSGP evolution to select the most appropriate instances.

7.2.1 Pre-Processing Strategies

We first introduce two methods for instance selection in regression. The Threshold Edited Nearest Neighbour (TENN) and Threshold Condensed Nearest Neighbour (TCNN) [Kordos and Blachnik, 2012] adapt instance selection algorithms for classification problems—Edited Nearest Neighbour (ENN) [Wilson, 1972] and Condensed Nearest Neighbour (CNN) [Hart, 1968], respectively—to the regression domain. They are presented in Algorithms 9 and 10.

These algorithms employ an internal regression method to evaluate the instances according to a similarity-based error. The decision of keeping or removing the i-th

Algorithm 9: TENN

Input: $T = \{(\mathbf{x_i}, y_i)\}_{i=1}^{n}$, k, α
Output: Instance set $C \subset T$

```
1  Shuffle T;
2  C ← T;
3  for i ← 1 to n do
4      ŷ ← regression(x_i, C \ (x_i, y_i));
5      N ← knn(k, T);
6      θ ← α · sd(N);
7      if θ = 0 then
8          θ ← α
9      if |y_i − ŷ| > θ then
10         C ← C \ (x_i, y_i)
11 return C;
```

Algorithm 10: TCNN

Input: $T = \{(\mathbf{x_i}, y_i)\}_{i=1}^{n}$, k, α
Output: Instance set $C \subset T$

```
1  Shuffle T;
2  C ← (x_1, y_1);
3  for i ← 2 to n do
4      ŷ ← regression(x_i, C);
5      N ← knn(k, T);
6      θ ← α · sd(N);
7      if θ = 0 then
8          θ ← α
9      if |y_i − ŷ| > θ then
10         C ← C ∪ (x_i, y_i)
11 return C;
```

instance from the training set is based on the deviation of the instance prediction $\hat{y}_i$ and the expected output y_i, given by $|\hat{y}_i - y_i|$. If this difference is smaller than a threshold θ, $\hat{y}_i$ and y_i are considered similar and the instance is accepted or rejected, depending on the algorithm. The threshold θ is computed based on the local properties of the dataset, given by $\alpha \cdot sd(N)$, where α is a parameter controlling the sensitivity and $sd(N)$ returns the standard deviation of the outputs of the set N, composed by the k nearest neighbours of the instance.

The internal regression method adopted by TCNN and TENN—the procedure *regression* presented in Algorithms 9 and 10—can be replaced by any regression method.

Algorithm 11: PSE method

Input: Training set (T), population (P), lower bound (λ)
Output: Instance set $C \subset T$

foreach $inst = (\mathbf{x}_i, y_i) \in T$ **do** `// Compute the median absolute error`
  $E \leftarrow [|p_1(\mathbf{x}_i) - y_i)|, |p_2(\mathbf{x}_i) - y_i)|, \ldots, |p_m(\mathbf{x}_i) - y_i)|]$;
  $inst.med \leftarrow median(E)$;
Sort T by med value in descending order;
$C \leftarrow \{\}$;
for $i \leftarrow 1$ **to** $|T|$ **do**
  $inst \leftarrow (\mathbf{x}_i, y_i) \in T$;
  $\tilde{r} \leftarrow \frac{(i-1)}{|T|-1}$; `// Compute the normalized rank`
  $prob_{sel} \leftarrow 1 - (1 - \lambda) \cdot \tilde{r}^2$; `// Probability of selecting` $inst$
  if $prob_{sel} \geq rand()$ **then** `// Add` $inst$ `to` C `with probability` $prob_{sel}$
    $C \leftarrow C \cup \{inst\}$;
return C;

Our implementation uses a version of the kNN (k-nearest neighbour) algorithm for regression to infer the value of $\hat{y}$. Besides the training set T, these algorithms receive as input the number of neighbours to be considered and a parameter α, which controls how the threshold is calculated. At the end, the set C of instances selected to be used to train the external regression method is returned.

TENN is a decremental method, starting with all training cases in the set C and iteratively removing the instances diverging from their neighbours. An instance $(\mathbf{x}_i, y_i)$ is considered divergent if the output $\hat{y}$ inferred by the model learned without the instance is dissimilar from its output (y_i). TCNN, on the other hand, is an incremental method, beginning with only one instance from the training set in C and iteratively adding only those instances that can improve search. The instance $(\mathbf{x}_i, y_i)$ is added only if the output $\hat{y}$ inferred by the model learned with C diverges from y_i.

7.2.2 GSGP Integrated Strategies

Both TENN and TCNN disregard any information about the external regression algorithm, since they are used in a pre-processing phase. In order to overcome this limitation, we propose a method to select instances based on their median absolute error, considering the output of the programs in the current population. The method, called Probabilistic instance Selection based on the Error (PSE), probabilistically selects a subset from the original training set at each ρ generations, as presented in Algorithm 11. The higher the median absolute error, the higher the probability of an

instance being selected to compose the training subset used by GSGP. The rationale behind this approach is to give higher probability to instances which are, in theory, more difficult to be predicted by the current population evolved by GSGP.

Given a GSGP population $P = \{p_1, p_2, \ldots, p_m\}$, the median absolute error of the i-th instance $(\mathbf{x}_i, y_i) \in T$ is given by the median value of the set $E = \{|p_1(\mathbf{x}_i) - y_i|, |p_2(\mathbf{x}_i) - y_i|, \ldots, |p_m(\mathbf{x}_i) - y_i|\}$. These values are used to sort T in descending order, and the position of the instance in T is used to calculate its probability of being selected to be part of the training set.

In order to compute this probability, the method normalizes the rank of the instance in T to the range $[0, 1]$ by

$$\tilde{r} = \frac{(i-1)}{|T|-1} \ , \tag{7.1}$$

where i is the position of the instance in the ordered set T, $|.|$ denotes the cardinality of the set and $\tilde{r} \in [0, 1]$ is the normalized rank. The value of $\tilde{r}$ is used to calculate the probability of selecting the instance, given by

$$prob_{sel} = 1 - (1 - \lambda) \cdot \tilde{r}^2 \ , \tag{7.2}$$

where λ is a parameter that determines the lower bound of the probability function. The higher the value of λ, the more instances are selected. Figure 7.1 presents the value of $prob_{sel}$ according to $\tilde{r}$, for $\lambda = 0.3$. The resulting number of instances is proportional to the area under the function (shaded area in the figure), equivalent to $\frac{2+\lambda}{3}$.

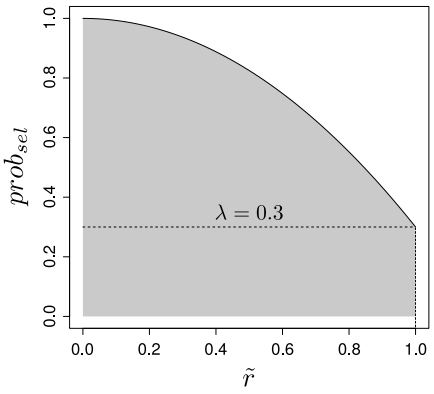


Figure 7.1: Probability of selecting an instance according to its normalized rank, for $\lambda = 0.3$.

7.3 Experimental Results

In this section we present an experimental analysis of the instance selection strategies in a collection of fifteen datasets, presented in Table 7.1, selected from the test bed introduced in Section 4.2.3. The results obtained by TCNN and TENN (Section 7.3.1), and PSE (Section 7.3.2) are compared with GSGP with all instances.

All executions used a population of 1,000 individuals evolved for 2,000 generations with tournament selection of size 10. The grow method [Koza, 1992a] was adopted to generate the random functions inside the geometric semantic operators, and the ramped half-and-half method [Koza, 1992a] used to generate the initial population, both with maximum individual depth equal to 6. The terminal set included the variables of the problem and constant values randomly picked from the interval $[-1, 1]$. The function set included three binary arithmetic operators $(+, -, \times)$ and the analytic quotient (AQ) [Ni et al., 2013]. The GSGP method employed the geometric semantic mutation and crossover—for Manhattan-based fitness function—operators from [Castelli et al., 2015a], both with probabilities equal to 0.5. The mutation step required by the mutation operator was defined as 10% of the standard deviation of the output vector (*out*) given by the training data. All instances in the training set were used as input for the instance selection methods and GSGP.

7.3.1 Comparing Instance Selection Methods

In this section we compare the results obtained by GSGP with and without the instance selection performed before the evolutionary stage (pre-processing). The selection was performed by TCNN (GSGP-TCNN) and TENN (GSGP-TENN) methods, with 10 different values for α equally distributed in the intervals $[0.1, 1]$ and $[5.5, 10]$, respectively, and $k = 9$—the same value adopted by Kordos and Blachnik [2012]. Table 7.1 presents the median training and test RMSE's and the data reduction obtained with α resulting in the largest data reduction by TCNN and TENN methods—1 and 5.5, respectively.

In order to investigate the significance of instance selection methods in GSGP, we compared it with a third strategy, where we randomly selected l instances from each dataset, with no replacement, to compose a new training set used as input by GSGP. The value of l is defined as the smallest size of the sets resulting from TENN and TCNN. Table 7.1 presents the median training and test RMSE's of these experiments in the last two columns (denoted as 'GSGP-Rnd'). The results obtained show that using TCNN and TENN do not make any systematic improvement on GSGP results.

Table 7.1: Median training and test RMSE and reduction (*% red.*) achieved by the algorithms for each dataset. Values highlighted in bold corresponds to test RMSE statically worse than GSGP, according to a Wilcoxon test with 95% confidence.

	GSGP		**GSGP-TCNN**			**GSGP-TENN**			**GSGP-Rnd**	
Dataset	**tr**	**ts**	**tr**	**ts**	**% red.**	**tr**	**ts**	**% red.**	**tr**	**ts**
air	7.89	8.42	7.76	**8.74**	38.60	8.06	**8.60**	1.90	7.65	8.38
bio	9.89	30.74	4.95	**36.29**	46.30	9.84	31.38	0.90	4.55	**34.39**
con	3.65	5.39	2.80	**6.40**	38.20	3.65	5.21	3.20	3.18	**5.95**
cpu	6.13	30.92	5.46	**33.61**	11.20	5.06	**51.54**	65.40	5.67	32.28
eneC	1.26	1.51	1.28	**2.49**	14.70	1.28	**1.83**	36.60	1.19	**1.71**
eneH	0.80	0.96	0.83	**1.87**	11.10	0.67	**1.84**	45.40	0.77	**1.11**
for	30.74	51.63	13.68	**101.90**	42.80	30.75	51.94	5.80	22.49	**57.57**
kei6	0.01	0.40	0.01	0.36	10.60	0.00	**1.25**	53.00	0.01	0.32
kei7	0.02	0.02	0.02	0.02	5.30	0.01	**0.40**	68.50	0.01	**0.05**
ppb	0.92	28.74	0.20	**32.08**	41.50	0.91	28.04	3.80	0.25	**30.50**
tow	20.44	21.92	19.82	**22.71**	12.60	20.44	**43.86**	41.90	20.40	**22.06**
vla1	0.01	0.04	0.01	**0.07**	20.90	0.01	**0.07**	43.40	0.01	**0.06**
wineR	0.49	0.62	0.40	**0.73**	51.10	0.49	0.62	0.10	0.41	**0.66**
wineW	0.64	0.70	0.66	**0.78**	52.30	0.64	0.69	0.10	0.60	**0.71**
yac	2.12	2.52	2.20	**5.19**	36.90	2.11	**2.83**	24.30	2.01	**2.63**

Moreover, the results obtained by them are no better than those generated by a random selection scheme. Hence, the strategies used by these methods do not seem appropriate for the scenario we have.

7.3.2 Evaluating the Effects of PSE

In this section, we first investigate the sensitivity of PSE parameters and then compare the performance of GSGP with and without the PSE method. PSE parameters ρ and λ have a direct impact on the number of instances selected and how they are selected. In order to analyse their impact on the search, we fixed the GSGP parameters and focused on looking at the results as we varied these parameters. The values of ρ were set to 5, 10 and 15 while we varied the value of λ in 0.1, 0.4 and 0.7. Table 7.2 presents the median training RMSE obtained by the GSGP with these PSE configurations.

The experiments with PSE adopt the values of ρ and λ resulting in the smallest median training RMSE, as presented in Table 7.2. Table 7.3 presents the median training and test RMSE's obtained by GSGP and by GSGP with PSE (GSGP-PSE). In order to identify statistically significant differences, we performed Wilcoxon tests with 95% confidence level, regarding the test RMSE of both methods in 50 executions. The symbol ▲(▼) in the last column indicates datasets where the GSGP-PSE performed better (worse) than the GSGP. Overall, GSGP with PSE performs better in terms of test RMSE than GSGP without PSE, being better in five datasets and worse in two.

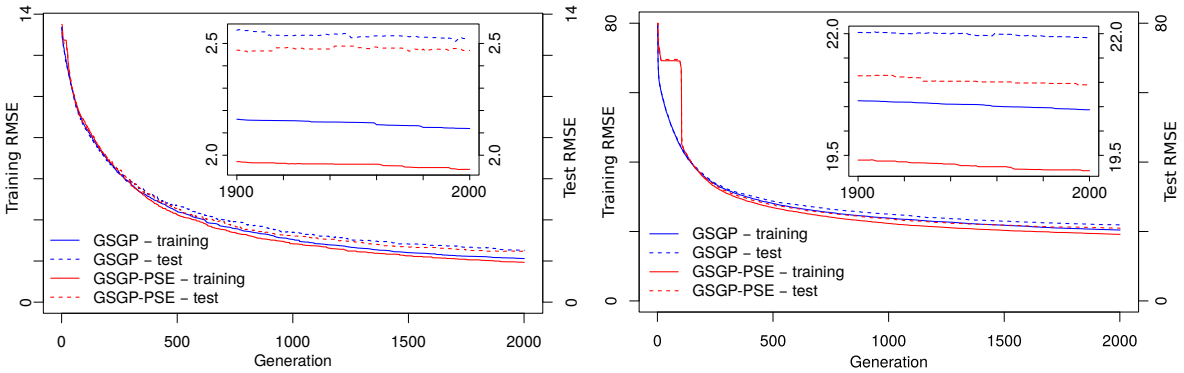
Table 7.2: Median training RMSE of the PSE with different values of λ and ρ for the adopted test bed. The smallest RMSE for each dataset is presented in bold (ties were decided by checking differences in less significant digits, when they existed, or randomly, otherwise).

	$\rho = 5$			$\rho = 10$			$\rho = 15$		
Dataset	$\lambda = 0.1$	$\lambda = 0.4$	$\lambda = 0.7$	$\lambda = 0.1$	$\lambda = 0.4$	$\lambda = 0.7$	$\lambda = 0.1$	$\lambda = 0.4$	$\lambda = 0.7$
air	8.03	8.15	8.11	**7.97**	8.05	8.16	8.12	8.05	8.11
bio	9.66	9.66	9.88	9.70	9.69	9.83	**9.53**	9.77	9.81
con	3.35	3.49	3.56	3.35	3.45	3.58	**3.34**	3.45	3.56
cpu	**4.93**	5.46	5.70	5.02	5.33	5.89	5.01	5.39	5.88
eneC	1.13	1.19	1.23	1.12	1.18	1.22	**1.12**	1.17	1.23
eneH	**0.66**	0.72	0.77	0.67	0.72	0.76	0.67	0.71	0.76
for	25.94	27.67	29.21	25.72	27.61	29.43	**25.55**	27.87	29.58
kei6	0.01	0.01	0.01	0.01	**0.01**	0.01	0.01	0.01	0.01
kei7	0.02	0.02	0.02	0.02	**0.02**	0.02	0.02	0.02	0.02
ppb	**0.50**	0.65	0.81	0.53	0.65	0.80	0.52	0.63	0.76
tow	19.22	19.74	19.98	19.22	19.61	20.09	**19.18**	19.61	19.92
vla1	0.01	0.01	0.01	0.01	0.01	0.01	**0.01**	0.01	0.01
wineR	0.47	0.48	0.49	0.47	0.48	0.49	**0.47**	0.48	0.49
wineW	**0.63**	0.64	0.64	0.63	0.64	0.64	0.63	0.64	0.64
yac	1.94	2.02	2.09	**1.94**	2.01	2.08	1.94	2.00	2.09

Table 7.3: Median training and test RMSE's obtained for each dataset. The symbol ▲(▼) indicates GSGP-PSE is statistically better (worse) than GSGP in the test set according to a Wilcoxon test with 95% confidence.

	GSGP		**PSE**		
Dataset	**tr**	**ts**	**tr**	**ts**	
air	7.88	8.42	7.97	8.55	
bio	9.89	30.74	9.53	32.16	▼
con	3.65	5.39	3.34	5.24	▲
cpu	6.13	30.92	4.93	33.44	
eneC	1.26	1.51	1.12	1.38	▲
eneH	0.80	0.96	0.66	0.84	▲
for	30.74	51.63	25.55	51.32	
kei6	0.01	0.40	0.01	0.32	
kei7	0.02	0.02	0.02	0.02	
ppb	0.92	28.74	0.50	28.96	▼
tow	20.44	21.92	19.18	20.95	▲
vla1	0.01	0.04	0.01	0.05	
wineR	0.49	0.62	0.47	0.62	
wineW	0.64	0.70	0.63	0.69	
yac	2.12	2.52	1.94	2.47	▲

Figure 7.2 compares the evolution of the fitness of the best individual along the generations in the training and test sets for GSGP and GSGP-PSE, for two different datasets. Note that GSGP errors are overall higher than PSE. For instance, looking at the convergence of the dataset *towerData*, if we stop the evolution at generation 1,000, GSGP would have a test error of 25.02 and GSGP-PSE of 23.64. GSGP needs 293 more generations to reach that same error.



(a) *yacht* dataset. (b) *towerData* dataset.

Figure 7.2: Median RMSE in the training and test sets over the generations for GSGP with and without PSE for the *yacth* and *towerData* datasets. The insets show an enlargement of the plots for the last 100 generations.

7.4 Conclusions

This chapter presented a study about the impact of instance selection methods on GSGP search. Two approaches were adopted: (i) selecting the instances in a pre-processing step; and (ii) selecting instances during the evolutionary process, taking into account the impact of the instance on the search.

Experiments were performed in a collection of 15 datasets in order to evaluate the impact of the instance selection. The first analysis showed GSGP fed with the whole dataset performs better in terms of test RMSE than when using subsets selected with TENN, TCNN or randomly. The second analysis indicated that GSGP with PSE can improve the test RMSE in relation to GSGP alone. Thus, the use of instance selection to reduce the semantic space is worth further investigation.

Chapter 8

Conclusions and Future Work

In the previous chapters we presented the issues investigated in this thesis, along with a review of related works studied in the field of semantic GP. We also presented a series of methods, algorithms, experiments and discussions regarding the conjectures presented in Chapter 1.

This chapter presents a summary of the work performed in order to tackle the issues and answer the questions presented in the first chapter, along with the conclusions drawn from the experimental analyses.

8.1 Issue 1: GSGP and the Population's Convex Hull

The region of the semantic space reachable by the geometric semantic crossover operator depends directly on the convex hull described by the population's semantics, such that the target output vector can be found—by means of the crossover operator—only if the output vector is inside the convex hull.

Although the proposed Geometric Dispersion (GD) framework does not ensure that the convex hull contains the desired output vector, it allows the repositioning of the population taking into account the convex hull described by the individuals during the evolution. The experimental analysis presented in Chapter 6 indicates that GD can improve the distribution of the population around the target output, improving the search performed by GSGP.

8.2 Issue 2: Further Analysis of the Geometric Semantic Crossover

Chapter 4 presented a study of the geometric semantic crossover operator. The study investigated the impact of the operator within GSGP in Manhattan metric semantic spaces and proposes two alternative strategies to combine the parents with the operator in Euclidean semantic spaces—one employing convex linear combination and one applying non-convex linear combination, both with coefficients optimally calculated according to the error in the training set.

The experimental analysis conducted in the selected test bed lead us to the conclusion that the geometric semantic crossover has beneficial impact on the search. In addition, the results showed that the proposed operator using non-convex linear combination deteriorates the performance measured by the test RMSE and the one employing convex linear combination is not better than the original operator for Euclidean semantic spaces. Also, the results indicated that the geometric semantic crossover operator for fitness function based on Manhattan distance is the best alternative among the geometric semantic crossover operators analysed during the experiments.

8.3 Issue 3: Sequential Symbolic Regression

Chapter 5 presented the Sequential Symbolic Regression (SSR), a first attempt to maintain the size of the solutions in a treatable level while applying the Geometric Semantic Crossover operator for Euclidean semantic space (GSXE). However, the method is not geometric semantic, since it employs the GSXE to "concatenate" functions generated by a canonical GP method. SSR starts with an empty solution that is iteratively constructed by combining new functions through the geometric semantic crossover operator. These functions are induced from the error relative to the difference between the solution found so far and the expected output.

We presented an empirical study in a selected test bed and compared SSR results to a canonical GP, to GSGP and to the Genetic Recursive Symbolic Regression (GRSR), regarding performance, measured by the median test RMSE and size of the resulting solutions. The results showed that GRSR has no significant difference in relation to the SSR regarding the size of the resulting solutions or performance. SSR performs significantly better than GP, but generates solutions statistically bigger, indicating that the SSR is a good alternative to GP when the performance is more important than the size of the functions. Regarding the GSGP, SSR has equivalent

performance and generates solutions statically smaller, indicating that SRR is a good alternative to GSGP, generating solutions with reduced size.

8.4 Issue 4: Instance Selection and GSGP

The size of the space searched by GP-based methods has direct impact on the performance of the search. The wider the space being explored, the harder is to find the optimum (or optima), since the number of possible candidate solutions increases with the space. Although GSGP does not search directly the semantic space, the modifications on the individuals' syntax performed by means of geometric semantic operators have direct interpretation in the semantic space.

Chapter 7 conjectures that, by reducing the dimensionality of the semantic space, the search performed by GSGP becomes easier. In order to attain such reduction, we proposed the use of subsets of instances selected from the training set instead of the whole set—since the dimensionality of the semantic space is equal to the size of the training set. Two strategies were investigated and analysed in a test bed: the use of instance selection methods for regression before the evolution begins; and a new method, called Probabilistic instance Selection based on the Error (PSE), which select instances during the evolution based on their median error regarding the current population. The results indicated that the use of instance selection methods before the evolution has no positive impact on the search. On the other hand, the experimental analysis showed that PSE can improve the search, indicating that reduction of the semantic space dimensionality can improve the search, but relies on the approach adopted to achieve the reduction.

8.5 Future Work

This thesis investigated different issues regarding geometric semantic genetic programming. Although we now can answer some questions—or, at least, we have good evidences about our statements—about this new branch of GP, there are numerous possibilities for future work.

8.5.1 Optimized Non-Convex Geometric Semantic Crossover

There are some directions regarding the optimized non-convex Euclidean-based geometric semantic crossover operator presented in Chapter 4, including the addition of

an intercept term in the Equation 4.4—which can improve the fitting—and employing regularization [Bishop, 2006], in order to control overfitting.

8.5.2 Sequential Symbolic Regression

Potential future SRR developments involve investigating other methods for combing functions and to dynamically control their impact (weight) on the final solution; applying a simple regression method—e.g. linear regression—as an initial approximation; utilising a random or induced function in replacement to the random constant present in the geometric semantic crossover operator; employing a stop criterion—regarding the SSR iterations—based on the method convergence; and adjusting the whole model using a strategy based on the backfitting algorithm [Breiman and Friedman, 1985].

8.5.3 Geometric Dispersion Operators

The Geometric Dispersion (GD) framework is an initial approach to better distribute the population around the desired output vector in the semantic space.

There are different new strategies to explore, including investigating the use of other operations—besides multiplication and addition—to move individuals with the GD operator, and the development of more sophisticated implementations that might reduce differences of computational time when it is compared to GSGP. Furthermore, an analysis of the impact of including information about the distribution of the population in other stages of the evolution, such as in the selection phase, are worth investigating.

8.5.4 Simplification within GSGP

Given the observation of Vanneschi et al. [2014b] that GSGP never modifies programs once they have been created, it is valid to hypothesize that the exponential growth of the individuals due to GSGP operators can be drastically reduced by simplifying the initial population and the random trees generated by the method. There are basically two strategies to simplify arithmetic expressions: by algebraic [Mori et al., 2007, 2009; Kinzett et al., 2008; Song et al., 2009; Johnston et al., 2010] or by numerical simplification [Koza, 1992a; Hooper and Flann, 1996; Ekárt, 2000; Zhang and Wong, 2008].

In the algebraic approach, algebra rules are used to manipulate the expression represented by the tree and reduce it to a simpler equivalent form. In the numerical approach, the vector of outputs produced by the tree when applied to the training cases

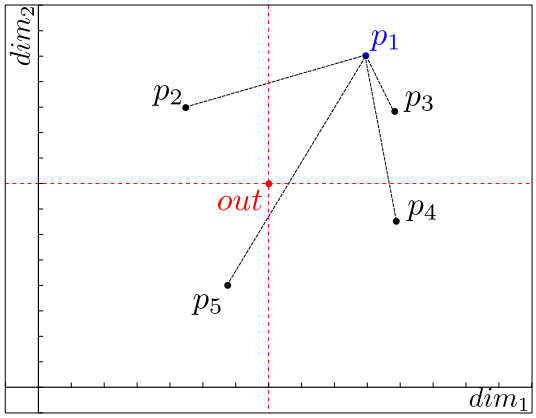


Figure 8.1: Selecting the best individual to mate with p_1.

(its semantics) is examined to determine if particular subtrees can be approximated by simpler structures or removed. Both strategies have advantages and disadvantages that can be tested to verify the impact on the performance, on the size of the final individuals and on the computational effort.

8.5.5 Parent Selection for Geometric Semantic Crossover

Given that the offspring generated by the geometric semantic crossover is located between its parents in the semantic space, we hypothesize that the resulting offspring can be improved if it is generated from parents with the largest number of dimensions where the desired output vector (out) is between them. Let p_1 be a first parent, previously selected by a canonical selection method, and $P \setminus p_1$ be the set of individuals from the population, excluding p_1. The proposed method selects the second parent to maximize the expression

$$bestParent(p_1) = \arg\max_{p \in (P \setminus p_1)} \sum_{i=1}^{n} betweeness(p_1, p, i) \ , \tag{8.1}$$

where n is the size of the training set and $betweeness(p_a, p_b, i)$ is a function that returns 1 if $out[i]$ is in the interval bounded by $s(p_a)[i]$ and $s(p_b)[i]$ and returns 0 otherwise. Figure 8.1 presents the principle behind the selection method. Let p_1 be the first parent selected and $P \setminus p_1 = \{p_2, p_3, p_4, p_5\}$, the desired output vector (out) is located between p_1, p_2 and p_1, p_4 regarding dim_1 and dim_2, respectively, between p_1, p_5 regarding both dimensions and no dimension regarding p_1, p_3.

However, this type of selection can bias the search, by removing the randomness present in other selection methods. An experimental analysis can be conducted to investigate the effects of the proposed selection method on the search.

8.5.6 Promoting Semantic Diversity

Chapter 3 introduced different non-geometric semantic methods that promote semantic diversity in the population level [Beadle, 2009; Beadle and Johnson, 2008, 2009a,b; Castelli et al., 2013c; Jackson, 2010a,b] and geometric semantic initialization methods that generate individuals semantically diverse—namely Competent Initialization (CI) [Pawlak, 2015] and Semantic Geometric Initialization (SGI) [Pawlak and Krawiec, 2016] methods. However, neither the non-geometric nor geometric semantic methods quantify the diversity in the individual level—i.e., how diverse is one individual from another.

A possible direction to explore semantic diversity involves proposing a metric to quantify the individual dissimilarity in relation to the other members of the population. Distance functions—defined by the metric adopted in the semantic space—are straightforward solutions. However, other approaches can be explored as well.

After defining the metric, a next step could consider new methods to promote semantic diversity. A possible approach consists in employing a semantic library—similar to the one used by the Locally Geometric Semantic Crossover (LGX) [Krawiec and Pawlak, 2012a, 2013b] and Approximately Geometric Crossover (AGX) [Krawiec and Pawlak, 2013a; Pawlak et al., 2014]—to generate individuals that obey a given diversity metric threshold.

Bibliography

Albinati, J., Pappa, G. L., Otero, F. E. B., and Oliveira, L. O. V. B. (2014). A study of semantic geometric crossover operators in regression problems. Semantic Methods in Genetic Programming. Workshop at Parallel Problem Solving from Nature 2014 conference.

Albinati, J., Pappa, G. L., Otero, F. E. B., and Oliveira, L. O. V. B. (2015). The effect of distinct geometric semantic crossover operators in regression problems. In Machado, P., Heywood, M. I., McDermott, J., Castelli, M., García-Sánchez, P., Burelli, P., Risi, S., and Sim, K., editors, *18th European Conference, EuroGP 2015*, volume 9025 of *Lecture Notes in Computer Science*, pages 3–15. Springer International Publishing. **Nominated for Best Paper Award at EuroGP 2015**.

Angeline, P. J. (1996). An investigation into the sensitivity of genetic programming to the frequency of leaf selection during subtree crossover. In Koza, J. R., Goldberg, D. E., Fogel, D. B., and Riolo, R. L., editors, *Proceedings of the 1st Annual Conference on Genetic Programming*, pages 21--29, Stanford University, CA, USA. MIT Press.

Angeline, P. J. (1997). Subtree crossover: Building block engine or macromutation. *Genetic Programming*, 97:9–17.

Angeline, P. J. and Pollack, J. B. (1992). The evolutionary induction of subroutines. In *Proceedings of the fourteenth annual conference of the cognitive science society*, pages 236–241.

Angeline, P. J. and Pollack, J. B. (1994). Coevolving high-level representations. In Langton, C., editor, *Artificial Life III*, pages 55--71, Reading, MA. Addison-Wesley.

Arnaiz-González, Á., Blachnik, M., Kordos, M., and García-Osorio, C. (2016). Fusion of instance selection methods in regression tasks. *Information Fusion*, 30:69--79.

Bache, K. and Lichman, M. (2014). UCI machine learning repository. `http://archive.ics.uci.edu/ml`.

Banerjee, S. and Roy, A. (2014). *Linear Algebra and Matrix Analysis for Statistics*. Texts in Statistical Science. Chapman and Hall/CRC.

Banzhaf, W., Nordin, P., Keller, R., and Francone, F. (1998). *Genetic Programming: An Introduction*. The Morgan Kaufmann Series in Artificial Intelligence Series. Morgan Kaufmann Publishers Inc.

Basilevsky, A. (2005). *Applied Matrix Algebra in the Statistical Sciences*. Dover Books on Mathematics. Dover Publications. ISBN 9780486153377.

Beadle, L. (2009). *Semantic and Structural Analysis of Genetic Programming*. PhD thesis, Computing, University of Kent, CT2 7NF.

Beadle, L. and Johnson, C. G. (2008). Semantically driven crossover in genetic programming. In *Evolutionary Computation, 2008. IEEE Congress on*, pages 111--116. IEEE.

Beadle, L. and Johnson, C. G. (2009a). Semantic analysis of program initialisation in genetic programming. *Genetic Programming and Evolvable Machines*, 10(3):307?337. ISSN 1573-7632.

Beadle, L. and Johnson, C. G. (2009b). Semantically driven mutation in genetic programming. In *Evolutionary Computation, 2009. IEEE Congress on*, pages 1336--1342.

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning (Information Science and Statistics), 1st edn. 2006. corr. 2nd printing edn*. Information science and statistics. Springer.

Blickle, T. (2000). Tournament selection. In Bäck, T., Fogel, D. B., and Michalewicz, Z., editors, *Evolutionary computation 1: Basic algorithms and operators*, volume 1, chapter 24, pages 181–186. CRC Press.

Botzheim, J., Cabrita, C., Kóczy, L. T., and Ruano, A. E. (2007). Genetic and bacterial programming for b-spline neural networks design. *Journal of Advanced Computational Intelligence*, 11(2):220–231.

Brameier, M. and Banzhaf, W. (2007). *Linear Genetic Programming*. Genetic and Evolutionary Computation. Springer US.

Breiman, L. (2001). Using iterated bagging to debias regressions. *Machine Learning*, 45(3):261--277.

Breiman, L. and Friedman, J. H. (1985). Estimating optimal transformations for multiple regression and correlation. *Journal of the American statistical Association*, 80(391):580--598.

Bryant, R. E. (1986). Graph-based algorithms for boolean function manipulation. *Computers, IEEE Transactions on*, 100(8):677--691.

Castelli, M., Manzoni, L., Silva, S., and Vanneschi, L. (2014a). Self-tuning geometric semantic GP. Semantic Methods in Genetic Programming. Workshop at Parallel Problem Solving from Nature 2014 conference.

Castelli, M., Silva, S., Vanneschi, L., Cabral, A., Vasconcelos, M. J., Catarino, L., and Carreiras, J. M. (2013a). Land cover/land use multiclass classification using GP with geometric semantic operators. In Esparcia-Alcázar, A. I., editor, *16th European Conference, EvoApplications 2013*, volume 7835 of *LNCS*. Springer Berlin Heidelberg.

Castelli, M., Silva, S., Vanneschi, L., Castelli, M., Silva, S., and Vanneschi, L. (2015a). A c++ framework for geometric semantic genetic programming. *Genetic Programming and Evolvable Machines*, 16(1):73–81. ISSN 1389-2576.

Castelli, M., Trujillo, L., and Vanneschi, L. (2015b). Energy consumption forecasting using semantic-based genetic programming with local search optimizer. *Computational Intelligence and Neuroscience*, 2015:8.

Castelli, M., Trujillo, L., Vanneschi, L., and Popovič, A. (2015c). Prediction of energy performance of residential buildings: A genetic programming approach. *Energy and Buildings*, 102:67 – 74. ISSN 0378-7788.

Castelli, M., Vanneschi, L., , and Popovič, A. (2015d). Predicting burned areas of forest fires: An artificial intelligence approach. *Fire Ecology*, 11(1):106–118.

Castelli, M., Vanneschi, L., and Felice, M. D. (2015e). Forecasting short-term electricity consumption using a semantics-based genetic programming framework: The south italy case. *Energy Economics*, 47:37 – 41. ISSN 0140-9883.

Castelli, M., Vanneschi, L., and Silva, S. (2013b). Prediction of high performance concrete strength using genetic programming with geometric semantic genetic operators. *Expert Systems with Applications*, 40(17):6856--6862.

Castelli, M., Vanneschi, L., and Silva, S. (2013c). Semantic search-based genetic programming and the effect of intron deletion. *Cybernetics, IEEE Transactions on*, 44(1):103--113.

Castelli, M., Vanneschi, L., and Silva, S. (2014b). Prediction of the unified parkinson's disease rating scale assessment using a genetic programming system with geometric semantic genetic operators. *Expert Systems with Applications*, 41(10):4608 – 4616.

Castelli, M., Vanneschi, L., Silva, S., and Ruberto, S. (2015f). How to exploit alignment in the error space: Two different gp models. In Riolo, R., Worzel, W. P., and Kotanchek, M., editors, *Genetic Programming Theory and Practice XII*, Genetic and Evolutionary Computation, pages 133–148. Springer International Publishing.

Deb, K. (2001). *Multi-objective optimization using evolutionary algorithms*. John Wiley & Sons.

Demšar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *The Journal of Machine Learning Research*, 7:1--30.

D'haeseleer, P. (1994). Context preserving crossover in genetic programming. In *Proceedings of the First IEEE Conference on Evolutionary Computation, 1994*, volume 1, pages 256--261. IEEE.

Dick, G. (2015). Improving geometric semantic genetic programming with safe tree initialisation. In Machado, P., Heywood, M. I., McDermott, J., Castelli, M., García-Sánchez, P., Burelli, P., Risi, S., and Sim, K., editors, *18th European Conference, EuroGP 2015*, volume 9025 of *LNCS*, pages 28–40. Springer International Publishing. ISSN 1611-3349.

Domingos, P. (2012). A few useful things to know about machine learning. *Commun. ACM*, 55(10):78--87. ISSN 0001-0782.

Eiben, A. E. and Smith, J. E. (2003). *Introduction to evolutionary computing*. Natural Computing Series. Springer-Verlag Berlin Heidelberg.

Ekárt, A. (2000). Shorter fitness preserving genetic programs. In Fonlupt, C., Hao, J.-K., Lutton, E., Schoenauer, M., and Ronald, E., editors, *4th European Conference, AE'99*, volume 1829 of *LNCS*, pages 73--83. Springer Berlin Heidelberg.

Friedman, J. H. (2001). Greedy function approximation: a gradient boosting machine. *Annals of Statistics*, pages 1189--1232.

Friedman, J. H. and Friedman, J. H. (2002). Stochastic gradient boosting. *Computational Statistics & Data Analysis*, 38(4):367 – 378. ISSN 0167-9473.

Futuyma, D. (2013). *Evolution*. Stony Brook University, third edition.

Galván-López, E., McDermott, J., O'Neill, M., and Brabazon, A. (2010). Towards an understanding of locality in genetic programming. In *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, pages 901--908, New York, NY, USA. ACM.

Garcia, S., Derrac, J., Cano, J. R., and Herrera, F. (2012). Prototype selection for nearest neighbor classification: Taxonomy and empirical study. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(3):417–435.

García, S., Fernández, A., Luengo, J., and Herrera, F. (2010). Advanced nonparametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining: Experimental analysis of power. *Information Sciences*, 180(10):2044--2064.

Gentle, J. E. (1998). *Numerical Linear Algebra for Applications in Statistics*. Statistics and Computing. Springer New York.

Gonçalves, I., Silva, S., and Fonseca, C. M. F. (2015). On the generalization ability of geometric semantic genetic programming. In Machado, P., Heywood, M. I., McDermott, J., Castelli, M., García-Sánchez, P., Burelli, P., Risi, S., and Sim, K., editors, *18th European Conference, EuroGP 2015*, volume 9025 of *LNCS*, pages 41–52. Springer International Publishing. ISSN 1611-3349.

Guillen, A., Herrera, L. J., Rubio, G., Pomares, H., Lendasse, A., and Rojas, I. (2010). New method for instance or prototype selection using mutual information in time series prediction. *Neurocomputing*, 73(10):2030--2038.

Hart, P. (1968). The condensed nearest neighbor rule (corresp.). *Information Theory, IEEE Transactions on*, 14(3):515--516.

Hooper, D. C. and Flann, N. S. (1996). Improving the accuracy and robustness of genetic programming through expression simplification. In *Proceedings of the 1st annual conference on genetic programming*, pages 428--428. MIT Press.

Iman, R. L. and Davenport, J. M. (1980). Approximations of the critical region of the friedman statistic. *Communications in Statistics-Theory and Methods*, 9(6):571--595.

Jackson, D. (2010a). Phenotypic diversity in initial genetic programming populations. In Esparcia-Alcazar, A. et al., editors, *13th European Conference, EuroGP 2010*, volume 6021 of *LNCS*, pages 98--109. Springer.

Jackson, D. (2010b). Promoting phenotypic diversity in genetic programming. In *Parallel Problem Solving from Nature, PPSN XI, part II*, volume 6239 of *LNCS*, pages 472--481. Springer Berlin Heidelberg.

Johnson, C. G. (2007). Genetic programming with fitness based on model checking. In Ebner, M., O'Neill, M., Ekárt, A., Vanneschi, L., and Esparcia-Alcázar, A. I., editors, *10th European Conference, EuroGP 2007*, volume 4445 of *LNCS*, pages 114--124. Springer.

Johnston, M., Liddle, T., and Zhang, M. (2010). A relaxed approach to simplification in genetic programming. In Esparcia-Alcázar, A. I., Ekárt, A., Silva, S., Dignum, S., and Şima Uyar, A., editors, *13th European Conference, EuroGP*, volume 6021 of *LNCS*, pages 110--121. Springer.

Keijzer, M. (2003). Improving symbolic regression with interval arithmetic and linear scaling. In Ryan, C., Soule, T., Keijzer, M., Tsang, E., Poli, R., and Costa, E., editors, *6th European Conference, EuroGP 2003*, volume 2610, pages 70–82. Springer Berlin Heidelberg.

Kinzett, D., Zhang, M., and Johnston, M. (2008). Using numerical simplification to control bloat in genetic programming. In Li, X., Kirley, M., Zhang, M., Green, D., Ciesielski, V., Abbass, H., Michalewicz, Z., Hendtlass, T., Deb, K., Tan, K. C., Branke, J., and Shi, Y., editors, *7th International Conference, SEAL 2008*, volume 5361 of *LNCS*, pages 493--502, Melbourne, Australia. Springer Berlin Heidelberg.

Kordos, M. and Blachnik, M. (2012). Instance selection with neural networks for regression problems. In Villa, A. E. P. et al., editors, *Proc. of the ICANN'12, part II*, pages 263--270. Springer Berlin Heidelberg.

Koza, J. R. (1992a). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, volume 1. MIT Press, Cambridge, MA, USA.

Koza, J. R. (1992b). Hierarchical automatic function definition in genetic programming. In Whitley, L. D., editor, *Foundations of Genetic Algorithms 2*, pages 297--318, Vail, Colorado, USA. Morgan Kaufmann.

Koza, J. R. (1994). *Genetic Programming II: Automatic Discovery of Reusable Programs*. MIT Press, Cambridge Massachusetts.

Krawiec, K. (2011). Learnable embeddings of program spaces. In *Genetic Programming: Proceedings of the 14th European Conference*, pages 166--177. Springer.

Krawiec, K. and Lichocki, P. (2009). Approximating geometric crossover in semantic space. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, GECCO '09, pages 987--994, New York, NY, USA. ACM.

Krawiec, K. and Pawlak, T. (2012a). Locally geometric semantic crossover. In *GECCO Companion '12*, pages 1487--1488. ACM.

Krawiec, K. and Pawlak, T. (2012b). Quantitative analysis of locally geometric semantic crossover. In Coello, C. A. C., Cutello, V., Deb, K., Forrest, S., Nicosia, G., and Pavone, M., editors, *PPSN XII. Part I*, volume 7491 of *LNCS*, pages 397--406. Springer.

Krawiec, K. and Pawlak, T. (2013a). Approximating geometric crossover by semantic backpropagation. In *Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation*, GECCO '13, pages 941--948, New York, NY, USA. ACM.

Krawiec, K. and Pawlak, T. (2013b). Locally geometric semantic crossover: a study on the roles of semantics and homology in recombination operators. *Genetic Programming and Evolvable Machines*, 14(1):31--63. ISSN 1573-7632.

Langdon, W. (2000). Size fair and homologous tree crossovers for tree genetic programming. *Genetic Programming and Evolvable Machines*, 1(1-2):95–119. ISSN 1389-2576.

Langdon, W. and Poli, R. (2002). *Foundations of Genetic Programming*. Springer-Verlag Berlin Heidelberg.

Lay, D. C. (2012). *Linear Algebra and Its Applications*. Pearson, 4th edition.

Lee, G. Y. (1999). Genetic recursive regression for modeling and forecasting real-world chaotic time series. *Advances in genetic programming*, 3:401.

Liu, Y. and Khoshgoftaar, T. (2004). Reducing overfitting in genetic programming models for software quality classification. In *Ninth IEEE International Symposium on High-Assurance Systems Engineering (HASE'05)*, pages 56--65. IEEE Computer Society.

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

Luke, S. and Spector, L. (1998). A revised comparison of crossover and mutation in genetic programming. In Koza, J. R., Banzhaf, W., Chellapilla, K., Deb, K., Dorigo, M., Fogel, D. B., Garzon, M. H., Goldberg, D. E., Iba, H., and Riolo, R. L., editors, *Proceedings of the Third Annual Genetic Programming Conference*, pages 208--214, San Francisco, CA. Morgan Kaufmann.

Majeed, H. and Ryan, C. (2006). A less destructive, context-aware crossover operator for GP. In Collet, P., Tomassini, M., Ebner, M., Gustafson, S., and Ekárt, A., editors, *Genetic Programming: Proceedings of the 9th European Conference on Genetic Programming*, volume 3905 of *Lecture Notes in Computer Science*, pages 36–48. Springer Berlin Heidelberg.

McDermott, J., White, D. R., Luke, S., Manzoni, L., Castelli, M., Vanneschi, L., Jaskowski, W., Krawiec, K., Harper, R., De Jong, K., and O'Reilly, U.-M. (2012). Genetic programming needs better benchmarks. In *Proceedings of the 14th Annual Conference on Genetic and Evolutionary Computation*, pages 791--798. ACM.

Mendes-Moreira, J., Soares, C., Jorge, A. M., and Sousa, J. F. D. (2012). Ensemble approaches for regression: A survey. *ACM Computing Surveys (CSUR)*, 45(1):10.

Miller, J. F. (1999). An empirical study of the efficiency of learning boolean functions using a cartesian genetic programming approach. In Banzhaf, W., Daida, J., Eiben, A. E., Garzon, M. H., Honavar, V., Jakiela, M., and Smith, R. E., editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 2, pages 1135--1142, Orlando, Florida, USA. Morgan Kaufmann.

Mitchell, T. M. (1997). *Machine Learning*. McGraw Hill.

Moraglio, A. (2007). *Towards a Geometric Unification of Evolutionary Algorithms*. PhD thesis, Department of Computer Science, University of Essex, UK.

Moraglio, A. (2011). Abstract convex evolutionary search. In *Proceedings of the 11th workshop on Foundations of genetic algorithms*, FOGA '11, pages 151--162. ACM.

Moraglio, A. (2014). An efficient implementation of GSGP using higher-order functions and memoization. In *Semantic Methods in Genetic Programming. Workshop at Parallel Problem Solving from Nature 2014 conference*.

Moraglio, A., Krawiec, K., and Johnson, C. G. (2012). Geometric semantic genetic programming. In *Parallel Problem Solving from Nature XII*, pages 21–31. Springer.

Moraglio, A. and Mambrini, A. (2013). Runtime analysis of mutation-based geometric semantic genetic programming for basis functions regression. In *Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation*, GECCO '13, pages 989--996, New York, NY, USA. ACM.

Moraglio, A. and Poli, R. (2004). Topological interpretation of crossover. In Deb, K., editor, *Proceedings of the Genetic and Evolutionary Computation Conference - GECCO 2004, Part I*, volume 3102 of *Lecture Notes in Computer Science*, pages 1377--1388. Springer Berlin Heidelberg.

Mori, N., McKay, B., Nguyen, X. H., and Essam, D. (2007). Equivalent decision simplification: A new method for simplifying algebraic expressions in genetic programming. In *Proceedings of 11th Asia-Pacific Workshop on Intelligent and Evolutionary Systems*, Yokosuka, Japan.

Mori, N., McKay, B., Xuan, N., Daryl, E., and Takeuchi, S. (2009). A new method for simplifying algebraic expressions in genetic programming called equivalent decision simplification. In Omatu, S., Rocha, M. P., Bravo, J., Fernández, F., Corchado, E., Bustillo, A., and Corchado, J. M., editors, *10th International Work-Conference on Artificial Neural Networks, IWANN 2009. Part II*, volume 5518 of *LNCS*, pages 171--178. Springer Berlin Heidelberg.

Murphy, K. P. (2012). *Machine learning: a probabilistic perspective*. MIT press.

Nguyen, Q. U. (2011). *Examining Semantic Diversity and Semantic Locality of Operators in Genetic Programming*. PhD thesis, University College Dublin, Ireland.

Nguyen, Q. U., Hoai, N. X., and O'Neill, M. (2009a). Semantics based mutation in genetic programming: The case for real-valued symbolic regression. In *15th international conference on soft computing, Mendel'09*, volume 9, pages 73--91.

Nguyen, Q. U., Hoai, N. X., O'Neill, M., McKay, B., and Galván-López, E. (2009b). An analysis of semantic aware crossover. In Cai, Z., Li, Z., Kang, Z., and Liu, Y., editors, *Computational Intelligence and Intelligent Systems*, volume 51 of *Communications in Computer and Information Science*, pages 56--65. Springer.

Nguyen, Q. U., Hoai, N. X., O'Neill, M., McKay, R. I., and Galván-López, E. (2011). Semantically-based crossover in genetic programming: application to real-valued symbolic regression. *Genetic Programming and Evolvable Machines*, 12(2):91--119.

Nguyen, Q. U., Hoai, N. X., O'Neill, M., McKay, R. I., and Phong, D. N. (2013). On the roles of semantic locality of crossover in genetic programming. *Information Sciences*, 235:195--213.

Nguyen, Q. U., Nguyen, X. H., and O'Neill, M. (2009c). Semantic aware crossover for genetic programming: the case for real-valued function regression. In Vanneschi, L., Gustafson, S., Moraglio, A., Falco, I. D., and Ebner, M., editors, *Proceedings of the 12th European Conference on Genetic Programming, EuroGP '09*, volume 5481 of *LNCS*, pages 292--302. Springer Berlin Heidelberg.

Nguyen, Q. U., O'Neill, M., Hoai, N. X., Mckay, B., and Galván-López, E. (2010). Semantic similarity based crossover in GP: The case for real-valued function regression. In Collet, P., Monmarché, N., Legrand, P., Schoenauer, M., and Lutton, E., editors, *9th International Conference, Evolution Artificielle, EA, 2009*, volume 5975 of *LNCS*, pages 170--181. Springer Berlin Heidelberg.

Ni, J., Drieberg, R. H., and Rockett, P. I. (2013). The use of an analytic quotient operator in genetic programming. *Evolutionary Computation, IEEE Transactions on*, 17(1):146--152.

Oliveira, L. O. V. B., Miranda, L. F., Pappa, G. L., Otero, F. E. B., and Takahashi, R. H. C. (2016a). Reducing dimensionality to improve search in semantic genetic programming. In Handl, J., Hart, E., Lewis, P. R., López-Ibánez, M., Ochoa, G., and Paechter, B., editors, *Proc. of the 14th International Conference on Parallel Problem Solving from Nature (PPSN XIV)*, volume 9921 of *LNCS*, pages 375--385. Springer International Publishing.

Oliveira, L. O. V. B., Otero, F. E. B., Miranda, L. F., and Pappa, G. L. (2016b). Revisiting the sequential symbolic regression genetic programming. In *Proc. of the 5th Brazilian Conference on Intelligent System (BRACIS)*. (to appear).

Oliveira, L. O. V. B., Otero, F. E. B., and Pappa, G. L. (2016c). A dispersion operator for geometric semantic genetic programming. In *Proc. of the the 2016 Genetic and Evolutionary Computation Conference*, GECCO '16, pages 773--780. ACM. **Received the Best Paper Award in the track "Genetic Programming"**.

Oliveira, L. O. V. B., Otero, F. E. B., and Pappa, G. L. (2016d). A generic framework for building dispersion operators in the semantic space. In *Genetic Programming Theory and Practice XIV*. Springer International Publishing. (to appear).

Oliveira, L. O. V. B., Otero, F. E. B., Pappa, G. L., and Albinati, J. (2015). Sequential symbolic regression with genetic programming. In Riolo, R., Worzel, B., and Kotanchek, M., editors, *Genetic Programming Theory and Practice XII*, Genetic and Evolutionary Computation, pages 73–90. Springer International Publishing.

Panyaworayan, W. and Wuetschner, G. (2002). Time series prediction using a recursive algorithm of a combination of genetic programming and constant optimization. *Facta universitatis-series: Electronics and Energetics*, 15(2):265--279.

Pawlak, T., Wieloch, B., Krawiec, K., Pawlak, T., Wieloch, B., and Krawiec, K. (2014). Semantic backpropagation for designing search operators in genetic programming. *Evolutionary Computation, IEEE Transactions on*, 19(3):326–340.

Pawlak, T. P. (2014). Combining semantically-effective and geometric crossover operators for genetic programming. In Bartz-Beielstein, T., Branke, J., Filipič, B., and Smith, J., editors, *Parallel Problem Solving from Nature–PPSN XIII*, volume 8672 of *Lecture Notes in Computer Science*, pages 454--464. Springer International Publishing.

Pawlak, T. P. (2015). *Competent Algorithms for Geometric Semantic Genetic Programming*. PhD thesis, Poznan University of Technology, Pozna'n, Poland.

Pawlak, T. P. and Krawiec, K. (2016). Semantic geometric initialization. In Heywood, I. M., McDermott, J., Castelli, M., Costa, E., and Sim, K., editors, *Proceedings of the 19th European Conference on Genetic Programming, EuroGP '16*, volume 9594 of *LNCS*, pages 261--277, Cham. Springer International Publishing.

Pawlak, T. P., Wieloch, B., and Krawiec, K. (2015). Review and comparative analysis of geometric semantic crossovers. *Genetic Programming and Evolvable Machines*, 16(3):351–386. ISSN 1389-2576.

Poli, R. (1999). Parallel distributed genetic programming. In Corne, D., Dorigo, M., and Glover, F., editors, *New Ideas in Optimization*, Advanced Topics in Computer Science, chapter 27, pages 403--431. McGraw-Hill, Maidenhead, Berkshire, England.

Poli, R., Langdon, W. B., and McPhee, N. F. (2008). *A Field Guide to Genetic Programming*. Lulu Enterprises, UK Ltd.

R Core Team (2015). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria.

Rodríguez-Fdez, I., Mucientes, M., and Bugarín, A. (2013). An instance selection algorithm for regression and its application in variance reduction. In *Fuzzy Systems (FUZZ), 2013 IEEE International Conference on*, pages 1–8. ISSN 1098-7584.

Ruberto, S., Vanneschi, L., Castelli, M., and Silva, S. (2014). ESAGP - a semantic GP framework based on alignment in the error space. In Nicolau, M., Krawiec, K., Heywood, M. I., Castelli, M., García-Sánchez, P., Merelo, J. J., Rivas Santos, V. M., and Sim, K., editors, *17th European Conference, EuroGP 2014*, volume 8599 of *LNCS*, pages 150–161. Springer Berlin Heidelberg.

Sebesta, R. W. (2012). *Concepts of Programming Languages*. Pearson, 10th edition.

Song, A., Chen, D., and Zhang, M. (2009). Bloat control in genetic programming by evaluating contribution of nodes. In *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation*, GECCO '09, pages 1893--1894, New York, NY, USA. ACM.

Spector, L., Harrington, K., and Helmuth, T. (2012). Tag-based modularity in tree-based genetic programming. In *GECCO '12: Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference*, pages 815--822, Philadelphia, Pennsylvania, USA. ACM.

Spector, L., Harrington, K., Martin, B., and Helmuth, T. (2011a). What's in an evolved name? The evolution of modularity via tag-based reference. In *Genetic Programming Theory and Practice IX*, Genetic and Evolutionary Computation, chapter 1, pages 1--16. Springer, Ann Arbor, USA.

Spector, L., Martin, B., Harrington, K., and Helmuth, T. (2011b). Tag-based modules in genetic programming. In *GECCO '11: Proceedings of the 13th annual conference on Genetic and evolutionary computation*, pages 1419--1426, Dublin, Ireland. ACM.

Stewart, J. (2008). *Calculus: Early Transcendentals*. Thomson Brooks/Cole, 6th edition.

Sun, W. and Yuan, Y.-X. (2006). *Optimization Theory and Methods: Nonlinear Programming*, volume 1 of *Springer Optimization and Its Applications*. Springer US.

Vanneschi, L., Castelli, M., Costa, E., Re, A., Vaz, H., Lobo, V., and Urbano, P. (2015). Improving maritime awareness with semantic genetic programming and linear scaling: Prediction of vessels position based on ais data. In Mora, A. M. and Squillero, G., editors, *18th European Conference, EvoApplications 2015.*, volume 9028 of *LNCS*, pages 732–744. Springer International Publishing.

Vanneschi, L., Castelli, M., Manzoni, L., and Silva, S. (2013). A new implementation of geometric semantic gp and its application to problems in pharmacokinetics. In Krawiec, K., Moraglio, A., Hu, T., Etaner-Uyar, A. c., and Hu, B., editors, *16th European Conference, EuroGP 2013*, volume 7831 of *LNCS*, pages 205--216. Springer Berlin Heidelberg.

Vanneschi, L., Castelli, M., and Silva, S. (2014a). A survey of semantic methods in genetic programming. *Genetic Programming and Evolvable Machines*, pages 1–20. ISSN 1389-2576.

Vanneschi, L., Silva, S., Castelli, M., and Manzoni, L. (2014b). Geometric semantic genetic programming for real life applications. In Riolo, R., Moore, J. H., and Kotanchek, M., editors, *Genetic Programming Theory and Practice XI*, Genetic and Evolutionary Computation, pages 191–209. Springer New York.

Walker, J. A. and Miller, J. F. (2008). The automatic acquisition, evolution and reuse of modules in cartesian genetic programming. *IEEE Trans. Evolutionary Computation*, 12(4):397--417.

White, D. R., McDermott, J., Castelli, M., Manzoni, L., Goldman, B. W., Kronberger, G., Jaśkowski, W., O'Reilly, U.-M., and Luke, S. (2013). Better GP benchmarks: community survey results and proposals. *Genetic Programming and Evolvable Machines*, 14(1):3–29. ISSN 1389-2576.

Wilson, D. L. (1972). Asymptotic properties of nearest neighbor rules using edited data. *Systems, Man and Cybernetics, IEEE Transactions on*, SMC-2(3):408--421.

Zhang, M. and Wong, P. (2008). Genetic programming for medical classification: a program simplification approach. *Genetic Programming and Evolvable Machines*, 9(3):229--255.

Zhu, Z., Nandi, A. K., and Aslam, M. W. (2013). Adapted geometric semantic genetic programming for diabetes and breast cancer classification. In *2013 IEEE International Workshop on Machine Learning for Signal Processing (MLSP)*, pages 1–5. IEEE.

Appendix A

Extended Results

In this Appendix we present the results of the statistical tests summarized in the previous chapters of this Thesis.

Table A.1 and A.2 present the results of the Wilcoxon tests performed in Section 4.2.3 in order to compare the different variations of the geometric semantic crossover operator introduced in Chapter 4.

Table A.1: Results of the Wilcoxon test for the experiments summarized in Table 4.5, regarding the training set, considering a confidence level of 95%. The symbol ▲(▼) indicates the configuration in the first row is statistically better (worse) than the configuration being tested against.

Dataset	**GSXE against**			**GSXM against**		**GSX-C against**
	GSXM	**GSX-C**	**GSX-L**	**GSX-C**	**GSX-L**	**GSX-L**
air	▼	▼	▼		▼	▼
bio	▼	▼	▼	▲	▼	▼
con	▼	▼	▼	▲		▼
cpu	▼	▼	▼	▲	▼	▼
eneC	▼		▼	▲	▼	▼
eneH	▼		▼	▲	▼	▼
for	▼	▼	▼	▲	▼	▼
kei5	▼	▼	▼	▲	▼	▼
kei6	▼		▼	▲	▼	▼
kei7	▼	▼	▼		▼	▼
kor1	▼	▼	▼	▲	▼	▼
kor2	▼	▼	▼	▲	▼	▼
kor12	▼	▼	▼	▲	▲	▼
ppb	▼	▼	▼	▲	▼	▼
tow	▼	▼	▼	▲	▼	▼
vla1	▼	▲	▼	▲	▼	▼
vla4	▼	▼	▲	▲	▲	▲
wineR	▼	▼	▼	▲	▼	▼
wineW	▼	▼	▼	▲	▼	▼
yac	▼	▼	▼		▼	▼
Total (wins)	0	1	1	17	2	1
Total (losses)	20	16	19	0	17	19

Table A.2: Results of the Wilcoxon test for the experiments summarized in Table 4.5, regarding the test set, considering a confidence level of 95%. The symbol ▲(▼) indicates the configuration in the first row is statistically better (worse) than the configuration being tested against.

Dataset	GSXE against			GSXM against		GSX-C against
	GSXM	GSX-C	GSX-L	GSX-C	GSX-L	GSX-L
air	▼	▼	▼		▼	▼
bio	▼		▲	▲	▲	▲
con		▼	▲		▲	▲
cpu	▲		▲	▼	▲	▲
eneC	▼		▼	▲	▼	▼
eneH	▼		▼		▼	▼
for		▲	▲		▲	▲
kei5	▼	▼	▼	▲	▼	▼
kei6	▼		▼	▲	▼	▼
kei7	▼		▲		▲	▲
kor1		▼	▼	▼	▼	▼
kor2		▲	▲	▲	▲	▲
kor12	▲	▲	▲	▲	▲	▲
ppb			▲		▲	▲
tow	▼	▼		▲		
vla1	▼		▲	▲	▲	▲
vla4	▼		▲	▲	▲	▲
wineR	▼		▲	▲	▲	▲
wineW	▼	▲	▲	▲	▲	▲
yac	▼	▼	▼		▼	▼
Total (wins)	2	4	12	11	12	12
Total (losses)	13	6	7	2	7	7

Table A.3 presents the results of the Wilcoxon tests performed in Section 6.3.2 in order to compare the effect on the RMSE of applying different variations of the geometric dispersion operators.

Table A.3: Results of the Wilcoxon test for the experiments summarized in Table 6.4, considering a confidence level of 95%. The symbol ▲(▼) indicates the configuration in the first row is statistically better (worse) than the configuration being tested against. We abbreviated GSGP+A, GSGP+AR, GSGP+M and GSGP+MR to +A, +AR, +M and +MR, respectively.

Dataset	**GSGP against**				**+A against**			**+AR against**		**+M against**
	+A	**+AR**	**+M**	**+MR**	**+AR**	**+M**	**+MR**	**+M**	**+MR**	**+MR**
air	▼	▼	▼	▼				▼		
bio	▼	▼	▼	▼		▲		▲	▲	
con		▼	▼	▼		▼	▼	▼	▼	
cpu								▼		
eneC							▼			
eneH										▼
for			▼							▲
kei5	▲	▲	▼	▼		▼	▼	▼	▼	▼
kei6	▼			▼						
kei7				▼			▼		▼	▼
ppb									▲	
tow			▼			▼		▼		▲
vla1				▼					▼	
vla4		▼			▼				▲	▲
wineR		▼							▲	
wineW					▲	▲				
Total (wins)	1	1	0	0	1	2	0	1	4	3
Total (losses)	3	5	6	7	1	3	4	5	4	3