Development of a hybrid genetic programming technique for computationally expensive optimisation problems

Umberto Armani

Submitted in accordance with the requirements for the degree of Doctor of Philosophy

The University of Leeds
School of Civil Engineering

February 2014

Intellectual Property and Publication Statements

The candidate confirms that the work submitted is his own, except where work which has formed part of jointly-authored publications has been included. The contribution of the candidate and the other authors to this work has been explicitly indicated below. The candidate confirms that appropriate credit has been given within the thesis where reference has been made to the work of others.

In the following the chapters that are based on the work that has formed part of jointly-authored publications are listed. The contribution of the candidate and the co-authors to the work reported in the publication is described after the title of the publication.

Chapter 5 ("Hybrid genetic programming"):

1. U. Armani, V. V. Toropov, A. Polynkin, O. M. Querin, L. F. Alvarez, "Enhancements to a hybrid genetic programming technique applied to symbolic regression", Proceedings of the 8th ASMO UK/ISSMO conference on engineering, design optimization, product and process improvement, London, UK, 2010.

   The whole genetic programming implementation was developed by the candidate, as well as all the Matlab software needed to perform the statistical tests. Models were also produced by the candidate using the software written by him. The co-authors provided fundamental help in planning the design of experiments and in the selection of the test functions on which symbolic regression experiments were performed.

2. U. Armani, S. Coggon, V. V. Toropov, "Derivation of deterministic design data from stochastic analysis in the aircraft design process", Proceedings of the 11th International Conference on Computational Structures Technology (CST 2012), Dubrovnik, Croatia, 2012.

The code used to produce PCE metamodels has been developed by Airbus, EADS and Frazer-Nash consultancy. The analysis of the PCE metamodels and the comparison with genetic programming metamodels (produced using the code described in publication 1) was carried out by the candidate.

Chapter 6 ("HyGP applications to industrial problems"):

3. U. Armani, D. J. Boon, V. V. Toropov, A. Polynkin, L. J. Clarke, M. B. Stowe "Generation of models related to aluminium surface treatment using genetic programming", Proceedings of the 9th world congress on structural and multidisciplinary optimization (WCSMO9), Shizuoka, Japan, 2011.

   The experimental models for the chemical processes were produced by the candidate using the genetic programming code developed by him (first described in publication 1). D. J. Boon, L. J. Clarke and M. B. Stowe provided the experimental data used for the model generation. V. V. Toropov and A. Polynkin provided important help for the optimal tuning of the genetic programming code and in the final revision of the paper.

4. U. Armani, Z. Khatir, A. Khan, V. V. Toropov, A. Polynkin, H. Thompson, N. Kapur "Control of physical consistency in metamodel building by genetic programming", Proceedings of the second international conference on soft computing technology in civil, structural and environmental engineering (CSC 2011), Chania, Greece, 2011.

   The novel approach described in the paper was implemented and added to the genetic programming code described in publication 1 by the candidate. The original idea the approach originated from can be ascribed to the candidate and to co-author V. V. Toropov. Co-author A. Polynkin helped tune the genetic programming tool. Co-authors Z. Khatir, A. Khan provided the experimental data used to produce the models and they also helped select the models for subsequent optimisation. Co-authors H. Thompson and N. Kapur important help in the final revision of the paper is also acknowledged.

5. U. Armani, V. V. Toropov, A. Polynkin, S. Shahpar "Application of explicit metamodels generated by genetic programming to computationally expensive optimisation problems" to be submitted for publication in Structural and Multidisciplinary Optimization journal (Editor-in-Chief: George I.N. Rozvany, ISSN: 1615-147X (print version), ISSN: 1615-1488 (electronic version)), Springer

   Kotanchek function and NASA rotor 37 metamodels were generated by the candidate using the genetic programming tool developed by himself (see publication 1). A. Polynkin performed the blade optimisation with the models produced by genetic programming, using the computing facilities and the software (SOPHY, PADRAM and SOFT) made available by S. Shahpar. V.V. Toropov provided essential help in genetic programming tool tuning and during the final revision of the paper.

6. H. Lohse-Busch, C. Hühne, D. Liu, V. V. Toropov and U. Armani "Parametric optimization of a lattice aircraft fuselage barrel using metamodels built with genetic programming" in B.H.V. Topping and P. Iványi, (Editors), Proceedings of the Fourteenth International Conference on Civil, Structural and Environmental Engineering Computing (CC2013)), Civil-Comp Press, Stirlingshire, United Kingdom, paper 230, 2013. doi:10.4203/ccp.102.230

   The HyGP genetic programming implementation was used by the author to generate the structural responses used by the coauthors to find through GA search the optimal fuselage structure described in the paper. The results were used by the coauthors to identify the main issues in anisogrid barrel fuselage design and to develop a new design.

Ai miei genitori,
e a Laura

"Ve l'offro dunque da semplice dilettante qual sono,
sicuro di non ingannarvi,
avendo provati e riprovati più volte questi piatti da me medesimo;
se poi voi non riuscirete alla prima,
non vi sgomentate;
buona volontà ed insistenza vuol essere,
e vi garantisco che giungerete a farli bene
e potrete anche migliorarli
imperocchè io non presumo di aver toccato l'apice della perfezione."

"Non vorrei però che per essermi occupato di culinaria
mi gabellaste per un ghiottone e per un gran pappatore:
protesto, se mai, contro questa taccia poco onorevole,
perchè non sono nè l'una, nè l'altra cosa.
Amo il bello ed il buono ovunque si trovino
e mi ripudia di vedere straziata,
come suol dirsi,
la grazia di Dio. Amen."

Pellegrino Artusi, *La scienza in Cucina*

("I thus present it to you now as the simple amateur that I am,
certain that I shall not disappoint you,
having tried and retried these dishes many times myself.
If at first you do not succeed,
do not despair;
with good will and persistence,
you shall manage to make them one day, I guarantee it,
and perhaps even to improve them.
For I, after all, cannot presume to have reached the acme of perfection."

"Finally, I should not like my interest in gastronomy
to give me the reputation of a gourmand or a glutton.
I object to any such dishonorable imputation,
for I am neither.
I love the good and the beautiful wherever I find them,
and hate to see anyone squander,
as they say,
God's bounty. Amen.")

Translation excerpted from Pellegrino Artusi, Murtha Baca, Luigi Ballerini, "Science in the Kitchen and the Art of Eating Well", Lorenzo Da Ponte Italian Library, 1997.

Acknowledgements

I would like to extend my sincere gratitude to Prof. Vassili Toropov, Prof. Philip Gaskell, Dr. Andrey Polynkin and Dr. Osvaldo Querin for the guidance they have provided throughout my studies. I have really appreciated being part of a great research team.

I also thank Dr. J. Eves for providing test data for the jet pump optimisation; I thank Dr. Z. Khatir, Dr. A. Khan, Dr. H. Thompson and Dr. N. Kapur for providing data and support for the hospital ward and oven optimisation problems, and for reviewing the paper describing such research activity; I am grateful to Mr. D. J. Boon, Dr. L. J. Clarke, Dr. M. B. Stowe for the data and the precious information regarding aircraft paints, which led to a conference paper. I also thank Mr. H. Lohse-Busch, Dr. C. Hühne and Dr. D. Liu for providing the numerical and experimental data for a few tests of my HyGP code on a real life structural optimisation problem, experiments that led to the submission of a few conference papers. I am really grateful to Mr. J. Lloyd from the University of Cambridge for the help he gave me on Gaussian Processes.

I would also like to thank Dr. Andy Sleigh, Mr. John Hodrien and Mrs. Julie Webster for the considerable help they have given me in developing the genetic programming code described in this thesis, in particular in setting up the parallel version.

Special thanks go to Dr. Stefano Tursi and Mr. Simon Coggon in Airbus UK and Dr. Shahrokh Shahpar in Rolls-Royce UK, who believed in my research activity and offered me a chance to apply its outcomes to industrial problems, providing test data sets and allowing me to use Airbus and Rolls-Royce computing facilities (really needed and appreciated for the NASA rotor 37 optimisation).

I am extremely grateful to all the people who day by day helped me with my studies. I thank Dr. Marco Berci for the offer he made me five years ago, Dr. Maurizio Esposito for the interesting discussions on neodarwinism and philosophy, Dr. Alessandro Condotta for the help in taming computing machines and the many friends and colleagues in the School of Civil Engineering of University of Leeds and in Building 9A/B of the Airbus plant in Filton (Bristol) for providing a nice, vibrant and fun work environment.

Finally, I want to thank my examiners Prof. Keith Worden from the University of Sheffield and Dr. Yong Sheng from the University of Leeds for the rigorous assessment of the work presented in this thesis.

Abstract

The increasing computational power of modern computers has contributed to the advance of nature-inspired algorithms in the fields of optimisation and metamodelling. Genetic programming (GP) is a genetically-inspired technique that can be used for metamodelling purposes. GP main strength is in the ability to infer the mathematical structure of the best model fitting a given data set, relying exclusively on input data and on a set of mathematical functions given by the user. Model inference is based on an iterative or evolutionary process, which returns the model as a symbolic expression (text expression). As a result, model evaluation is inexpensive and the generated expressions can be easily deployed to other users.

Despite genetic programming has been used in many different branches of engineering, its diffusion on industrial scale is still limited. The aims of this thesis are to investigate the intrinsic limitations of genetic programming, to provide a comprehensive review of how researchers have tackled genetic programming main weaknesses and to improve genetic programming ability to extract accurate models from data. In particular, research has followed three main directions. The first has been the development of regularisation techniques to improve the generalisation ability of a model of a given mathematical structure, based on the use of a specific tuning algorithm in case sinusoidal functions are among the functions the model is composed of. The second has been the analysis of the influence that prior knowledge regarding the function to approximate may have on genetic programming inference process. The study has led to the introduction of a strategy that allows to use prior knowledge to improve model accuracy. Thirdly, the mathematical structure of the models returned by genetic programming has been systematically analysed and has led to the conclusion that the linear combination is the structure that is mostly returned by genetic programming runs. A strategy has been formulated to reduce the evolutionary advantage of linear combinations and to protect more complex classes of individuals throughout the evolution.

The possibility to use genetic programming in industrial optimisation problems has also been assessed with the help of a new genetic programming implementation developed during the research activity. Such implementation is an open source project and is freely downloadable from `http://www.personal.leeds.ac.uk/~cnua/mypage.html`.

Contents

List of Tables

List of Figures

Abbreviations

Acronym	**W**hat (it) **S**tands **F**or
ANN	**A**rtificial **N**eural **N**etwork
DoE	**D**esign **o**f **E**xperiments
EA	**E**volutionary **A**lgorithm
EP	**E**volutionary **P**rogramming
ES	**E**volutionary **S**trategy
GA	**G**enetic **A**lgorithm
GP	**G**enetic **P**rogramming
IQR	**I**nter**Q**uartile **R**ange
LHS	**L**atin **H**ypercube **S**ampling
LSE	**L**east **S**quares **E**rror
MAM	**M**ultipoint **A**pproximation **M**ethod
MARS	**M**ultivariate **A**daptive **R**egression **S**plines
MLSM	**M**oving **L**east **S**quares **M**ethod
PCE	**P**olynomial **C**haos **E**xpansion
RBF	**R**adial **B**asis **F**unction
RMSE	**R**oot **M**ean **S**quare **E**rror
RSM	**R**esponse **S**urface **M**ethodology
SQP	**S**equential **Q**uadratic **P**rogramming

Chapter 1

Introduction

1.1 Definition of a model and of a metamodel

Mathematical models permeate all fields of science. They are used on a daily basis in every sector of industry to understand the behaviour of different kinds of systems, which may be either physical, as in engineering, may describe social and economical phenomena, as in finance, or may relate to more abstract entities, as in mathematics and computer science.

A model is defined in mathematical terms as the relationship $\mathbf{f}$ between the selected input or predictor variables $\mathbf{x}$ and the measured responses $\mathbf{y}$ of a system:

$$\mathbf{y} = \mathbf{f}(\mathbf{x}) \qquad | \qquad \mathbf{f} : D \subset \mathbb{R}^N \rightarrow \mathbb{R}^M \tag{1.1}$$

where $\mathbf{x} = \{x_i\}$ for $i = 1, \dots, N$ and $\mathbf{y} = \{y_j\}$ for $j = 1, \dots, M$. The dimensionality of the model is defined as the number of input variables N. For simplicity, in the following the system response will be considered univariate, so $f : D \subset \mathbb{R}^N \rightarrow \mathbb{R}$ and $y \in \mathbb{R}$. The generality of the analysis is not compromised by this assumption as long as the mathematical representation of the model and not its exploitation in statistical terms is concerned. In fact, the probability density function $p(\mathbf{y})$ associated to the output $\mathbf{y}$ can be obtained through the multiplication of the single multiple inputs - single output probability density functions $p(y_i)$ only if the components y_i are statistically independent (Bishop 1995).

Traditional analytical models (also called *fundamental* models (Vladislavleva 2008)) are built from fundamental or constitutive laws, which are the basic deterministic rules describing the immutable behaviour of the constituents of the system under analysis. For example, models of the speed and acceleration of a ball rolling down an inclined plane can be built from Newton's laws. Models of this kind are however subject to a range of limitations that undermines their application to complex phenomena. First of all, the validity of a model is limited by the hypotheses that were assumed during its development: these may be either not realistic or simplistic in a real-life scenario (see for example the importance of extracting from experimental data the inverse dynamics model of an antropomorphic robot arm to correct the physics-based rigid-body-dynamics model as detailed in Rasmussen and Williams (2006, p. 23)). Secondly, phenomena may be too complex or simply their knowledge too scarce to allow for an analytical model to be built (Pierce et al. 2008).

Modern computers have allowed for the development of alternative strategies to tackle a level of complexity that would be otherwise unapproachable using analytical methods. Numerical simulations, or more in general "computer codes" (Simpson et al. 2001), are based on algorithms that systematically apply fundamental or constitutive laws to describe the behaviour and the interactions among discrete parts composing the system under study. Although numerical simulations are still based on general mathematical and physical assumptions, as analytical models are, they do not return an explicit mathematical expression describing the relationship between the system input variables or parameters and the system response or output.

Most of today's engineering analysis relies on the execution of computer simulations (for example in structural analysis or computational fluid dynamics). This approach suffers however from two main drawbacks. First of all simulations do not provide a general insight into the system under analysis, but just the output corresponding to given inputs; secondly, it is difficult to understand the relative influence of each system input variable on the system output. Furthermore, computer simulations are generally computationally expensive (Simpson et al. 2001, Ramu et al. 2010), so their direct use for the exploration of a system response under different input configurations (for example in forecasting, extrapolation and optimisaton) is not recommended (Jin et al. 2001, Harewood et al. 2007).

The need for direct simulations to explore the behaviour of a complex system can be limited by the use of metamodelling techniques. *Metamodelling* or *regression* (Chetwynd et al. 2006) is the process of building an approximation $\tilde{f}$ of the relationship f between the inputs and the outputs of a system (Eq. (1.1)) from a limited number of samples or records $\{\mathbf{x}_k, f(\mathbf{x}_k)\}$ generated by real experiments or computer simulations:

$$y = \tilde{f}(\mathbf{x}) + \epsilon \tag{1.2}$$

where ϵ takes into account the inherent error of the approximation process, the error due to quantities that were not controlled nor observed (Friedman 1991) and measurement or numerical errors (noise).

The approximated relationship $\tilde{f}$ is called *metamodel*, *surrogate* model or *empirical* model of the true underlying input-output relationship f (Eq. (1.1)), to highlight the fact that it is built uniquely from data. According to Vladislavleva (2008, p. 5), metamodels are "explicit models" that "relate black-box inputs and outputs", although this definition is generally loosened to include also models that do not have an explicit representation (Simpson et al. 2001, Jin et al. 2001, Toropov et al. 2005). Regardless their implicit or explicit nature, metamodels provide a means to approximate the response of a system that is far less expensive than a direct numerical simulation or physical experiment (Kroo 2004). As a result, the use of metamodels is nowadays an established way to reduce the computational cost of industrial products analysis and development (Simpson et al. 2001, Ong et al. 2003, Bonte et al. 2005, Toropov et al. 2005, Harewood et al. 2007, Shahpar et al. 2008, Syberfeldt et al. 2009).

1.1.1 Requirements for metamodels and metamodelling techniques

The importance for industry of accurate metamodels can be ascribed to two basic reasons. Metamodels, expecially when they are in explicit form, provide a synthetical representation to data (Kordon and Lue 2004) and increase understanding of the relationship between a system input variables and output, possibly providing physical insight into the the system under analysis (Friedman 1991, Lew et al. 2006, Winkler et al. 2007). The availability of an empirical model also eases the identification of the most influential variables on the system output (sensitivity analysis[1]) (Vladislavleva 2008, Ramu et al. 2010). However, as briefly introduced in the previous section, the main advantage granted by metamodels is engineering analysis time and cost reduction. Metamodels for example are today commonly used for optimisation and uncertainty analysis in lieu of computationally expensive simulations (Toropov et al. 2005, Lew et al. 2006, Harewood et al.

[1]for example metamodels generated by Polynomial Chaos Expansion allow for the analytical extraction of sensitivity information on model input variables (Sudret 2008, Eldred et al. 2008, Arwade et al. 2010).

2007, Zeguer and Bates 2011). Their quick and inexpensive evaluation makes the use of population-based search algorithms affordable, fostering research into this class of meta-heuristic methods (Lamberti and Pappalettere 2011). Metamodels are also suited for more traditional, gradient-based optimisation techniques (Quarteroni et al. 2000, Ramu et al. 2010) for their ability to smooth experimental or computational noise (Loweth et al. 2011).

Given the importance of metamodels, a wide range of metamodelling techniques has been developed and is today available to designers and engineers. Although accuracy and robustness are the main parameters driving the selection of a suitable technique, additional criteria are considered by the final user. Friedman (1991) and Jin et al. (2001) referred to additional performance criteria that are taken into account when selecting a metamodelling technique:

1. *smoothness*: ability to produce metamodels that are continuous and have continuous derivatives;

2. *efficiency*: computational cost for building and using a metamodel (evaluation cost);

3. *transparency*: defined by Jin et al. (2001) as “the capability of providing the information concerning contributions of different variables and interactions among variables”. Friedman (1991) refers to “interpretability” in this regard;

4. *simplicity*: simple techniques are characterised by easy adaptation to each problem and by a reduced number of parameters that have to be set by the user.

The previous additional criteria may affect the choice for a particular technique as much as accuracy and robustness. Efficiency is critical for the success of a metamodelling technique. In general the evaluation of a metamodel is far quicker than the execution of a direct simulation. However, higher metamodel evaluation time necessarily implies longer analysis. The costs incurred to generate a metamodel should also be considered. Not only do the time and computational power to process the building data set have to be taken into account, also the cost of generating (or gathering) the data has to be considered. In this regard, the *curse of dimensionality* (Bellman 1961) is an unavoidable issue whenever high dimensional spaces are dealt with (Ong et al. 2003, Kroo 2004, Smits et al. 2005, Singh et al. 2007, Eldred et al. 2008, Vladislavleva 2008, Arwade et al. 2010): due to the expansion of the design space, to reach the same data density the samples size has to be increased exponentially with the number of dimensions (Friedman 1991, Smits et al.

2005). As collecting or extracting a sufficient number of records may be expensive, metamodelling techniques that minimise the amount of input data are generally preferred.

Transparency and interpretability can give an insight into the behaviour of the system under study, revealing connections with fundamental models. They also ease the analysis and exploitation of a metamodel, its deployment and its use. The availabilty of an explicit metamodel, for example in the form of a text expression, simplify metamodel exploitation in industrial computer codes.

Finally, simplicity should not be considered a minor issue. Metamodelling techniques are complex mathematical and statistical tools: as such, their use could be compromised should they require from the user a detailed understanding of their mathematical subtleties. A quotation from Simpson et al. (2001, p. 135) clearly identifies the problem: "the results of the statistical analysis [...] were difficult for people responsible for the day-to-day operation to interpret and use".

1.1.2 Motivation, research aims and objectives

The paramount role of metamodels and metamodelling techniques in design and optimisation motivates the research effort to improve the existing methodologies and to find new ones.

In the last decades technology advances have encouraged the development of a new class of nature-inspired algorithms that can be applied to metamodelling. Genetic Programming (GP) is one of them: inspired by Darwin's and Wallace's theory of natural evolution and by the later discovery of DNA recombination mechanisms, it can be used to generate metamodels from a set of data samples produced by experiments or computer simulations. The main strength of genetic programming is its ability to both find the optimal mathematical structure and the optimal values of the coefficients of the metamodels. Secondly, the metamodels generated by GP have features that are particularly appreciated in engineering: they are explicit, that is they are symbolic mathematical expressions, and they are global, as the validity of the generated expression extends to the whole design space.

These two properties explain the interest that this technique has attracted in the last two decades and motivate the research activity presented in this work, aimed at exploring the potentialities of genetic programming for metamodelling purposes. In particular, the main research aims are:

1. assessing what the current genetic programming limitations are in terms accuracy, robustness, efficiency, transparency and simplicity and comparing them to existing metamodelling techniques strengths and weaknesses;
2. exploring new strategies for improving the genetic programming algorithm, in particular the integration of deterministic search algorithms in the artificial evolution paradigm;
3. exploring possible approaches to let genetic programming benefit from any available prior knowledge regarding the behaviour of the underlying relationship between the inputs and the outputs of the system under analysis;
4. assessing and encouraging the use of genetic programming in industry and academia as a valuable modelling tool. The final part of the thesis is dedicated to genetic programming application to industrial modelling and optimisation problems.

The research activity presented in this work has focused both on the theoretical aspects of genetic programming as well as on the practical issues concerning its implementation. The main outcomes and contributions set for the research activity here presented are:

1. the development of an open source, parallelised C++ GP code (called HyGP) which can be run on laptop machines as well as on high-performance clusters by engineers and designers;
2. the identification and implementation of a statistical methodology to compare different GP implementations, necessary step to assess possible improvements to GP paradigm;
3. the generation of valid metamodels for real engineering use, as a proof of the validity of the tecnique developed

1.1.3 Structure of the work

This chapter provides a general description of the metamodelling process. The most common global metamodelling techniques and a few methodologies used to generate the building data sets (design of experiments or DoE) are reviewed. A few mid-range and local approximation techniques are also described. Genetic programming is introduced as an explicit and global metamodelling technique and a few industrial and academic problems where it has been applied are shown.

Chapter 2 is a general introduction to evolutionary algorithms, class which genetic programming belongs to. The differences among evolutionary strategies, evolutionary programming, genetic algorithms and genetic programming are discussed and common theoretical principles described.

In Chapter 3 the basic GP paradigm, as described by Koza (1992), is dissected and analysed in each single part, from population initialisation to selection, genetic operations and fitness evaluation. Details of the alternative GP representations to the one used by Koza (1992) are also given.

In Chapter 4 the main pitfalls of the basic GP algorithm are reviewed. The key properties of closure and sufficiency are described. The issue of code growth or *bloat* is explained and a review of different strategies to tackle it are described. A few common approaches to reduce fitness evaluation cost are presented. Advanced GP implementations featuring multiple genotype-phenotype mapping and multiobjective fitness functions are also described.

Chapter 5, 6 and 7 are dedicated to the description of the GP implementation developed during the research activity. Chapter 5 opens with a general description of memetic or hybrid GP algorithms, a class of GP implementations in which deterministic algorithms are used to tune the numerical coefficients of the GP individuals. Then the new GP implementation, called HyGP, is presented and a few enhancements to its basic implementation presented and validated on a few test regression problems. An optimisation problems solved with HyGP and a comparison with other metamodelling techniques (PCE and MLSM) are also described. The application of the developed GP implementation to a few industrial optimisation problems is described in Chapter 6. Chapter 7 reports a novel strategy to bias the HyGP search towards more compactness and simpler metamodels. A comparison between HyGP and gaussian processes performance on a few test regression problems is also decribed.

General conclusions are given in Chapter 8, followed by recommendations for future work. Three appendices are also included to encourage the further use of HyGP. Appendix A details the structure of HyGP code and describes the implementations for sequential and parallel execution. Some suggestions to further parallelise the code are also given. Appendix B is a guide on how to use HyGP, whereas Appendix C contains the HyGP input settings used to generate most of the metamodels described in this dissertation.

1.1.4 Overview of related papers

This thesis includes the material presented in the following papers:

- V. V. Toropov, A. Polynkin, U. Armani, L. F. Alvarez, "Application of metamodel building by genetic programming to industrial optimization problems", Proceedings of the IV european conference on computational mechanics (ECCM 2010), Paris, France, 2010;
- U. Armani, V. V. Toropov, A. Polynkin, O. M. Querin, L. F. Alvarez, "Enhancements to a hybrid genetic programming technique applied to symbolic regression", Proceedings of the 8th ASMO UK/ISSMO conference on engineering, design optimization, product and process improvement, London, UK, 2010;
- U. Armani, D. J. Boon, V. V. Toropov, A. Polynkin, L. J. Clarke, M. B. Stowe, "Generation of models related to aluminium surface treatment using genetic programming", Proceedings of the 9th world congress on structural and multidisciplinary optimization (WCSMO9), Shizuoka, Japan, 2011;
- U. Armani, Z. Khatir, A. Khan, V. V. Toropov, A. Polynkin, H. Thompson, N. Kapur, "Control of physical consistency in metamodel building by genetic programming", Proceedings of the second international conference on soft computing technology in civil, structural and environmental engineering (CSC 2011), Chania, Greece, 2011;
- U. Armani, V. V. Toropov, A. Polynkin, S. Shahpar, "Application of explicit metamodels generated by genetic programming to computationally expensive optimisation problems", to be submitted to Structural and Multidisciplinary Optmization, 2012;
- U. Armani, S. Coggon, V. V. Toropov, "Derivation of deterministic design data from stochastic analysis in the aircraft design process", Proceedings of the Eleventh International Conference on Computational Structures Technology (CST 2012), Dubrovnik, Croatia, 2012;
- H. Lohse-Busch, C. Hühne, D. Liu, V. V. Toropov and U. Armani "Parametric optimization of a lattice aircraft fuselage barrel using metamodels built with genetic programming" in B.H.V. Topping and P. Iványi, (Editors), Proceedings of the Fourteenth International Conference on Civil, Structural and Environmental Engineering Computing (CC2013)), Civil-Comp Press, Stirlingshire, United Kingdom, paper 230, 2013. doi:10.4203/ccp.102.230

1.2 Metamodelling process

Metamodelling is a cyclic process. The main metamodelling stages can be identified as (Simpson et al. 2001, Vladislavleva 2008, Ramu et al. 2010):

1. data generation
2. metamodel generation or development
3. problem analysis and metamodel reduction

In the following sections the main issues of each step are briefly presented.

1.2.1 Data generation

Data generation consists of all the operations required for gathering, filtering and selecting the data that are used to generate the model. As introduced in Section 1.1, data are collected as a set of samples, records or fitness cases, each having the following structure:

$$\{\, x_1\; x_2\; \ldots\; x_N \quad y \,\} \quad | \quad x_i \in \mathbb{R}\;\; i = 1, \ldots, N \quad y \in \mathbb{R} \tag{1.3}$$

where x_i is a particular value of the input variable i, N the number of input variables assumed for the problem and y the corresponding output or response of the system under study. The measured response y may be affected by experimental or computational errors. In Eq. (1.3) a single scalar output is considered: the definition of sample can be extended to multiple output systems adding the remaining responses. A set of samples like the one defined in Eq. (1.3) is called *data matrix* (Vladislavleva 2008), *building data set* or *training data set* (Friedman 1991).

Each row of a data matrix define a *design point* in a bounded region of the system input space, called *design space*. For a given number of samples the accuracy of a metamodel can be increased using techniques called *Design of Experiments* (DoE) for the selection of the position of the design points. The optimal design point distribution indicated by a DoE depends however on the specific metamodelling technique used: the most common DoEs will be described in Section 1.2.5. It is worth mentioning that although the use of specific DoEs is recommended, in real-life industrial application it is not always possible to use an optimal design point distribution, as often metamodels have to be built on already existing data: in these cases the extraction of a reduced building data set using data filtering and balancing techniques may help reduce the cost of metamodel generation without compromising on the quality of the approximation (Harmeling et al. 2006, Vladislavleva 2008).

1.2.2 Metamodel generation

Metamodelling techniques process data matrices to generate an implicit or explicit mathematical function $\tilde{f}$ (for simplicity, the output y is assumed scalar):

$$y = \tilde{f}(\mathbf{x}) + \epsilon(\mathbf{x}) \tag{1.4}$$

although the error ϵ may be the result of non-controlled input variables or measurement errors (see Eq. (1.2)), it will be assumed that such error is only due to the approximation, as the metamodelling process only relies on data and it can influence neither modelling assumptions nor data generation.

The quality of a metamodel is a measure of the distance between the metamodel $\tilde{f}$ and the true underlying function f (Eq. (1.1)) it aims to approximate (Friedman 1991, Vladislavleva 2008). The integral error I is an ideal definition of such distance (Friedman 1991):

$$I = \int_D w(\mathbf{x})\, \Delta(\tilde{f}(\mathbf{x}), f(\mathbf{x}))\, d\mathbf{x} \tag{1.5}$$

where $\Delta()$ is a function defining the distance between $f(\mathbf{x})$ and $\tilde{f}(\mathbf{x})$ with $\mathbf{x}$ in the whole design space D and $w(\mathbf{x})$ is a weight function.

A first issue resulting from the definition of the integral error is that, no matter how the validation data set D_V has been chosen, the error on this data set depends necessarily on the training data set D_T used to build the model. Bishop (1995) follows a framework that is useful to understand what are the main issues arising in model building[2]. Considering an error function defined as sum of squares, he shows that the average squared error over the set of all possible training data sets D_T can be decomposed as sum of two terms, a squared *bias* and a *variance*:

$$\int \varepsilon_{D_T}[\{\tilde{f}(\mathbf{x}) - \langle y|\mathbf{x}\rangle\}^2] p(\mathbf{x}) d\mathbf{x} = (bias)^2 + variance \tag{1.6}$$

[2]the framework is reported by Bishop as an introduction to artificial neural networks, a modelling technique will be described in Section 1.2.4.2. Nonetheless, the same framework can be applied to any modelling technique.

where the integral is computed on the complete design space D, ε_{D_T} stands for the expectation operator over the ensemble of training data sets D_T and $\langle y|\mathbf{x}\rangle$ is the average value of the target y given the input point $\mathbf{x}$. $p(\mathbf{x})$ is the probability density of $\mathbf{x}$.

Bias and variance are defined as (Bishop 1995, p. 335):

$$(bias)^2 = \frac{1}{2}\int \{\varepsilon_{D_T}[y(\mathbf{x})] - \langle y|\mathbf{x}\rangle\}^2 p(\mathbf{x})d\mathbf{x} \tag{1.7}$$

$$variance = \frac{1}{2}\int \varepsilon_{D_T}[\{y(\mathbf{x}) - \varepsilon_{D_T}[y(\mathbf{x})]\}^2]p(\mathbf{x})d\mathbf{x} \tag{1.8}$$

The above listed definitions allows to ascribe a non-zero error of the model (as defined in Eq. 1.6) to two different phenomena. The model output for a given input $\mathbf{x}$ may be on average different from the target value $y(\mathbf{x})$, giving rise to a non-zero bias (Eq. 1.7). This happens when a model is too simple or not flexible: models affected by high bias are usually said to suffer from *oversmoothing*. On the opposite, if the metamodelling technique is really sensitive to the choice of the training data set, the resulting predicted values may exhibit a large variance for a given $\mathbf{x}$, hence the term variance in Eq. 1.8. Models that fits the training data perfectly are likely to have high variance, but lose the large scale behaviour of the true underlying function: such models are said to suffer from *overfitting*. It is clear that acceptable models represent a trade-off between bias and variance, oversmoothing and overfitting, which results in a minimum of the error defined in Eq. 1.6.

To further complicate things, the integral in Eq. 1.6 is evaluated on the whole design space D. For practical reasons is not obviously possible to evaluate it on the whole design space, and this forces to compute its approximation on a finite data set D_V, usually called *validation data set*. Usually indicators like the maximum absolute error, the root mean square error (RMSE) or the coefficient of determination R^2 (Jin et al. 2001) are used. The error on the validation data set gives an indication of how well the model is able to *generalise*, or predict the output of the underlying function on new input points, but it is clear that a zero error on the validation data set does not necessarily imply that the model has the same behaviour of the true underlying function.

In order to reach a good compromise between reduced bias and reduced variance, it has been observed that increasing concurrently the number of training points and the complexity of the model (i.e. number of weights) can be a successul strategy (Bishop 1995, Vladislavleva 2008). Prior knowledge regarding the expected behaviour of the true underlying function can also be effectively exploited to reduce bias (Bishop 1995) (an example will be given in Chapter 5, Section 5.4). Other common strategies to improve generalisation ability are reported in Bishop (1995), where the interested reader can find useful references. *Regularization* is a class of methods that relies on an additional penalty to the error function to curb the amount of highly non-linear variations in the model. In general the extra term is a sum of squared second partial derivatives of the model with respect to the input variables. *Weight decay* strategy penalises the sum of the squares of the adaptive parameters of the model, so it is effective when these parameters are linked to the curvature of the model. *Early stopping* is based on the idea that the parameters tuning process tends to generate a overfitted model, so stopping it before it reaches such state can be an effective way to improve generalisation ability. The *hold out method* uses a validation data set to check the quality of the model during the training process: to avoid overfitting even on the validation data set, when tuning is completed the performance of the model is assessed on a third, indipendent *test data set* (Lew et al. (2006) provide a good example of how these technique is used to set up a genetic programming code). *Cross validation* follows the same idea, addressing however the often scarce possibility to build extra validation data sets. The training data set is divided into N clusters of points, of which $N - 1$ are used to train the model and the one left out acts as validation data set. The average error computed over N validation clusters gives the final error measure. When the validation cluster is reduced to a single point, the method is generally called *leave-one-out method* (a useful example is reported in Viana and Haftka (2009)).

1.2.3 Problem analysis

The problem analysis stage aims at extracting information from the generated model to better understand the relative importance of each variable on the model response. This is the purpose of sensitivity analysis. A variety of techniques are available to study the influence of input variables on metamodel output: methods based on partial derivatives analysis are accurate but provide local information (Helton and Davis 2003), whereas techniques based on the analysis of output variance allow to rank input variables according to their contribution on output variance (Sobol 1993, Helton and Davis 2003, Sudret 2008, Arwade et al. 2010).

From the information acquired through sensitivity analysis, the whole metamodelling process can be repeated using a reduced set of the most important input variables (dimensionality reduction) to improve the accuracy of the model (Simpson et al. 2001, Vladislavleva 2008). The diagram in Fig. 1.1 represents the cyclic nature of this process: data generation, model development and sensitivity analysis are repeated to progressively refine the metamodel until an acceptable quality is reached.

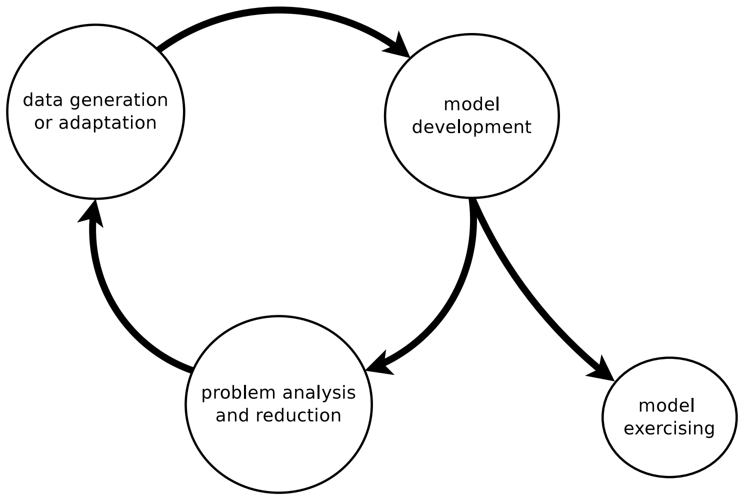


FIGURE 1.1: Metamodelling stages

Once a metamodel is generated and accepted, it can be used for different purposes. This stage is usually referred to as “model exploitation” or “model exercising” (Simpson et al. 2001) (see Fig. 1.1). Metamodels can help explore the behaviour of a system for new sets of input values, or systematically evaluated for optimisation, robust design purposes or uncertainty analysis (Helton and Davis 2003, Lew et al. 2006). As the metamodel response is in any case affected by error, validation of any predicted behaviour through a final set of simulation or experiments is usually performed. Significative examples of the entire metamodelling process and metamodel exploitation can be found in Harewood et al. (2007), Ramu et al. (2010) and Arwade et al. (2010).

1.2.4 Commonly used global metamodelling techniques

Three classes of metamodelling techniques are commonly used in academia and industry (Friedman 1991, Bishop 1995, Alvarez 2000, Jin et al. 2001, Simpson et al. 2001, Rasmussen and Williams 2006, Ramu et al. 2010):

- response surface methodologies (RSM)
- artificial neural networks (ANNs)
- kriging (also known as Gaussian Processes)

These techniques are commonly termed “global” as they generate a single implicit or explicit metamodel that approximates the response of the system under study on the whole design space. Due to the non-linearities that the response to approximate may exhibit, generating globally accurate metamodels is generally more difficult than producing local approximations. In Section 1.2.7 it will be shown how partitioning the design space is just one of the many strategies used by researchers to improve metamodel accuracy.

1.2.4.1 Response surface methodologies (RSM)

Response surface methodologies assume that the mathematical structure (Vladislavleva 2008) of the metamodel is defined a priori as a linear combination of a set of mathematical bases, also called *model bank* (Polynkin et al. 2008), defined by the user. For this reason these techniques are called “parametric regression techniques” (Friedman 1991, Sebag et al. 1997, Vladislavleva 2008). The numerical coefficients of the linear combination are identified by a least-squares approach using the building data set provided by the user (Simpson et al. 2001, Ramu et al. 2010)).

The choice of the mathematical bases is wide. For their simplicity and smoothness, first (Eq. 1.9) and second order (Eq. 1.10) polynomials are typically used (Simpson et al. 2001, Ramu et al. 2010):

$$\tilde{y}(x_1, \dots, x_N) = \beta_0 + \sum_{i=1}^{N} \beta_i x_i \tag{1.9}$$

$$\tilde{y}(x_1, \dots, x_N) = \beta_0 + \sum_{i=1}^{N} \beta_i x_i + \sum_{i=1}^{N} \beta_{ii} x_i^2 + \sum_{i=1}^{N} \sum_{j=1, i>j}^{N} \beta_{ij} x_i x_j \tag{1.10}$$

where β_i, β_{ij} are coefficients to be tuned.

To approximate non-linear responses the order of the polynomial can be increased to 3-rd, 4-th or higher orders (Ramu et al. 2010). However, higher order polynomials implies a larger number of coefficients to be tuned, hence a larger building data set (Helton and Davis 2003). Noise and undesired oscillations on the design space boundary are also to be expected. To tackle highly non-linear problems usually polynomials are abandoned in favour of more complex sets of functions. For example, Polynkin et al. (2008) used intrinsecally linear functions and rational functions. In order to avoid trial and error for the selection of the best set of functions, some knowledge of the likely behaviour of the response to be approximated is required.

A wise selection of the model bank may ease metamodel analysis and exploitaiton. The use of Hermite, Lagrange, Jacobi and Laguerre polynomials in Polynomial Chaos Expansion (PCE) dramatically simplifes metamodel sensitivity analysis: the variance contribution of each variable (Sobol indices - Sobol (1993)) to the total output variance can be extracted analytically from the PCE, avoiding the use of more computationally intensive Monte Carlo approaches (Eldred et al. 2008, Sudret 2008, Arwade et al. 2010).

1.2.4.2 Artificial neural networks (ANNs)

Artificial neural networks (ANNs) inception can be ascribed to the model of artificial neuron developed by McCulloch and Pitts (1943) in the 1940s. The first artificial neural networks were introduced by Rosenblatt (1958), who called them *perceptrons*, and Widrow and Lehr (1990), which instead coined for them the term *adalines*. Since then, their development have made them fit for classification and regression purposes in different branches of engineering, science and finance (Simpson et al. 2001, Pierce et al. 2006 2008).

ANNs are appreciated for metamodelling tasks as they are "universal approximators", in other words they can approximate to an arbitrary accuracy any continuos function and its derivatives, provided that ANNs of proper complexity are chosen (Bishop 1995, Pierce et al. 2006). ANN metamodels can be thought as "grids" composed of single operators called "neurons", "perceptrons" or "units". A perceptron is a mathematical model that associates a set of input variables **x** with an output value y. The relationship between **x** and y is assumed to consist of a linear combination of the input variables x_i which is transformed by a non-linear function $g(\,\cdot\,)$, also called *activation function* (Bishop 1995, p. 117), as shown in 1.11:

$$y = g\left(\sum_{i=1}^{N} w_i x_i + w_0\right) = g\left(\sum_{i=0}^{N} w_i x_i\right) \quad | \; x_0 = 1 \tag{1.11}$$

where w_0 is called *bias* and N is the dimension of the input space. The typical graphical representation of a unit is given in Fig. 1.2.

A common choice for the activation function $g(\,\cdot\,)$ is the *logistic sigmoid* (Bishop 1995, Simpson et al. 2001), for reasons that will be explained shortly:

$$g(a) = \frac{1}{1 + e^{-a}} \tag{1.12}$$

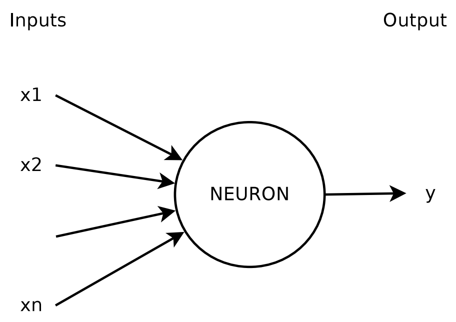


FIGURE 1.2: Neuron typical representation

but other functions can be used (for example the Heaviside step function (Bishop 1995)).

Artificial neural networks are made linking a certain number of neurons, in order to make an "architecture". Architectures are defined by the number of neuron layers and by rules that constrain the links that can be established between units. If links are not permitted between a neuron and any other neuron that contributes directly or indirectly to the generation of its inputs, the neural network is said to have a *feed-forward* architecture. An example of a two-layer feed-forward architecture is shown in Fig. 1.3, in which the set of units between the input and the output ones are called *hidden* units. Acting

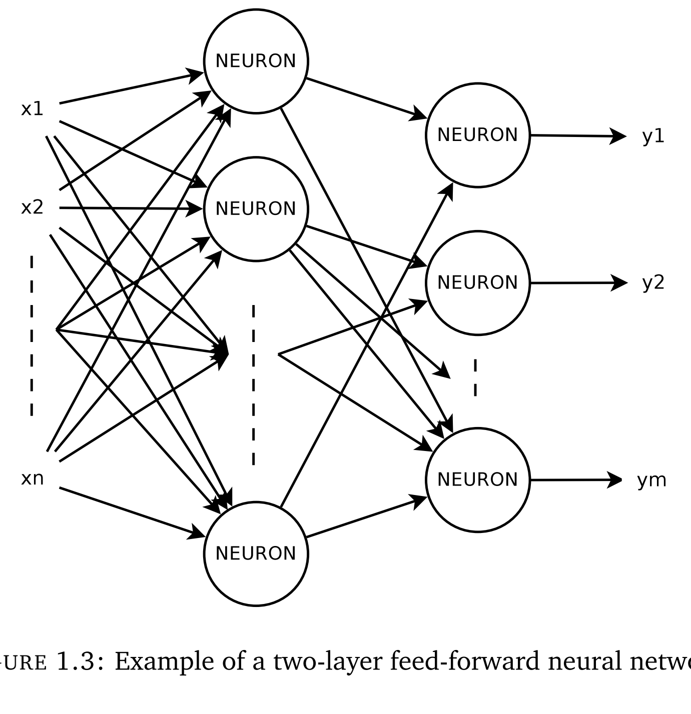


FIGURE 1.3: Example of a two-layer feed-forward neural network

on the size and the structure of the architecture the accuracy of the approximation produced by neural networks can be increased (Simpson et al. 2001). A major strength of the two-layer feed-forward architecture represented in Fig. 1.3 is that it can approximate to arbitrary accuracy any continuous function and its derivatives, provided that the number of hidden units is sufficiently large (Bishop 1995). Typically ANNs architectures used for engineering applications employ tens or even hundreds of neurons (Flood 2011).

It is important to stress that mappings represented by feed-forward neural networks can be cast into explicit models. In particular, the mathematical expression of each output y_k of the two-layer neural network represented in Fig. 1.3 is (Bishop 1995, p. 119):

$$y_k = \tilde{g}\left(\sum_{j=0}^{M} w_{kj}^{(2)} g\left(\sum_{i=0}^{N} w_{ji}^{(1)} x_i\right)\right) \tag{1.13}$$

where the sums expressing the linear combinations as in 1.11 start from 0 in order to include the bias ($x_0 = 1$). Symbols $g(\cdot)$ and $\tilde{g}(\cdot)$ represent activation functions. In general, each unit can have its own activation function, different from the others.

In order to use neural networks, the weights of each neuron (see for example w_i with $i = 0, \ldots, N$ in Eq. (1.11)), have to be tuned. The weight tuning process is generally called *training* or *supervised learning* (Mohammadi and Mahdavi 2008, Flood 2011) and aims at minimising a metric defined as a function of the errors between the response predicted by the neural network and the actual one on a given training data set (Simpson et al. 2001, Pierce et al. 2006 2008). The metric to be minimised can be for example the sum of the square errors (Bishop 1995, Orr 1996, Simpson et al. 2001) but others can also be used (Chetwynd et al. 2006, Pierce et al. 2008).

Two iterative approaches are commonly used to solve such minimisation problem, deterministic (gradient-based) and stochastic. In the former, the *back-propagation* algorithm (Bishop 1995, Rogers and LaMarsh 1995, Simpson et al. 2001, Pierce et al. 2006, Mohammadi and Mahdavi 2008, Flood 2011) is used to compute first the gradient of the error metric as a function of the weights, and then improved optimal sets of weights are progressively found exploiting this information using a gradient-based technique (for example using conjugate gradient method (Pierce et al. 2008)). The main advantage of this approach is that it is computationally efficient and therefore faster than approximating the derivatives using numerical differentiation (Bishop 1995). Derivatives are indeed found through multiplications of the values at the input and output ends of each weight, which are already computed during the evaluation of the neural network (*forward-propagation*), and the derivative of the activation function. Using the logistic sigmoid activation function (Eq. 1.12) gives the additional advantage that its derivative can be expressed as a function of the activation function itself. A detailed description of the process can be found in Bishop (1995, p. 141-146). Gradient-based error minimisation may however lead to local minima of the error metric, let alone the fact that the set of weight found depends on the initial weight provided by the user (Pierce et al. 2006, Chetwynd et al. 2006) and the technique can be applied only if the activation functions and the error function are differentiable (Bishop 1995). The latter class of training methods uses population-based algorithms, which are able to explore globally the error surface and to avoid local minima (differential evolution is for example used in Chetwynd et al. (2006)). However, they are necessarily less efficient.

Radial basis functions networks (RBF networks) can be analysed using the same formalism of artificial neural networks, although the idea they are based on and the range of techniques that can be used to train them are different from ANNs. Radial basis functions approximation is based on the idea that any continuous function can be approximated with arbitrary accuracy by a linear superposition of localised bump functions (Bishop 1995).

The bump functions are generated by so called radial functions, which exhibit a monotonic increase or decrease function of the distance (radius) from a central point $\mathbf{x}_c \in \mathbb{R}^N$ (Orr 1996, Jin et al. 2001, Rendall and Allen 2008). Typical radial functions are the Gaussian, which decreases monotonically with the distance from the centre $\mathbf{x}_c$:

$$h(\|\mathbf{x}-\mathbf{x}_c\|) = e^{-\dfrac{\|\mathbf{x}-\mathbf{x}_c\|^2}{2\sigma^2}} \tag{1.14}$$

where σ and $\mathbf{x}_c$ are parameters to be tuned (Bishop (1995, p. 169) also reports the generalised expression of the Gaussian, having however more parameters to be tuned). The multiquadric, which increases monotonically with the distance from the centre, is another radial basis function:

$$h(\|\mathbf{x}-\mathbf{x}_c\|) = \frac{\sqrt{r^2 + \|\mathbf{x}-\mathbf{x}_c\|^2}}{r} \tag{1.15}$$

where r is a numerical parameter used to tune the rate of increase of the radial function.

RBFs networks can then be represented as ANNs having a two-layer feedforward architecture, in which the weights belonging to the first layer are used to determine the mathematical features of the basis functions (see parameters σ, $\mathbf{x}_c$, r in 1.14 and 1.15) and only the weights of the second layer impose a relative importance of the activations as in standard ANNs (Bishop 1995, Mohammadi and Mahdavi 2008). Fig. 1.4 shows the architecture of a typical RBF network (for simplicity, the case of a multiple input - single output mapping has been considered).

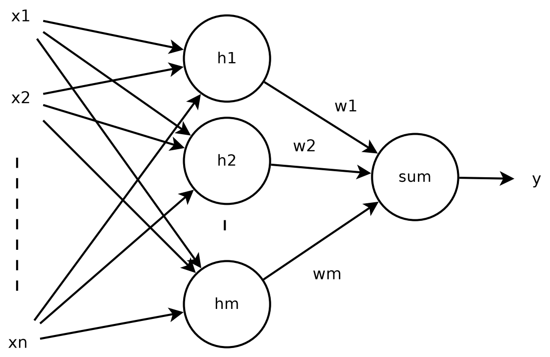


FIGURE 1.4: Representation of a RBF neural network

As ANNs, even RBFs networks can be cast in their explicit form, as a linear combination of radial basis functions (Bishop 1995, Jin et al. 2001, Silva et al. 2007, Rendall and Allen 2008, Mohammadi and Mahdavi 2008):

$$\tilde{y} = \sum_{i=1}^{M} w_i h_i(\|\mathbf{x} - \mathbf{x}_{c_i}\|) + w_0 = \sum_{i=0}^{M} w_i h_i(\|\mathbf{x} - \mathbf{x}_{c_i}\|) \quad | \; h_0 = 1 \tag{1.16}$$

where w_i are the weights of the radial basis function and M is the number of units in the hidden layer (usually far fewer then the size of the training data set (Bishop 1995)). The mathematical structure reported in 1.16 affords RBFs networks to be trained in a different way than the one used for ANNs, which is faster than back-propagation based search algorithms (Nabney 2003, Mohammadi and Mahdavi 2008). If Eq. 1.16 is used to build a model from a training data set made of N points $(\mathbf{x}_i, y_i)$, the error ε_i in each point can be written in matrix notation as:

$$\varepsilon_i = y(\mathbf{x}_i) - \tilde{y}(\mathbf{x}_i) = y(\mathbf{x}_i) - \mathbf{Hw} \tag{1.17}$$

where H is the matrix having as rows the radial basis functions $h_i(\|\mathbf{x} - \mathbf{x}_{c_i}\|)$, ..., $h_i(\|\mathbf{x} - \mathbf{x}_{c_M}\|)$ evaluated on the N training data points and $\mathbf{w}$ is the (column) vector of weights to be found. The weights in Eq. 1.17 that minimise the sum-of-squares error function can be found using a direct method using linear matrix inversion techniques (Bishop 1995), which is faster than the iterative methods exploiting back-propagation algorithm.

It should be mentioned however that this possibility implies that all the weights that define the mathematical properties of the radial basis functions (see for example σ, $\mathbf{x}_c$, r

in 1.14 and 1.15) are already known. There are techniques that use only the coordinates in the input space of the training sample points to set such parameters (*unsupervised* methods) but, although fast, they may lead to suboptimal identification of radial basis functions (Bishop 1995). In case optimisation of these properties is required, weight optimisation methods used for ANNs (based on back-propagation algorithm or population based strategies) can be applied.

1.2.4.3 Kriging method

Kriging is also referred to as Gaussian Process (Rasmussen and Williams 2006, p. 30) or Design and Analysis of Computer Experiments (DACE) (Simpson et al. 2001, p. 132). A Gaussian Process is "a collection of random variables, any finite number of which have a joint Gaussian distribution" (Rasmussen and Williams 2006, p. 13). In other words, if a randomly selected set of n outputs $f(\mathbf{x}_i)$ is taken, where f is the function we want to approximate, then the joint probability density p of this set is described through a multivariate Gaussian distribution:

$$p(f(\mathbf{x}_1), f(\mathbf{x}_2), \ldots, f(\mathbf{x}_n)) = N(\mu, \Sigma) \tag{1.18}$$

$$= (2\pi)^{-D/2}|\Sigma|^{-1/2}exp(-\frac{1}{2}(\mathbf{f}-\bar{\mathbf{f}})\Sigma^{-1}(\mathbf{f}-\bar{\mathbf{f}})) \tag{1.19}$$

where N stands for Gaussian distribution (also known as normal distribution), μ is the mean of the distribution (vector of size n, defined as the average of $f(\mathbf{x}_i)$, $\bar{\mathbf{f}}$) and Σ is the covariance matrix of the distribution. Often the values $f_i = f(\mathbf{x}_i)$ are referred to as the "variables" of the Gaussian Process.

An important step in Gaussian Processes is the definition of single entries of the covariance matrix Σ. It is assumed that the single covariance terms $k(y, y')$ the covariance matrix is composed of, which define the relationship between the outputs $f(\mathbf{x}_i)$, can be written as a function of the sole inputs $\mathbf{x}_i$ (Rasmussen and Williams 2006, p. 14, 18):

$$k(y, y') = k(\mathbf{x}, \mathbf{x}') = k \tag{1.20}$$

So the covariance matrix Σ is defined as:

$$\Sigma = \begin{bmatrix} k(\mathbf{x}_1, \mathbf{x}_2) + \sigma_n^2 & \cdots & k(\mathbf{x}_1, \mathbf{x}_n) \\ \vdots & \ddots & \vdots \\ k(\mathbf{x}_n, \mathbf{x}_1) & \cdots & k(\mathbf{x}_n, \mathbf{x}_1) + \sigma_n^2 \end{bmatrix} \tag{1.21}$$

where $k(\mathbf{x}_i, \mathbf{x}_j)$ are the covariance functions previously introduced. The covariance functions turn into variances when $\mathbf{x}_i = \mathbf{x}_j$: in this case the variance term σ_n^2 has been added to account for the noise on the observation (the same variance for each observation is assumed in matrix 1.21). In case observations can be assumed noise-free, the variance term σ_n^2 can be removed.

Once all the elements required by the multivariate Gaussian probability density have been defined, such probability density acts as a "structure" of the metamodel allowing to extract the expected reponses on the points where predictions are desired (predictions). This process is called *conditioning*, and the particular definition of the multivariate Gaussian probability density affords to perform it analytically: the interested reader can find the details in Rasmussen and Williams (2006, p. 15-17). As a final result, predictions and corresponding confidence intervals are returned, but no analytical expression is produced: to extract predictions on a new test data set the conditioning process has therefore to be repeated.

The selection of the covariance function $k(\mathbf{x}_i, \mathbf{x}_j)$ deserves particular attention. The squared exponential covariance function is commonly used (Simpson et al. 2001, Jin et al. 2001, Bonte et al. 2005, Rasmussen and Williams 2006):

$$k(\mathbf{x}_i, \mathbf{x}_j) = \sigma_f^2 \, exp \left[-\frac{1}{2} (\mathbf{x}_i - \mathbf{x}_j)^T M (\mathbf{x}_i - \mathbf{x}_j) \right] + \sigma_n^2 \, \delta_{ij} \tag{1.22}$$

where σ_f, M and σ_n are called *hyperparameters* (Rasmussen and Williams 2006, p 20), n being the number of records in the building data set. Many other standard covariance functions can be used, while new ones can even be composed adding together existing functions (a useful overview is given in Rasmussen and Williams (2006, Ch. 4,5)). The particular values of the hyperparameters used in the covariance function allow to change the behaviour of the resulting model, which can either interpolate or smoothen the input data, depending if noise on observations is assumed ot not (see Simpson et al. (2001), Ramu et al. (2010) and also Rasmussen and Williams (2006, p. 20) for a useful example).

The wide range of covariance functions available if on the one hand increases the flexibility of Gaussian Process technique, on the other hand introduces the issue of establishing a rigorous way to optimise the selection of the covariance function. Two levels of optimisation are therefore implicit in Gaussian Process modelling: the selection of the covariance function "structure" and the selection of the hyperparameters once such structure has been chosen (Rasmussen and Williams 2006, p. 108-111). The strategy that is generally followed to perform such selection is based on Bayes' theorem, as it implies the maximisation of the marginal likelihood $p(\mathbf{y}|X,\theta)$, or the probability of the observations $\mathbf{y}$ given the hyperparameters θ and the structure of the covariance function H_i. The advantage of such approach is that the marginal likelihood can be computed analytically for the multivariate Gaussian probability density (Rasmussen and Williams 2006, p. 112). Its logarithm is reported below (Rasmussen and Williams 2006, p. 113):

$$log\,p(\mathbf{y}|X,\theta) = -\frac{1}{2}\mathbf{y}^T K_y^{-1}\mathbf{y} - \frac{1}{2}log|K_y| - \frac{n}{2}log(2\pi) \tag{1.23}$$

where $K_y = K_f + \sigma_n^2 I$ is the covariance matrix for the noisy target $\mathbf{y}$ obtained from the noise-free covariance matrix K_f. The extraction of kriging model responses in unsampled region implies therefore solving an unconstrained nonlinear optimisation problem (Simpson et al. 2001), that is finding the maxima of Eq. 1.23, with all the related issues (local optima, dependence of the found optimum on the initial guesses, etc..). The other advantage of this approach, called Bayesian inference, is that it automatically involves a trade-off between model fit and model complexity, differently to what happens in parametric techniques like RMS and ANNs where this balance is reached acting on parameters and model structure, often relying on cross-validation strategies (although bayesian learning of weights is also used for ANNs (Bishop 1995, Ch. 10)).

Often the response $y(\mathbf{x})$ of a system is approximated either using a deterministic non-zero mean or adding to the Gaussian Process $f(\mathbf{x})$ a predefined explicit global model $g(\mathbf{x})$, usually a polynomial (Jin et al. 2001) (Rasmussen and Williams 2006, p. 27):

$$\tilde{y}(\mathbf{x}) = g(\mathbf{x}) + f(\mathbf{x}) \tag{1.24}$$

The global behaviour of the metamodel is given by the first term, the function $g(\mathbf{x})$, while the Gaussian Process $f(\mathbf{x})$ introduces local variations to locally fit the building data set (basically the Gaussian Process is used to model the residuals).

1.2.5 Selection of the design of experiment

The use of experimental design techniques, or *Design of Experiments* (DoE), aims at increasing the efficiency of the metamodel generation process. DoEs were originally developed for physical experiments, to reduce the influence of the random errors affecting the measurement process (Alvarez 2000, Simpson et al. 2001). The same methodologies were later extended to the design of deterministic computer experiment, which are less prone to noise, although some errors may still be generated by incomplete convergence, truncation errors or problem discretisation (Alvarez 2000, Bonte et al. 2005, Ramu et al. 2010). According to the assumptions on input data random error, DoE optimality criteria change (Simpson et al. 2001).

A design of experiment is a list of coordinates defining the points in a design space in which the response of the system under study has to be sampled. The input variables are defined as *factors*, and the number of input configurations to be sampled per factor are called *levels* (Simpson et al. 2001, Ramu et al. 2010).

The range of DoEs is wide. A proper DoE selection does not only have to take into account the metamodelling technique used to process the gathered data, but it also has to consider the cost of producing or obtaining the system response and the purpose of the metamodel (i.e. model identification or sensitivity analysis) (Simpson et al. 2001, Ramu et al. 2010). In the following sections a survey of the most common DoEs is presented.

1.2.5.1 Full factorial design

Full factorial design is the most basic DoE. All combinations of all factors at all levels are sampled. The number of levels is assumed equal for each factor, and as a result the number of DoE points is $N_l^{N_f}$, where N_l is the number of levels and N_f the number of factors.

The main advantage of this design is the extensive information that can be gathered. In case response surface methodology is chosen for metamodel building, a 3^{N_f} design allows for the estimation of the main (linear) and quadratic effects and interactions of a second order polynomial in N_f input variables (Alvarez 2000, Simpson et al. 2001). The technique becomes however prohibitive for high-dimensional and expensive systems. In Fig. 1.5 an example of a tridimensional full factorial DoE having 5 levels per factor is shown (125 points).

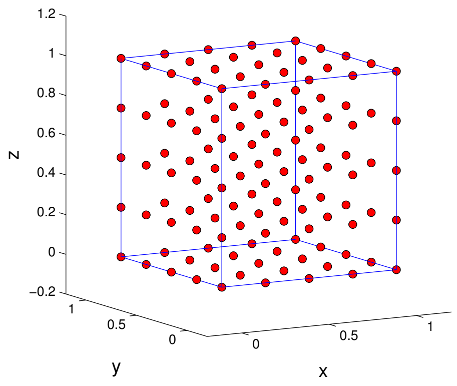


FIGURE 1.5: Example of full factorial DoE for three factors (design space: (0, 1)×(0, 1)×(0, 1))

Fractional factorial designs are obtained from full factorial design removing a certain percentage of the original points (Alvarez 2000, Simpson et al. 2001). For their reduced size but also lower sampling resolution if compared to full factorial designs, *fractional factorial* designs are used in the early modelling stages, in particular for screening design variables and identify those with the greatest influence on the response (Alvarez 2000, Simpson et al. 2001, Lew et al. 2006).

1.2.5.2 Central composite design

Central composite designs are 2-level full-factorial design augmented with a central point and two additional 'star' points for each factor, located at a given distance α from each factor lower and upper bound. In total five levels for each factor are used, so the total number of point in the DoE is $2^{N_f} + 2N_f + 1$ for N_f factors. In case the 'star' points are located on the faces of the hypercube defined by the 2-level full factorial design, the design is referred to as *face-centred central composite design*. An example of central composite design for 3 factors is shown in Fig. 1.6.

Central composite designs allow for the evaluation of linear and quadratic terms in response surface modelling (Alvarez 2000, Ramu et al. 2010). However, the number of points may be still excessive in case of high-dimensional and costly systems (Alvarez 2000, Ramu et al. 2010).

1.2.5.3 Box-Behnken design

Box-Behnken designs are a family of three-level designs, used in particular for fitting second order polynomials (RSM). They are available only for a limited range of factors (3

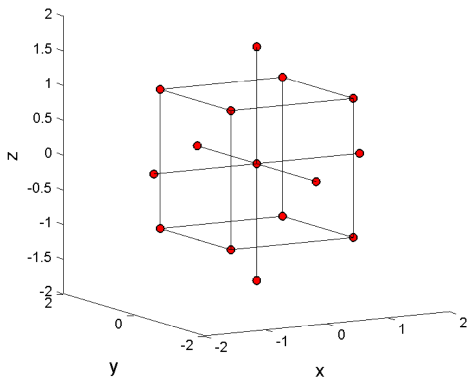


FIGURE 1.6: Example of central composite DoE for three factors (design space: (-1, 1)×(-1, 1)×(-1, 1))

to 7) (Ramu et al. 2010) and they do not contain the points corresponding to the vertices of the hypercube defined by the lower and upper bounds of each factor, as shown in Fig. 1.7. As a result, Box-Behnken designs are suitable for situations where factors cannot concurrently assume values at the extremes of their ranges, or these configurations are expensive or impossible to test (Simpson et al. 2001, Ramu et al. 2010).

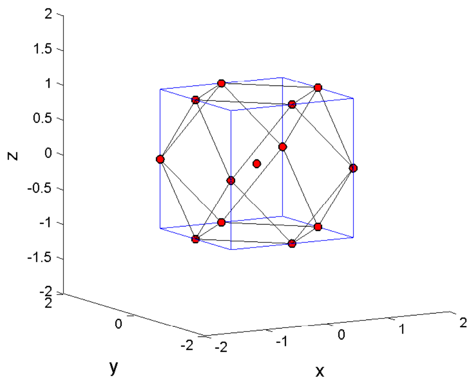


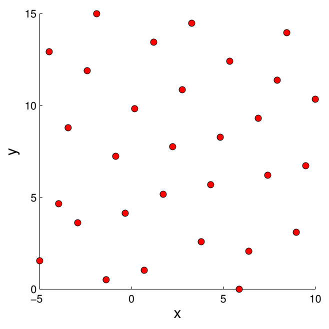
FIGURE 1.7: Example of Box-Behnken DoE for three factors (design space: (-1, 1)×(-1, 1)×(-1, 1))

1.2.5.4 Latin Hypercube design

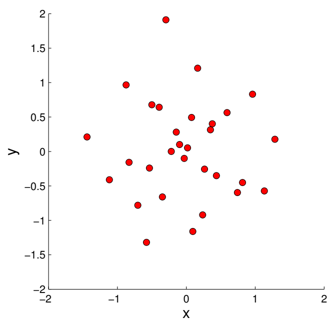
Latin Hypercube designs belong to the class of so-called ‘space filling’ designs (Simpson et al. 2001), whose main feature is to consider all regions of the design space equally important. As a result, they ensure that all input variables are represented, no matter if the system response is dominated only by few of them: for this reason their use is recommended in the early stage of design, when variables’ contribution and metamodel form cannot be specified (Simpson et al. 2001).

A Latin hypercube design is composed of a number of points equal to the number of levels N_l, set by the user (Alvarez 2000). To guarantee a uniform exploration of the design space, stochastic or deterministic approaches are commonly used to generate the position of the DoE points. Deterministic approaches implicitly assume that all factors follow a uniform probability distribution and rely on optimisation algorithms that maximise the minimum distance between DoE points or minimise more complex cost functions. Audze and Eglais's cost function for example is based on the analogy between a set of points and a system made of unit mass particles. The repulsion force between points is modelled through a potential energy defined as the reciprocal of the squared distance between points, so that a optimal uniform point distribution can be obtained minimising the sum of the points' potential energies (Alvarez 2000, Bates et al. 2004). An example of deterministic Latin Hypercube design for two factors is shown in Fig. 1.8A.

Following instead a probabilistic approach, the sampling range of each factor is divided in a number of subranges equal to the desired levels, in order to let each sub-range have the same probability to be sampled, according to the probability distribution functions (pdf) of the input variables. A single DoE point coordinate is then selected from each subrange, generally using a uniform probability distribution. These single coordinates are then assembled into DoE points through random association (Helton and Davis 2003). Fig. 1.8B provides an example of stochastic Latin Hypercube design for two factors.



(A) uniform (deterministic) Latin Hypercube design (design space: (-5, 10)×(0, 15))



(B) stochastic Latin Hypercube design (for input variables modelled by a normal distribution in (-2, 2)×(-2, 2))

FIGURE 1.8: Examples of uniform (deterministic) (A) and stochastic (B) Latin Hypercube DoEs for two factors

Latin Hypercube designs are appreciated for their space-filling properties and their versatility (Simpson et al. 2001). They are commonly used for generating kriging building data sets (Alvarez 2000, Bonte et al. 2005, Ramu et al. 2010) and usually recommended in response surface methodologies during the early stages of metamodelling when the best mathematical structure of the metamodel is unknown (Simpson et al. 2001). It should however not be neglected that Latin Hypercube designs cannot be used in case input variables are correlated, they are not reproducible (randomly generated) and that the user is responsible to set the proper DoE size to ensure that the sampling density is high enough for the selected metamodelling technique (Ramu et al. 2010).

1.2.6 Advantages and drawbacks of established techniques

The brief description provided in the previous sections is sufficient to assess the strengths and drawbacks of the most common metamodelling techniques.

1.2.6.1 Response surface methodologies

Response surface methodologies, as in general all techniques based on parametric regression, are explicit, so design variables' contributions to the system response can be identified easily, at least if the mathematical bases are simple (for example low order polynomials). The resulting metamodels smoothness is appreciated for reducing the problem of noise. Finally, RSM is really easy to use.

On the other hand, the parametric nature of the technique requires that the user has some knowledge regarding the possible mathematical behaviour of the response to be approximated (Lew et al. 2006), otherwise the overall computational cost of metamodel building is likely to be increased by preliminary screening aimed at finding a set of proper mathematical basis functions (Singh et al. 2007, Ramu et al. 2010). As acknowledged by Polynkin et al. (2008), the selection of a proper basis can increase dramatically the accuracy of the metamodel. The common approach to use low-order polynomials may lead to poor metamodel accuracy in case of highly nonlinear problems (Jin et al. 2001), and increasing the order of the polynomials may introduce noisy behaviour and also may require a dramatic increase in the DoE size for tuning the polynomials' coefficients (Helton and Davis 2003, Eldred et al. 2008). As a result, RSM is not recommended for highly non-linear and highly dimensional problems (Simpson et al. 2001, Jin et al. 2001).

1.2.6.2 Artificial neural networks

Artificial neural networks and radial basis function networks are able to generate accurate metamodels of highly non-linear and high dimensional problems (Simpson et al. 2001, Chetwynd et al. 2006), although Flood (2011) reported that five or six independent variables are a typical limit to the dimensionality of the problems that can be approached by ANNs. ANNs can process both deterministic (noise-free) as well as experimental data (Rogers and LaMarsh 1995, Chetwynd et al. 2006).

A major criticism is that ANNs lack transparency, as in general they do not produce explicit metamodels (although in simple cases an analytical expression can be extracted - see for example Pierce et al. (2008, p. 1398)). Secondly, some prior knowledge or preliminary testing is required for a proper selection of the activation functions, the ANN architecture and the total number of neurons (Pierce et al. 2006, Chetwynd et al. 2006, Pierce et al. 2008). All these parameters may affect considerably the accuracy of the ANN, being lack of generalisation and overfitting (modelling of noise instead of the underlying structure of the data) the main risks (Pierce et al. 2006 2008). Rogers and LaMarsh (1995) proposed an empirical method to determine (manually) the number of neurons in ANNs having a single hidden layer. In Pierce et al. (2006), Chetwynd et al. (2006), Pierce et al. (2008) different training, validation and test data sets were used to select the optimal number of neurons in the the hidden layer of a two-layer ANN. A few researchers ventured further, suggesting the use of genetic algorithms to automise the process of finding optimal ANN architectures (an example of ANN architecture evolution can be found in Zhang and Mühlenbein (1995) and in Flood (2011)). This possibility is motivated by the ability of genetic algorithms to perform a directed search in a space of user-defined programs. Genetic algorithms' paradigm will be introduced in Chapter 2.

Thirdly, ANNs training cost is also to be reckoned with. In terms of the computational cost of training data gathering or generation, the total number of samples required may be high (Simpson et al. 2001), especially if independent training, validation and test data sets are used to select the optimal ANN's architecture and size (Chetwynd et al. 2006, Pierce et al. 2008). As for training time, it may vary considerably according to the architecture chosen, the number of neurons and, in case a backpropagation algorithm is used, the set of initial weight selected (Rogers and LaMarsh 1995). In general however ANNs training time is far larger than the time required for tuning a parametric model (Lew et al. 2006) and may pose a limit to the total number of samples that can be used (Chetwynd et al. 2006, Flood 2011). Simpson et al. (2001) in this regard reported (p. 134): "...the procedure is to toss the data directly into the NN software, use tens of thousands of parameters in the fit, let the workstation run 2-3 weeks grinding away doing the gradient descent, and *voilá*, out comes the result"[3]. Mohammadi and Mahdavi (2008) also provided some evidence of the high training time (more than 10 hours for backpropagation ANNs). To cope with the computational cost of the technique, recurring to parallelised computer codes is strongly suggested (Rogers and LaMarsh 1995, Simpson et al. 2001). Radial basis function networks are generally faster to train than more complex ANNs (Mohammadi and Mahdavi 2008).

1.2.6.3 Kriging method

Kriging method is flexible and able to deal with highly non-linear and mid-dimensional problems (less than 50 input variables (Simpson et al. 2001)). In case it is used as an interpolation technique, problems with noisy input data may arise. Kriging requires the user to define the structure H_i of the covariance function and, if used to model the residuals as in Eq. (1.24), the definition of an additional global metamodel, so it is not as easy to set up as RSM. Training is time consuming for high dimensional problems and for large training data sets (Rasmussen and Williams 2006), and fitting problems have been reported for some full factorial and central composite designs (Jin et al. 2001). The metamodel generated by kriging is not explicit, and the influence of each design variable on the output response cannot be easily evaluated (Jin et al. 2001).

A synthesis of the advantages and drawbacks of each of class of techniques is reported in Table 1.1 (the content was adapted from (Simpson et al. 2001, pag. 143)).

1.2.7 Local and mid-range approximation techniques

Global and transparent metamodels are ideal for their simplicity, interpretability and practically inexpensive evaluation. Some compromise on accuracy should however be accepted for high dimensional and highly non-linear systems. In such cases mid-range or local approximation techniques usually provide more accurate metamodels. This class

[3]The author would like to thank Mr. René Meissner from Airbus for the interesting discussion on neural networks at DiPART conference held in ASRC, Bristol in December 2011. The application of neural network metamodelling to industrial problems having a number of input variables smaller than 100 was confirmed to be incredibly demanding in terms of the size of the design of experiment and the duration of the neural network training.

TABLE 1.1: Recommendations on metamodelling technique selection

Technique	Characteristics / Appropriate uses
Response surfaces	• well-established and easy to use • best suited for applications with random error • appropriate for **low-dimensional problems** (< 10 input variables)
Neural networks	• good for highly-nonlinear or **high dimensional problems** • best suited for deterministic applications • high computational cost (often more than 10000 training data points); best for repeated application
Kriging (Gaussian Process)	• extremely flexible but complex • well-suited for deterministic applications • can handle **mid-dimensional problems** (< 50 input variables) • suited for smooth underlying functions, not varying over many orders of magnitude • criticism of limited support reported in Simpson et al. (2001) overcome by extensive research and availability of open-source toolboxes (see for example Rasmussen and Nickish (2010), Rasmussen (2006) and the links to other Gaussian Process packages provided in the former)

of methods leaves the idea of approximating a response through a single metamodel extending on the whole design space. Unfortunately this results in loss of simplicity and transparency.

Taylor expansion is one of the simplest local approximation techniques. The system response y is approximated in a region around a sampled design point $\mathbf{x_0}$ as a sum of derivatives of increasing order:

$$y = f(\mathbf{x_0}) + \sum_{i=1}^{N} \left[\frac{df}{dx_i}\right]_{\mathbf{x_0}} (x_i - x_{0i}) + \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \left[\frac{d^2 f}{dx_i dx_j}\right]_{\mathbf{x_0}} (x_i - x_{0i})(x_j - x_{0j}) + o(\|\mathbf{x} - \mathbf{x_0}\|^2) \tag{1.25}$$

where N is the dimensionality of the problem. Taylor expansion is still explicit, but may be accurate only in a narrow region centred in the known design point $\mathbf{x_0}$ for highly non-linear functions. Additional computational cost results from the estimation of the partial derivatives. Yet, Taylor expansion is important conceptually, as it shows that the assessment of the non-linearity of a (differentiable) function has to take into account the extension of the input domain around the known point: theoretically any (differentiable) response can be approximated by a constant or linear function on a narrow enough design space centred in $\mathbf{x_0}$. Furthermore, the reduction of the input region usually reduces the level of interaction between input variables. Splitting the design space and performing a piecewise regression is a strategy that motivates many local and mid-range approximation and metamodel-based optimisation techniques.

Recursive partitioning regression (Friedman 1991) is based on the recursive splitting of the design space: in each subregion the system response is approximated usually by constant or linear functions. The adaptive control on the size of the partitions may ease the analysis of the interactions among the input variables. A main restriction of the method is the lack of continuity at the subregion boundaries and the lack of transparency.

Splines are popular piecewise polynomial fitting procedures (Friedman 1991, Quarteroni et al. 2000). The building data set is interpolated or fitted through a set of polynomials of order k, each defined in a subregion of the design space. The overall metamodel is required to be continuous to order $k-1$. The application of splines to high dimensional problems ($N > 2$) is hindered by the rapid increase in the DoE size required to tune the coefficients of the polynomials (curse of dimensionality) (Friedman 1991).

Multivariate adaptive regression splines (MARS) (Friedman 1991) is based on recursive partitioning regression. The main weakness of recursive partitioning, the lack of continuity, is solved replacing the constant or first order terms with splines of order q, so that continuity can be achieved in the metamodel and its derivatives. A MARS metamodel can be written in the form of an additive model:

$$\tilde{f}(\mathbf{x}) = \sum_{m=1}^{M} a_m B_m(\mathbf{x}) \tag{1.26}$$

where the basis functions B_m contain in their own definition the design space subregion in which they are not zero:

$$B_m^{(q)}(\mathbf{x}) = \prod_{k=1}^{K_m} \left[s_{km} \cdot \left(x_{v(k,m)} - t_{km}\right)\right]_+^q \tag{1.27}$$

K_m is the number of splits resulting from the recursive partitioning of the design space, s_{km} is a function used to determine which half of the subregion has to be considered ($s_{km} = \pm 1$), t_{km} is the position on the axis defined by the input variable $x_{v(k,m)}$ where the splitting of the design space has taken place. The parameter q is the order of the spline, whereas the subscript $+$ indicates the positive part of the argument (Jin et al.

2001):

$$\left[s_{km}\cdot\left(x_{v(k,m)}-t_{km}\right)\right]_+^q=\begin{cases}\left[s_{km}\cdot\left(x_{v(k,m)}-t_{km}\right)\right]^q & if\ \ s_{km}\cdot\left(x_{v(k,m)}-t_{km}\right)>0\\ 0 \quad otherwise\end{cases} \tag{1.28}$$

A MARS metamodel can be recast in a sum of terms of increasing dimensionality, so transparency is guaranteed to some extent. The accuracy of the technique is critically dependent on the size of the training data set (Jin et al. 2001).

The *multipoint approximation method* (MAM) (Polynkin and Toropov 2011) is not a metamodelling technique per se, but it deserves to be mentioned as it is an optimisation technique which relies on metamodels defined iteratively on subregions of the design space. In MAM the system response is approximated by a linear combination of metamodels $\phi_i(\mathbf{a},\mathbf{x})$:

$$\tilde{f}(\mathbf{x})=\sum_{i=1}^{K}b_i\phi_i(\mathbf{a},\mathbf{x}) \tag{1.29}$$

where the size of the model bank K is set by the user. The vector $\mathbf{a}$ represents the parameters or coefficients to be tuned in each metamodel ϕ_i and b_i are the coefficients of the linear combination of metamodels ϕ_i. Polynkin et al. (2008) suggested using different techniques like RSM, kriging and RBFs to generate the single metamodels ϕ_i. Although in general Eq. (1.29) can be used as a global metamodel, better accuracy is reached in MAM considering it as a mid-range approximation. The design space is explored through the iterative definition of a trust region on which Eq. (1.29) is locally tuned. MAM has been successfully used in high dimensional problems, featuring up to 500 input variables (Polynkin et al. 2008).

Finally, the *moving least squares method* (MLSM) (Choi et al. 2001, Toropov et al. 2005) is an example of metamodelling technique which generates accurate global metamodels, although the gain in accuracy is paid by a reduced transparency, as the metamodels are not returned in explicit form. MLSM belongs to the class of local parametric approximations described by Friedman (1991), which are global parametric metamodels whose parameter values are computed by locally weighted least-squares fitting. MLSM models take the form (Friedman 1991):

$$\tilde{f}(\mathbf{x})=g(\mathbf{x}|\{\hat{a}_j(\mathbf{x})\}_1^p) \tag{1.30}$$

where g is a given parametric model having p parameters $\hat{a}_j$ to be tuned. Parameter values are obtained through locally weighted least-squares fitting:

$$\{\hat{a}_j(\mathbf{x})\}_1^p = argmin \sum_{i=1}^{N} w(\mathbf{x}, \mathbf{x}_i)[y(\mathbf{x}_i) - g(\mathbf{x}|\{\hat{a}_j(\mathbf{x})\}_1^p)]^2 \tag{1.31}$$

where N is the number of points in the building data set. The weight w associated to each sampling point $\mathbf{x}_i$ decays as the evaluation point $\mathbf{x}$ moves away from the building point $\mathbf{x}_i$. The typical behaviour is exponential (Toropov et al. 2005, Loweth et al. 2011):

$$w(\mathbf{x}, \mathbf{x}_i) = exp(-\theta r_i^2) \tag{1.32}$$

where θ is a (hyper)parameter (*closeness of fit* parameter) and r_i is a normalised distance from the building point $\mathbf{x}_i$ and the evaluation point $\mathbf{x}$. Acting on θ is possible to either increase smoothing or improve local accuracy.

As reported by Friedman (1991), the quality of the approximation is "determined more by the choice of w and to a lesser extent by the particular parametric function g used. Typically polynomials of 1st, 2nd or 3rd order are used for g. As the weights depend on the position of $\mathbf{x}$, also the coefficients of the MLSM basis functions depend on $\mathbf{x}$. As a result, MLSM does not return an explicit metamodel. As least-squares weight tuning is performed using a direct technique, MLSM metamodel training is relatively inexpensive. For this reason MLSM is successfully used in uncertainty analysis in conjunction with Monte Carlo approaches, as done for example in Toropov et al. (2005).

1.3 Introduction to genetic programming

Genetic programming (GP) (Koza 1992) is an iterative and population-based search technique that aims at finding "a suitable program in the space of all possible programs" (Langdon et al. 1999, pag. 167). A program can be defined as a set of instructions that solves a given problem or task (Zhang and Mühlenbein 1995, Barbosa and Bernardino 2011). As introduced in Section 1.1.2, the distinctive feature of genetic programming is the mechanism or strategy used to explore the search space: trial solutions are generated through genetically-inspired operations. The definition of optimality criteria allows GP to direct the search towards desirable or optimal programs.

The non-specific nature of genetic programming is its main strength. Genetic programming can be used to perform a directed search and optimise any entity that, once coded, can be processed and evaluated by GP. Thanks to its versatility, GP has been used in many fields of engineering for metamodelling, classification and design (Barbosa and Bernardino 2011). Table 1.2 provides a list of different applications in which genetic programming has been successfully employed. Further examples can be found in Barbosa and Bernardino (2011). The most common task genetic programming is used for is *symbolic regression* (Barbosa and Bernardino 2011), an expression commonly used by GP researchers to refer to the process of generating a metamodel through the recombination of mathematical operators considered in their symbolic form (Lew et al. 2006, Vladislavleva 2008).

TABLE 1.2: GP applications (Ps stands for program synthesis, Sr for symbolic regression)

Application	Authors	Aim
• Evolution of images and videos	Sims (1987 1993)	Ps
• Evolution of neural networks	Zhang and Mühlenbein (1995)	Ps
• Resolution of integral equations	Blickle (1996)	Sr
• Evolution of non-linear model of fluid flow in a coupled water tank system	Gray et al. (1996)	Sr
• Evolution of polyethylene rheological model and strain energy function of hyperelastic materials	Schoenauer et al. (1996)	Sr
• Evolution of a non-linear dynamic model of helicopter rotor speed controller and engine	Gray et al. (1997)	Sr
• Evolution of music	Johanson and Poli (1998)	Ps
• Evolution of digital circuits	Kalganova and Miller (1999)	Ps
	Iba and Terao (2000)	Ps
• Evolution of delay-time algorithms for anti-air missile proximity fuses	Nyongesa et al. (2001)	Sr
• Evolution of models of shear strength of reinforced concrete deep beams	Ashour et al. (2003)	Sr
• Evolution of temporal rules	Sætrom and Hetland (2003)	Sr
• Evolution of a dynamic model of a planar 10-bar truss	Shaw et al. (2004)	Ps
• Modelling of blown film properties	Kordon and Lue (2004)	Sr
• Rediscovery of Newton's law of gravity	Smits et al. (2005)	Sr
• Evolution of metamodels of the structural modulus and Poisson ratio of honeycomb structures, of natural frequency and mode shape in a 9 degree-of-freedom mass-spring system	Lew et al. (2006)	Sr
• Evolution of a differential equation solution	Koza (1992)	Sr
	Buchsbaum (2007)	Sr
• Evolution of a model for a diesel engine NO_x's emissions	Winkler et al. (2007)	Sr
• Evolution of models for the velocity to acceleration ratio in earthquakes	Kermani et al. (2009)	Sr
• Discovery of physical laws (Hamiltonian and Lagrangian)	Schmidt and Lipson (2009a)	Sr
• Evolution of solutions of implicit equations	Schmidt and Lipson (2010)	Sr

1.3.1 Genetic programming as a metamodelling technique

The application of genetic programming to metamodelling is motivated by the possibility to reformulate metamodelling as a search for the most accurate model in a set or space of mathematical models (Schmidt and Lipson 2010). As we have already seen, the expression "symbolic regression" is generally used to stress GP ability to search for metamodels' mathematical structure or function through the recombination of symbols representing mathematical operators. Not only is GP able to generate the mathematical structure of the best metamodel approximating the given training data, but it also finds the optimal values of its numerical coefficients (Smits and Kotanchek 2004, Lew et al. 2006, Schmidt and Lipson 2010). GP search in the space composed of all the mathematical expressions that can be built using functions and variables given by the user is iterative: new trial solutions are generated performing evolution-inspired operations on metamodels symbolic expressions, their quality evaluated on a given building data set and used to select a subset of metamodels from which a new generation of metamodels is spawned. At the end of the search a metamodel is symbolic form is returned.

Following the definitions provided in the previous sections, genetic programming can be defined as a non parametric, global and explicit metamodelling technique. Its main strengths are:

- GP is able to infer knowledge from input data (Schmidt and Lipson 2009a).
  GP can infer the mathematical structure as well as optimise the value of the parameters of the metamodel that best fits the building data set. Knowledge of the mathematical structure of the true underlying function relating system inputs and output is not required, except for the definition of the functions that GP can use to generate the models. Highly non-linear functions can be used to model highly non-linear responses. As a result, typical preliminary analysis usually performed in parametric regression (like screening or variable selection) is not needed.

- GP is able to select the most influential input variables from the set provided by the user. In this sense, GP can perform sensitivity analysis: a few examples where GP has been used for this purpose are provided in Nordin et al. (1999), Smits et al. (2005), Lew et al. (2006), Vladislavleva (2008).

- GP has "creative potential" (Schmidt and Lipson 2009a).
  GP is able to find innovative metamodels and more in general innovative solutions,

which might not have been anticipated by the user or designer (Chellapilla 1997, Soule and Foster 1998a, Schmidt and Lipson 2009a). GP creative potential is a reflection of GP ability to extensively explore the whole design space in an automated and directed way. In this sense GP is referred to as a machine learning technique, and it can be considered a form of artificial intelligence (Zhang and Mühlenbein 1995). The generation of innovative solutions is a common feature of evolutionary techniques (Beyer and Schwefel 2002, Kroo 2004).

- GP metamodels are returned in an explicit, symbolic form.
  The explicit text expression generated by GP ensures the maximum metamodel transparency and readability (Kordon and Lue 2004, Kermani et al. 2009). The analysis of GP generated metamodels may help extract some knowledge regarding the fundamental models describing the system under analysis (Barbosa and Bernardino 2011). Such feature is referred to as "human insight" by Smits and Kotanchek (2004), being particularly valuable as it helps increase user trust in the metamodel. The availability of a symbolic model also allows for symbolic post-processing operations, like symbolic differentiation (Schmidt and Lipson 2009a), and it may ease sensitivity analysis (Kordon and Lue 2004) and uncertainty analysis (Pierce et al. 2008). For example Smits et al. (2005) and Schmidt and Lipson (2009a) used GP expressions for the identification of *metavariables*, cluster of variables that can be used as transforms to generate low complexity models.

- GP metamodels are practically inexpensive to evaluate, being text expressions.

- GP are in general compact and easy to handle (Smits and Kotanchek 2004, Lew et al. 2006).
  GP metamodels can be easily saved as text files and deployed in this form to the final users. No extra tools or prior knowledge of genetic programming is required to use the metamodels once generated.

- GP is able to approximate noisy functions and provide smooth models (the smoothing ability depends on parameters set by the user).
  Noise may however increase symbolic regression computational cost (Zhang and Mühlenbein 1995, Sætrom and Hetland 2003, Schmidt and Lipson 2009a).

However, GP is still a population-based search algorithm, so a few drawbacks should be expected:

- GP is computationally demanding.

  Although some implementations can reduce dramatically the computational cost, GP is extremely computationally expensive (Nordin et al. 1999).

- GP metamodels are generally less accurate than metamodels generated using other techniques (RBFs, ANNs, MLSM) (Smits and Kotanchek 2004), although in some scenarios GP can outperform neural network metamodels, RSM based on polynomials (Lew et al. 2006) and, as shown in Chapter 7, Gaussian Process metamodels.

- the result of a GP search is generally not unique (Winkler et al. 2007).

  GP usually generates many metamodels of different shape and size with comparable accuracy. As a result, the selection of the best metamodel may require additional analysis, for example to assess metamodel accuracy and smoothness on new samples or robustness to input data uncertainty (as done for example in Pierce et al. (2008) for ANNs). Although time consuming, this process may still be useful to gain insight into the problem (Kroo 2004). Usually the smaller is the building data set, the larger is the set of metamodels of comparable accuracy.

- although explicit, in some cases GP metamodels do not allow for a clear physical interpretation (Schmidt and Lipson 2009a). Expression complexity may also hinder readability (Sims 1993).

In the following chapters GP strengths will be explained in more detail and different solutions to overcome GP weaknesses will be explored.

Chapter 2

Darwin, evolutionary algorithms and genetic programming

The reformulation of the concept of metamodelling as a search for the optimal metamodel in the space of metamodels motivates the application of genetic programming to regression. In order to understand how symbolic regression is performed by genetic programming it is however worth to separate for a while the evolutionary mechanisms used to perform a search from the specific purpose, in this case metamodelling, of the search itself. In this chapter it is shown how the evolutionary paradigm has been derived from evolutionary theories and genetics, the most common evolutionary algorithms are described and the issue of representation in genetic programming is introduced.

2.1 Stochastic and deterministic search techniques

An optimisation problem can be defined in general terms as (Haftka and Gürdal 1993):

$$find: \mathbf{x} \in D \subset \mathbb{R}^N \tag{2.1}$$

$$minimising: f(\mathbf{x}) \tag{2.2}$$

$$such\ that: g_j(\mathbf{x}) \geq 0 \quad j = 1, \ldots, n_g\ , \tag{2.3}$$

$$h_k(\mathbf{x}) = 0 \quad k = 1, \ldots, n_e \tag{2.4}$$

where D is the design space, $f(\mathbf{x})$ is the *objective* or *cost* function (Fogel 1994), g_j are the inequality constraints and h_k the equality constraints. The fact that a minimisation problem is assumed in (2.2) is not restrictive, as a maximisation problem can be reformulated as a minimisation one (Haftka and Gürdal 1993). A simple way to do it is to apply a monotonically decreasing function $w(y)$ to the function $y = f(\mathbf{x})$ that has to be maximised: the minima of $w(y)$ are the maxima of $f(\mathbf{x})$. Common function used for this purpose are the reciprocal ($w(y) = 1/y$) and the opposite of the natural logarithm ($w(y) = -ln(y)$) (Bishop 1995), although attention should be paid to the range of output values of the function to be transformed to avoid undefined operations.

The iterative search for a solution defined by some optimality criteria in a design space can be approached using two kinds of solvers: deterministic or stochastic. In either case, a set of trial solutions $\mathbf{x}_i$ are iteratively explored to minimise the predefined cost function, which represents the quality of the found trial solution.

Deterministic and stochastic methods differ in the way the trial solutions are generated (Fogel 1994). Deterministic methods rely on the cost function gradient or other sensitivity information evaluated at the location in the design space provided by the latest trial solution found by the solver. The position of the next trial solution is indicated by a vector indicating the direction in which the design space has to be explored and the distance from the previous trial solution. As a result, deterministic methods require that a distance can be defined in the mathematical space they are expected to explore, and that the cost function be regular enough that gradient can be computed (Quarteroni et al. 2000). For this reason, they may not be reliable in presence of noise. Furthermore, the solution found depends on the position of the initial guess provided by the user.

On the other hand, stochastic techniques only exploit the values that the cost function assumes on a set of trial solutions. Such information is used to generate the next set of trial solutions by means of a specific criterion: biology, evolution theory, social sciences, music, physics, metallurgy and astronomy have provided the inspiration for a variety of stochastic search algorithms (Lamberti and Pappalettere 2011). As the new solutions are not indicated by relative displacements with respect to the previous ones, in general stochastic methods can be used to explore spaces where a distance can not be defined. This property makes stochastic methods particularly versatile and allow for their application to a wide range of problems, among which the generation of metamodels. Secondly, they can be used with non-differentiable cost functions (Chetwynd et al. 2006). Thirdly, the design space exploration is less affected by the position of the initial guesses than in deterministic methods, and local solutions are more likely to be avoided (Pierce et al. 2006, Chetwynd et al. 2006) as a result of a more extensive design space exploration. For their robustness, stochastic techniques are also able to cope with data affected by measurement or computational noise. A wide range of stochastic methods is nowadays available: genetic algorithms (GA), particle swarm optimization (PSO), ant colony optimization (ACO), firefly algorithm (FA), bacteria foraging, harmony search (HS), big bang-big crunch (BB-BC), hunting search (HuS), simulated annealing (SA), charged system search (CSS), etc (Lamberti and Pappalettere 2011). In general it is not possible to identify a priori the best stochastic method, as their performance is generally dependent on the search or optimisation problem under study (Lamberti and Pappalettere 2011).

In conclusion, deterministic algorithms are fast, efficient and accurate as long as some regularity conditions are assumed on the cost function to be explored (Fogel 1994). Such conditions are not easily satisfied by cost functions built from real-life engineering applications, which may feature multiple optima (multimodal cost function), may be affected by noise and may also be non differentiable, if discontinous. Stochastic methods are particularly effective in this scenario, and are also appreciated for their ability to return a set of "good" solutions or designs rather than a single one (for example for trade-off analysis using Pareto front) and in multi-objective searches (Kroo 2004).

2.2 Neo-Darwinian paradigm

Evolutionary algorithms represent a class of stochastic methods, which genetic programming belongs to. Their main inspiration has come from evolution theories, biology and genetics. Darwin's and Wallace's evolution theories (Kutschera 2003), as well as Lamarck's, have particularly influenced the first researchers on evolutionary computation (Fogel 1994), and the DNA discovery from F. Crick and J. D. Watson in 1953 and its role in organism reproduction have provided a mechanical model to imitate for the generation of new trial solutions.

In the following a brief description of Lamarck's, Darwin's and Wallace's theories of evolution is given and the contribution of modern genetics to evolutionary algorithms implementation discussed. Not to hurt the reader's sensitivity, the evolution theories here presented are meant to be merely a source of inspiration for computational methods: this thesis neither confute nor support the validity of such theories.

2.2.1 Lamarck, Darwin and Wallace

According to Darwin's and Wallace's theory of evolution, species are able to adapt under the pressure of the environment in which they live. Such pressure is typically the result of the finite quantity of resources available to the species, food being the typical example. Adaptation, or evolution, is made possible by the exchange of biological information (coded in what would be later called "DNA") from parents to offspring through the reproduction process, which may also introduce variations in such information (Kutschera 2003). Whenever a variation in the biological information results in an organism that is fitter than others to live in a particular environment, such organism is more likely to reach the reproduction phase and, as a result, it is more likely to spread its biological characteristics to its offspring and so to future generations. As the environment itself is likely to change, living organisms are the specific results of an endless process of adaptation to a constantly changing environment, where the prize is the survival. In this sense Darwin's theory is also called the "survival of the fittest", although this expression was introduced by the philosopher H. Spencer (Kutschera 2003).

Darwin's and Wallace's theories of evolution have eventually led to the formulation of neo-Darwinism, a theory of evolution according to which the acquisition of new biological features is entirely ascribed to genetic mechanisms occurring during reproduction. As a matter of fact Darwin did not entirely exclude the possibility that organisms can acquire some characteristics during their lives and pass them to the offspring, idea that is the core of Lamarck's theory of evolution (Banzhaf et al. 1998, Kutschera 2003, eco 2009). This possibility, referred to as "the inheritance of acquired characters"[1], was instead excluded by Wallace (Kutschera 2003).

2.2.2 Neo-Darwinism

Neo-Darwinism is the term used to refer to the most widely accepted collection of evolutionary theories (Fogel 1994). Its main tenets derive from Darwin's and Wallace's theory of evolution (Fogel 1994):

[1]The classic example of the giraffe is provided in Wallace (1858): "... The powerful retractile talons of the falcon- and the cat-tribes have not been produced or increased by the volition of those animals; but among the different varieties which occurred in the earlier and less highly organized forms of these groups, those always survived longest which had the greatest facilities for seizing their prey. Neither did the giraffe acquire its long neck by desiring to reach the foliage of the more lofty shrubs, and constantly stretching its neck for the purpose, but because any varieties which occurred among its antitypes with a longer neck than usual at once secured a fresh range of pasture over the same ground as their shorter-necked companions, and on the first scarcity of food were thereby enabled to outlive them. ..."

1. NATURAL SELECTION
2. REPRODUCTION
3. VARIATION

Natural selection is the “pressure” or “force” that is responsible for the death or the survival of an organism. It is generally determined by the fact that resources, like food or a mating partner, are finite in nature so the organism able to access them is more likely to survive and reproduce, whereas the one that cannot reach them will find more difficulties in transferring its biological information through mating. Where many organisms have the same access to resources, they are likely to engage in a fight to prevail. As a result, natural selection is closely linked to competition among individuals.

The winners or survivors to natural selection are likely to mate and transmit their successful biological information to their offspring, which may have an advantage with respect to other organisms due to the inheritance of successful traits from the parents. The possibility of an organism to generate individuals similar to itself is called *reproduction*, and it is vital as it is the only way to spread successful biological information.

The biological information, which determines the structure, the functions and indirectly the behaviour of the organism itself, is expected to be affected by casual changes during reproduction. As a result, offspring’s biological information may be slightly different from its parents’. Offspring’s new traits are the effect of such *variation* in the biological information passed from parents to offspring. A priori this change may be either good or bad for the offspring: only natural selection and the struggle for resources will determine if variation is beneficial to the individual.

2.2.2.1 Genotype and phenotype

Darwin and Wallace observed the biological variation from parents to offspring and recognised its importance in explaining the adaptation of the species to the environment. Yet, they could not figure out the inner cause of such variation (Kutschera 2003). The development of genetics, from Mendel’s experiments to DNA discovery in 1953 by J. D. Watson and F. Creek, provided the insight that was missing to link the three principles of natural selection, reproduction and variation. The synthesis of Darwin’s and Wallace’s theories with genetics has led to the birth of Neo-Darwinism.

Geneticists discovered that the set of physical characteristics or observable properties of any organism, called *phenotype* or *phenome*, are coded into a chain of molecules called deoxyribonucleic acid (DNA) stored in each cell of the organism. The DNA of an organism is also referred to as the *genotype*, *genome* or genetic code of that organism (Banzhaf et al. 1998). To our purposes it is sufficient to know that the genetic code contains all the "information" required for an organism to build its own body and to regulate all the functions indispensable to its life and to its interaction with the external environment (Sims 1993). As the interaction of an organism with the environment critically depends on its phenotype, and being this a reflection of its genotype, natural selection can be considered as a selector of the fittest genotypes.

It is worth noting that the experiences and learning processes an organism may go through during its life have no effect on its genetic code and so they cannot be passed to offspring according to Neo-Darwinism, as that would imply the acceptance of "the inheritance of acquired characters" supported by Lamarck. In other words, variations in the genotype produce a different phenotype, but changes in the phenotype (due to training, wounds or experience for example) cannot affect the genotype.

2.2.2.2 Crossover and mutation

Reproduction is a key element of Neo-Darwinism as it is the main cause of phenotypical variation in species. Such variation is the result of the recombination and alteration that parents' genotypes undergo to generate the offspring's genotypes. The mechanisms through which parents' DNA is copied and transmitted to offspring are extremely complex and their detailed description is beyond the aims of this work. A simplified description is however useful to understand the way evolutionary algorithms generate trial solutions.

In "sexual" reproduction offspring's genetic codes are generated blending together parents' DNAs. The mixing mechanism has been named *crossover* by geneticists, as blocks of DNAs are exchanged between the parents' DNAs to generate offspring's DNAs. The process involves two stages: initially the correct coupling between the two parents' splitted DNA chains is established putting similar parts of the two DNAs together, then, once the match is established, a few blocks are swapped between the splitted DNA chains. This mechanism is called *homologous crossover* or *homologous recombination*. In Fig. 2.1 the process is schematically illustrated: the portions of parents' splitted DNA chains are first aligned in correspondence of the black circle, which represents the same DNA base. The DNA portions following the matching point, represented by the other symbols, are then swapped.

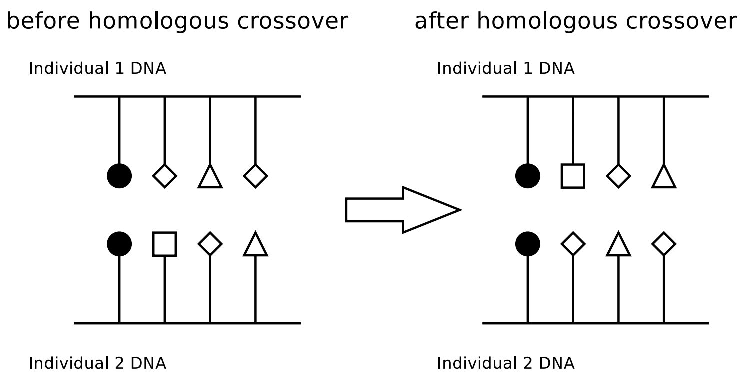


FIGURE 2.1: Homologous crossover

Homologous crossover guarantees both stability and variability of the genotype. The recombination of parents' genotypes is mild enough to ensure that the offspring have feasible and successful genetic codes, yet it is extensive enough to introduce some new features in offspring's phenotypes that may (or may not) be beneficial for the fight for survival.

There is another mechanism that is recognised to cause genotype variation, called *mutation*. Mutation is a random variation of a part of the DNA and spontaneously occurs during reproduction, although it can be caused by external factors (contact with mutagens, for example). Mutation can have either a positive or a destructive effect on the organism. In Fig. 2.2 it is schematically shown how a portion of DNA chain is transformed by mutation into another, maybe new, sequence of DNA (the triangle, third symbol from the left, is changed into a circle).

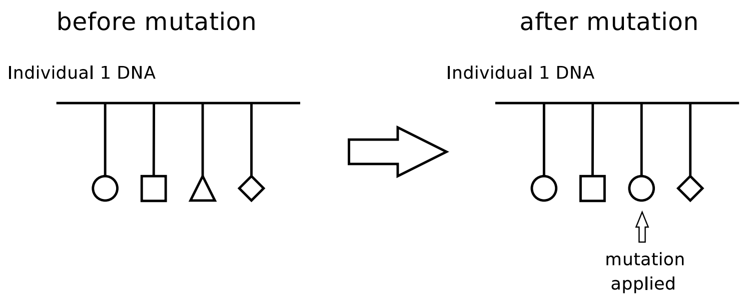


FIGURE 2.2: Mutation

2.2.2.3 Pleiotropy and polygeny

Establishing a direct correspondence between single portions of DNA, which for simplicity will be called *genes*, and specific phenotypical traits of an organism is particularly difficult.

Borrowing the terminology from set theory, DNA portions are not linked to phenotypical traits by a bijective function.

Geneticists use the terms “pleiotropy” and “polygeny” to describe the complex relation between genotype and phenotype. As reported by Fogel (1994, p. 3), “. . . Pleiotropy is the effect that a single gene may simultaneously affect several phenotypic traits. Polygeny is the effect that a single phenotypic characteristic may be determined by the simultaneous interaction of many genes”. Due to the complicate mapping between genotype and phenotype, “the results of genetic variations are generally unpredictable due to the universal effect of pleiotropy and polygeny” (Fogel 1994, p. 3).

Pleiotropy and polygeny are important issues also in evolutionary algorithms, which rely on the codification of a solution through a genotype.

2.3 Evolution as a search mechanism

The evolution mechanism depicted by Neo-Darwinism can be easily interpreted as a strategy to explore new organism’s physical features without seriously compromising the possibility to exploit existing successful features, as genotype variation is gradual and multiple copies of the set of so-far successful genes are in any case retained by the population. So evolution as theorised by Neo-Darwinism can be considered in a sense as a search of the genotype corresponding to an optimal phenotype, the one which ensures the survival and the best adaptation of a species to a certain environment. Quoting Fogel (1994), “Darwinian evolution is intrinsically a robust search and optimization mechanism” (p. 3), and also “evolution is an obvious optimizing problem-solving process” (p. 4).

Evolutionary algorithms or evolutionary computing (Winkler et al. 2007) are the result of the attempts to transpose the evolutionary and genetic mechanisms implied by Neo-Darwinism into an algorithm that can be used to perform a directed search or exploration of a user-defined space. The basic components shared by all evolutionary algorithms are (Sims 1993, Michalewicz 1996, Eiben and Schoenauer 2002):

1. a genotype-phenotype mapping, also called *representation*. Evolutionary algorithms handle and refine *programs*, be they vectors, mathematical equations or other entities. A way to represent such programs is key for the artificial evolution process to be able to evolve them. The chosen representation has to allow for *evolvability*, as it “must give a non-vanishing likelihood that variation produces performances

improvement" (Altenberg 1994). "Evolvability" is not always easy to guarantee (Affenzeller and Wagner 2004), as this property should be ensured by the user through appropriate coding and representation.

2. a population of genotypes, also called individuals or chromosomes. Evolutionary algorithms require several individuals to be able to search effectively for the optimal genotype. As reported in Zhang and Mühlenbein (1995), "the evolutionary approach differs from most other search techniques in that it makes a parallel search simultaneously involving hundreds or thousands of points in the search space". The generation of an initial population of individuals from which the evolution can start is then required.

3. a function that assumes the role of the selection pressure, according to the Darwinian concept of environment. Such function is called *fitness function*. Usually the fitness value associated to a genotype is computed by a computer according to specific criteria, but there are examples of evolutionary algorithms used in art where fitness value is evaluated by the user (Sims 1993, Johanson and Poli 1998).

4. a selection process which probabilistically selects the genotypes having the best fitness (Fogel 1994). Darwinian selection relies on *heritability*, or the assumption that "better individuals are more likely to produce better offspring" (Blickle and Thiele 1997, p. 361). Quoting Blickle and Thiele (1997, p. 361), "...without heritability, selection of better individuals makes no sense".

5. a set of "genetic" operations that alter parents' genotypes to produce offspring's genotypes.

6. a set of numerical parameters that defines population size, the maximum number of generations and a few other issues related to the practical implementation of the algorithm (Michalewicz 1996).

The basic set of operations that are common to all evolutionary algorithms are shown in Table 2.1 (Koza 1992, Banzhaf et al. 1998, Brameier and Banzhaf 2007, Poli et al. 2008). The algorithm starts with the random generation of the initial population of genotypes (1), from which an iterative process develops. At each iteration, also called *generation*, each individual of the current population undergoes fitness evaluation (2) (it may require the execution of the program), selection (3), reproduction (crossover and

TABLE 2.1: Evolutionary algorithm

1. Initialization of the population
2. Fitness evaluation (may require execution)
3. Selection
4. Reproduction:
   - ▶ replication
   - ▶ crossover
   - ▶ mutation
5. Termination criterion check. If met go to 6 otherwise go to 2.
6. Stop (maximum number of generations reached or termination criterion met)

mutation) (4). The algorithm stops when a particular termination criterion is met (5), i.e a particularly fit individual is found, or the maximum number of generations is reached (6).

An important feature of the evolutionary paradigm as described in Table 2.1, so far gone unnoticed, is its versatility. Deterministic (gradient-based) techniques can be used to search for solutions to the optimisation problem defined in (2.1-2.2-2.3-2.4) as long as a distance in the design space D can be defined. There are many optimisation problems in engineering where however this is not possible. For example, how to define the distance between two electric circuits? And between two algorithms? The evolutionary paradigm provides a way to explore a user-defined space through variation of a "code" representing a trial solution. As a result, it can be used to optimise every entity or object that can be given a representation and evaluated so that the conditions previously listed are satisfied. Table 1.2 gives an example of the variety of problems that can be solved using an evolution-based algorithm.

2.3.1 Classification of evolutionary algorithms

Evolutionary algorithms are usually classified in four major categories (Fogel 1994, Banzhaf 1994, Banzhaf et al. 1996, Affenzeller and Wagner 2004, Kroo 2004, Collet and Schoenauer 2004)): evolutionary programming (EP), evolution strategies (ES), genetic algorithms (GA) and genetic programming (GP). The evolutionary algorithms taxonomy is represented in Fig. 2.3.

Evolutionary programming (EP) was originally presented by L. J. Fogel in his doctoral dissertation at UCLA (USA) in the 1960s (Fogel 1964, Bull 2008). Fogel's aim was to generate artificial intelligence. Evolutionary programming was conceived as a set of

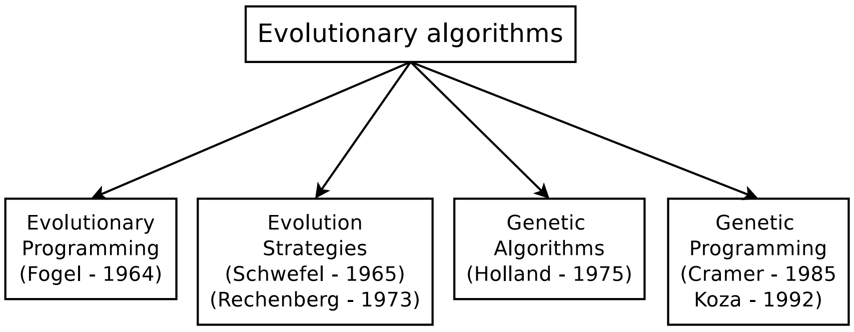

Evolutionary algorithms
Evolutionary Programming (Fogel - 1964)
Evolution Strategies (Schwefel - 1965) (Rechenberg - 1973)
Genetic Algorithms (Holland - 1975)
Genetic Programming (Cramer - 1985 Koza - 1992)

FIGURE 2.3: Classification of evolutionary algorithms

operations that can evolve mathematical models called *finite state machines* for the prediction of symbols of a sequence. A variable-length genotype was chosen to represent the finite state machines and mutation was used as the "principal search mechanism used to create one or more offspring per parent" (Bull 2008, p. 1).

Evolution strategies (ES) were developed by H.-P. Schwefel and I. Rechenberg, collaborating in Germany in the 1960s and 1970s (Fogel 1994, Beyer and Schwefel 2002). Evolutionary strategies were first conceived as a "set of rules for the automatic design and analysis of consecutive experiments with stepwise variable adjustments driving a suitably flexible object/system into its optimal state in spite of environmental noise" (Beyer and Schwefel 2002, p. 4-5) and successfully applied to the design of a 3D convergent-divergent hot water flashing nozzle (Beyer and Schwefel 2002). Only later this set of rules were formally codified to the constant-length representation known today and applied to the optimisation of real-valued functions (Beyer and Schwefel 2002). Originally, a single offspring was generated from a single parent through a mutation operator and both genotypes were left in the population to compete for survival. Later, new strategies were introduced to generate one or more offspring from one or more parents.

Genetic algorithms (GA) appearance dates back to the 1970s and is attributed to the work of J.H. Holland at the University of Michigan (US) (Holland 1975). Genetic algorithms have been used since mainly for optimisation of real-valued functions. The representation chosen by Holland to represent a point in a design space is a string of bits of costant length (Holland 1975 1992, Koza 1992, Van Belle and Ackley 2002), usually called *GA chromosome*. Crossover is traditionally the main GA operator for the generation of trial solutions (Holland 1992, Chellapilla 1997, Luke and Spector 1998, Van Belle and Ackley 2002). An example of parents' chromosomes recombination produced by a crossover operator (two point crossover) is shown in Fig. 2.4.

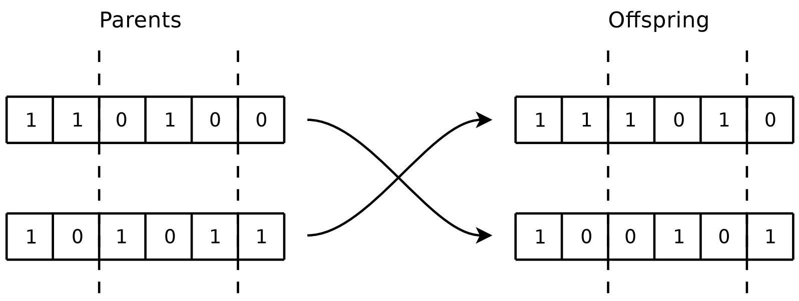


FIGURE 2.4: Two-point crossover in genetic algorithms

Due to the inherent "granularity" of the chromosomes, GA is better at handling integer variables. If GA is applied to optimisation problems featuring continuos input variables, the maximum accuracy of the solution found by GA is necessarily affected by the design space discretisation imposed by the binary encoding used to define chromosomes (Kroo 2004).

Genetic programming (GP) was introduced by Cramer (1985) and Koza (1992). Its typical uses range from regression of mathematical models, solution of differential equations, integral calculation to pattern recognition, electric circuits and antennas synthesis. Differently from ES and GA, Genetic programming relies on a variable length chromosome (Koza 1992, Zhang and Mühlenbein 1995, Nordin et al. 1999, Van Belle and Ackley 2002), typically but not exclusively tree-shaped, whose size can be increased as a result of genetic modifications. Crossover is the main search operator according to Koza (1992), although mutation is often used to promote variability in the population of trial solutions.

GP versatility can be ascribed to the variable length representation chosen to code GP individuals, which allows to increase the size, the shape and the complexity of the trial solutions (Sims 1993, Zhang and Mühlenbein 1995, Johanson and Poli 1998).

There are clear historical reasons behind the classification of the different dialects of evolutionary computing presented above. Evolutionary programming, genetic algorithms and genetic programming were given birth in the US, whereas evolutionary strategies were developed in Germany. A few of them were initially conceived as strategies to address specific tasks, like evolutionary programming and evolutionary strategies. In their original formulations, evolutionary algorithms can also be distinguished according to the representation chosen, tree-shaped as in GP or linear as in GA, whether they have a variable or fixed length representation, and according to the mechanism used to generate trial solutions. Evolutionary programming and evolutionary strategies indeed rely on mutation for offspring generation, as opposed to genetic algorithms and genetic programming that exploit mainly crossover (Koza 1992, Banzhaf et al. 1996, Luke and Spector 1998).

The previous classification has however faded with time, probably as a result of the ever increasing use of evolutionary algorithms in engineering and of the attempts to improve their performance through the addition of novel features. Consequently, the boundaries among EP, ES, GA and GP are nowadays blurred: for example, the evolutionary programming implementation used by Chellapilla (1997) can be described as a conventional GP implementation in which only mutation operators are used. The matricial representation chosen by Kalganova and Miller (1999) to evolve electric circuits is far more complex than the string of bits assumed by Holland (1975), so that recognising in it a GA defining feature is hardly possible. A few researchers agree that some original differences among evolutionary algorithms have disappeared: Barbosa and Bernardino (2011) for example classify GP as a GA used for the automatic generation of computer programs, in particular of metamodels. Collet and Schoenauer (2004) support a more radical point of view, stating that “all evolutionary algorithms are born equal if the user is provided with enough parameters to tune” (Collet and Schoenauer 2004, p. 214).

In conclusion, it seems that a comprehensive way to classify an evolutionary algorithm should take into account the task the algorithm is used for, the selected representation and the genetic operators used. In the following chapter a detailed analysis of the genetic programming paradigm based on syntax tree representation will be presented.

Chapter 3

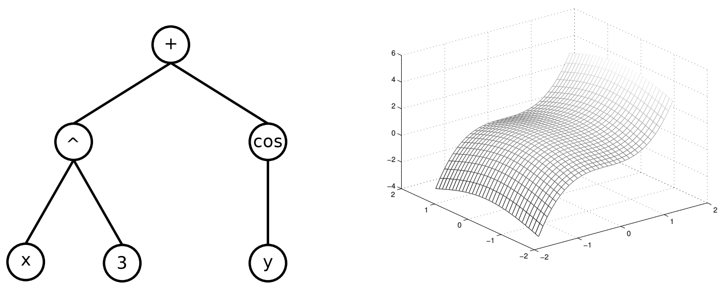
GP for symbolic regression tasks

In Chapter 1 genetic programming was generally introduced as a technique able to generate metamodels through the recombination of mathematical operators considered in their symbolic form (*symbolic regression*). Chapter 2 provided the theoretical background needed to understand how evolutionary algorithms perform a search in a user defined space. In this chapter further details will be given on each stage of the genetic programming algorithm applied to symbolic regression. Although the description will mainly focus on the tree-based GP implementation described by Koza (1992), at the end of the chapter alternative GP representations will be briefly reviewed.

3.1 GP basic implementation

One of the requirements for the application of genetic programming to symbolic regression is the identification of a metamodel representation. The first and most common genotype representation in genetic programming is the *syntax tree*, or simply *tree*, introduced by Cramer (1985) and Koza (1992) and still widely used (Sims 1993, Soule et al. 1996, De Jong and Pollack 2003, Smits and Kotanchek 2004, Winkler et al. 2007, Vladislavleva 2008, Barbosa and Bernardino 2011). Mathematically a syntax tree is a directed graph (Lew et al. 2006), composed of links and nodes. Each node is connected to other nodes through links, which represent a dependency of the operation contained in the node on arguments contained in the connected nodes. The dependencies between connected nodes are unambiguously defined through a hierarchy based on node position: "offspring" nodes placed below a "parent" node are all arguments of the parent node, so in order to evaluate the operation contained in the parent node all the operations contained in the offspring have to be executed first. A syntax tree is defined by all the nodes that stem out of a *root* node, which is placed above all other nodes and is not argument of any other node. In syntax trees recursion is excluded by the impossibility to define a parent node as argument of a node at a lower level. Links to nodes not immediately above or below the current node are also forbidden.

Although syntax trees can be used to represent different entities, they are particularly suited for describing mathematical functions. A mathematical expression can be converted in a syntax tree (genotype) associating a node to each indivisible mathematical entity the expression is made of and using links to define the dependencies between functions and corresponding arguments. The term *arity* defines the number of offspring nodes a parent node can have (Lew et al. 2006, Vladislavleva 2008): mathematical functions used for metamodelling purposes mostly have one or two arguments, so they are represented by unary (arity 1) or binary (arity 2) *functional nodes*. Variables and constants instead are represented with *terminal nodes* (arity 0). For example, the mathematical expression $x^3 + cos(y)$, whose mathematical behaviour (phenotype) is shown in Fig. 3.1B, can be represented through the syntax tree in Fig. 3.1A. The evolutionary paradigm acts



(A) *Genotype* (B) *Phenotype*

FIGURE 3.1: Syntax tree (genotype) of $x^3 + cos(y)$ and corresponding phenotype for $x \in [-\pi/2, \pi/2]$ and $y \in [-\pi/2, \pi/2]$

is able to improve metamodel accuracy modifying through genetic operations the size and shape of the syntax trees as well as the content of the trees' nodes.

The syntax tree is hystorically the original representation used in genetic programming. For its simplicity, it was the representation selected for the development of a new GP implementation, which is the outcome of the research activity presented in this work and will be described in Chapter 5. Different representations have however been developed to extend tree-based genetic programming capability and improve its performances

(Kantschik and Banzhaf 2001b, Withall et al. 2009). The two major alternatives to tree-based GP are linear GP and graph-based GP (Langdon et al. 1999, Kantschik and Banzhaf 2001b, Barbosa and Bernardino 2011). Although these representations can be used for symbolic regression, their intrinsic nature make them more suitable for other tasks, like algorithm induction and image recognition, as it will be shown in Section 3.3. The following sections aim instead at providing an insight into the basic components of tree-based GP paradigm.

3.1.1 Tree-based GP: definitions

A formal definition of a few parameters relating syntax tree size and shape is necessary for the application of genetic operators and it is also important to indirectly control the quality of the metamodels during the evolution. In this dissertation the following conventions are adopted:

1. the *size* of a syntax tree is the number of nodes the tree is composed of;
2. each node in a tree is identified by a number in the interval ranging from 1 to the size of the tree. The *root* node is identified by number 1. The remaining nodes have integer identification numbers that increase descending towards the bottom of the tree, assuming that whenever a binary node is encountered the left child node and its children come first in the counting, as shown in Fig. 3.2;
3. the *depth* of a node is the length of the shortest branch linking the current node to the root node. The length of the branch is measured in number of nodes, excluding the current node and including the root node. The root node has depth 0;
4. the *depth* of a syntax tree is the maximum depth among the depths of the nodes composing the tree.

Fig. 3.2 will help with the previous definitions: node number 4 has depth 2, as it is two "steps" apart from the root node (first step to node 2, second step to node 1). The depth of the trees is 2. Tree depth and tree size are precious parameters as they are commonly used to avoid excessive growth of an individual. More details on their use will be given in Section 4.5, Chapter 4.

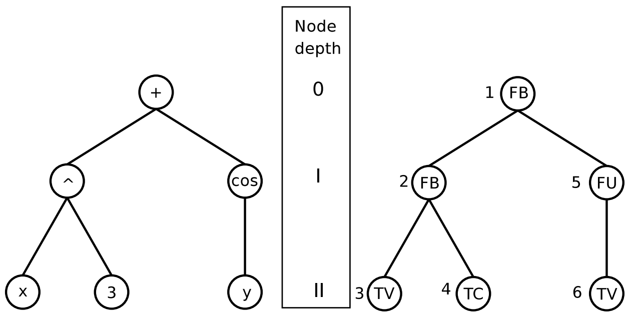


FIGURE 3.2: Depth and node identification in a syntax tree. On the left the syntax tree (genotype) of $x^3 + cos(y)$. On the right a generalization of the tree: FB is used to represent a binary node, FU a unary node, TV a terminal node containing a variable, TC a terminal node containing a constant. The number close to each node of the tree on the right is the node identification number. The depth of each node is indicated in the column between the pictures.

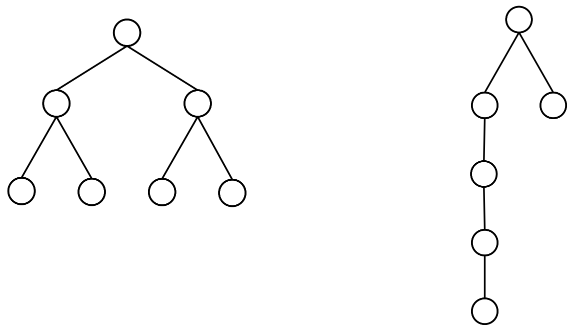
3.1.2 Initialisation of the population

Population initialisation is the process through which a certain number of syntax trees (alsos referred to as *individuals*) is randomly generated combining a set of symbols (Lew et al. 2006, Poli et al. 2008), called *primitives*, provided by the user. As seen in the previous section, these symbols either represent mathematical functions (*functional set* F), which require a positive number of arguments (arity>0), or variables and numerical constants (*terminal set* T). In the original GP formulation, Koza (1992) introduced the use of the *ephemeral random constant*, $\Re$, a symbol that does not correspond to any specific real value but is replaced by one or more constants randomly generated in a given range at the beginning of the evolution. In case $\Re$ is not used and specific constants are not included in the terminal set, GP can still generate numerical coefficients by clustering variables in neutral arithmetic operations (for example x/x produces 1 for $x \neq 0$).

The initialisation process is critical to the success of the evolution. The convergence rate, or number of generations needed to produce an acceptable individual, usually improves if as much different combinations of primitives as possible are generated, resulting from the increase in the likelihood of finding right from the first generation a set of "good" individuals, or syntax trees corresponding to metamodels with high accuracy and generalisation ability. According to the experiments performed by Langdon (2000)[1], GP populations appear to retain a long-term memory of the way they are initialised: Langdon (2000, p. 110) showed that the average tree size to depth ratio in the population tends to keep constant and equal to the value of this ratio set during the initialisation. The size to

[1]Langdon used a standard tree-based GP, with standard subtree crossover, no mutation, tournament selection, no elitism (Langdon 2000, p. 111).

depth ratio is a paremeter that defines the shape of a tree, as shown in Fig. 3.3: "bushy" trees have larger size to depth ratios than "slender" trees (Langdon 2000).



(A) *Bushy tree (size to depth ratio = 3.5)* (B) *Slender tree (size to depth ratio = 1.5)*

FIGURE 3.3: Different tree shapes resulting from different size to depth ratios

Although Langdon (2000) focused mainly on the exploration of various syntax tree shapes, his results confirm the importance of search space sampling on the evolution success. Many techniques have been developed to ensure variability in the initial population. The most common are *no limit* method, *full* method, *grow* method and *ramped (half-and-half)* method.

The *no limit* method (Koza 1992) does not assume any limit on the maximum depth of the trees. The algorithm that implements this method chooses recursively a random kind of node (terminal or functional) and assigns it a random primitive of the same kind, ensuring also the syntactical correctness of the links between nodes. When all the branches of a tree are assigned a terminal node the process ends and a legal tree is returned. Trees generated by this method are likely to be extremely different in size and shape. However, there is absolutely no control on the maximum size and depth of the tree, which could be theoretically infinite.

The *full* method (Koza 1992, Barbosa and Bernardino 2011) generates trees of a depth defined by the user. To assign the nodes, the algorithm selects randomly functional nodes (binary or unary) unless the depth of the node to be generated is equal to the user-defined maximum depth. In this case the node is forced to be a terminal node and its type (variable or constant) is chosen randomly in the terminal set. Trees generated by full method may appear quite "bushy" and dense respect to the trees obtained with other methods.

The *grow* method (Koza 1992, Barbosa and Bernardino 2011) generates on average more slender and stretched trees with respect to the full method, as the branches' lengths are not forced to reach the tree's maximum depth defined by the user. The *grow* algorithm chooses randomly a node among the *whole* set of primitives (joined functional and terminal set), being forced to pick up a terminal node only if the corresponding node's depth is equal to the maximum depth set by the user. As a result, terminal nodes might be chosen even in the early stages of the generation process, giving birth to shorter branches and then less dense trees.

The *ramped* method (Koza 1992, Montana 1995, Whigham 1995, Luke and Spector 1997, Langdon 1998, Langdon et al. 1999, Collet et al. 2000, Topchy and Punch 2001, Van Belle and Ackley 2002, Poli et al. 2008), was developed to increase tree shape variability by the combined use of the previous two methods (grow and full). In its original form, called *ramped half-and-half* (Koza 1992), the method generates half of the population using the *full* method and the other half using the *grow* one, imposing a range of depth limits for each method. Variations have been introduced to modify the relative percentage of the population initialised with each method.

The previous methods are typically used to generate a set of individuals of relatively small size and depth, as the size of the solution is in general not known a priori and the evolution process can anyway adapt trees' size and shape. Against this common practice, Poli and Langdon (1998) support the idea that initial populations should contain individuals having size or depth similar to the ones observed in the end-of-run solutions. The same attitude is shared by (Chellapilla 1997) (see for example the high size limit used for initialisation in Chellapilla (1997, p. 211)), and it is the result of an alternative approach to evolutionary search. According to these researchers genetic operators perform a sort of "interpolation" between parents to generate children, instead of progressively building up a pool of solutions out of more and more specific subtrees refined by the evolution process.

Although grow, full and ramped methods are the most common in GP community, many researchers have put forward alternative solutions. Langdon (2000) criticised the ramped half-and-half method for a bias towards the generation of short and bushy (full) trees for a given size limit. He proposed the *ramped uniform initialisation* method as a more balanced alternative. Although interesting, the method is considerably more complex than the above mentioned and for this reason is not here explored further. In conclusion, it is also important to remind that initialisation is not necessarily random, and any prior knowledge regarding the shape or primitives of the desired solution can be exploited at this initialisation stage. For example, high-quality metamodels generated by other metamodelling techniques can be embedded in the initial population, hopefully providing important genetic material that can boost the search for an even better metamodel (Poli et al. 2008).

3.1.3 Fitness function

The fitness function has the role of creating "environmental pressure" into the artificial world populated by syntax trees. It is a function that assesses the quality of each individual phenotype and assigns it a score, called *fitness value*. It is assumed that a smaller fitness value corresponds to more accurate metamodels.

Historically, the first metric used to measure fitness value was the sum of the absolute errors (Koza 1992) (examples may be found also in Nordin et al. (1996) and Smits and Kotanchek (2004)):

$$F(k,t) = \sum_{j=1}^{m} |f(\mathbf{x}_j) - \tilde{f}_{k,t}(\mathbf{x}_j)| \tag{3.1}$$

where:

- $F(k,t)$ is the fitness value of individual k at generation t ($\tilde{f}_{k,t}$);
- m is the number of fitness cases in the training data set;
- $f(\mathbf{x}_j)$ is the actual response of the system at the fitness case $\mathbf{x}_j$;
- $\tilde{f}_{k,t}(\mathbf{x}_j)$ is the value returned by individual k at generation t for the fitness case $\mathbf{x}_j$.

The metric defined by Eq. (3.1), based on 1-norm (Quarteroni et al. 2000), has also been used in its averaged form (average of the sum of the absolute errors) (Langdon et al. 1999). Many other metrics have been introduced, based on 2-norm (root mean square error or sum of the squared errors (Banzhaf 1994, Chellapilla 1997)), p-norm, inf-norm (p-norm with p=$\inf$ is the magnitude of the maximum error on the training data set) or hit (error defined only beyond a given threshold) (Smits and Kotanchek 2004). One of the most common fitness functions is anyway the root mean square error (RMSE), or the squared averaged 2-norm:

$$F(k,t) = \sqrt{\frac{1}{m} \sum_{j=1}^{m} |f(\mathbf{x}_j) - \tilde{f}_{k,t}(\mathbf{x}_j)|^2} \tag{3.2}$$

where the symbols used have the same meaning indicated for Eq. (3.1). Regardless the norm chosen to define the fitness function, dividing the norm by the number of fitness cases is generally recommended in case the number of fitness cases fed into the GP implementation is increased throughout the evolution, to avoid sudden increments in fitness value due only to the addition of new fitness cases.

Koza (1992) used the expression *raw fitness* to refer to the fitness functions defined as sum of errors. He specifically referred to the sum of absolute errors (Eq. (3.1)), but here the term will be used to refer to all metrics mentioned so far. He used *raw fitness* to formulate additional and more complex fitness functions, called *standardised fitness*, *adjusted fitness* and *normalised fitness*: as these formulations will not be used in the following, they are not expored further. More details can be found in Koza (1992).

3.1.4 Selection

The selection process identifies the fittest individuals in the population and grants them the privilege of passing their genotypes to the new generations through the following reproduction stage. The fitness function defined in the previous section determines the evolutionary success of each individual, according to the *heritability* hypothesis introduced in Section 2.3, Chapter 2: better fitness values determine higher probability to be selected, whereas worse fitness values reduce such probability. As the probabilistic nature of the selection process is another assumption of the evolutionary paradigm, it is important to remind that even though selection encourages the evolution of the fittest individuals, it does not prevent low fitness individuals from spreading their genotype through reproduction.

Fitness proportionate selection (Koza 1992, Holland 1992, Whigham 1995, Nordin et al. 1996, Lew et al. 2006) is the original selection method proposed by J. R. Koza. This selection method assigns to each individual a probability to be selected that is proportional to its fitness value. The selection process can be easily implemented using the normalised fitness $n_{k,t}$ defined in Koza (1992): the fitness range $[0, 1]$ is split in a number of subintervals equal to the number of individuals and having width $n_{k,t}$. The subinterval containing a number randomly generated in the range $[0, 1]$ determines the selected individual. This process is often compared to the spin of a roulette wheel having slices of different angles, proportional to the individuals' fitness values and so it can also be found in literature as *roulette wheel selection* (Fogel 1994, Ferreira 2001, Lew et al. 2006).

Another basic selection strategy is *truncation selection* (Zhang and Mühlenbein 1995, Blickle and Thiele 1997). In this method a subset of the population, usually defined by a percentage of the best individuals, is admitted into a *mating pool*, the rest of the population being excluded from any further operation. Individuals undergoing genetic operations are randomly selected in the mating pool with uniform probability.

Selection have a profound effect on evolution (Luke and Spector 1997 1998), therefore it should be performed in a balanced way. If selection gives too much evolutionary advantage to highly fit individuals population variability may be excessively reduced and the evolutionary search constrained to a subregion of the design space (in symbolic regresson the design space is the space made by all the mathematical expressions that can be built using the given primitives and within the constraints imposed by trees' size or depth limits). On the other hand, if selective pressure is too low, and individuals probability to be selected loosely related to fitness value, the risk of a blind or undirected exploration of the search space is high. In both cases the chances of success are dramatically reduced: the evolution is likely to converge prematurely to a "globally suboptimal solution" (Koza 1992, p. 104). Such behaviour has been observed in all dialects of evolutionary computation (see Secton 2.3.1, Chapter 2) and for this reason many conclusions obtained in GA and ES can be applied to GP. This possibility is also underpinned by the independence of the selection mechanism from the rest of the GP implementation (Blickle and Thiele 1997, p. 362).

In this regard the selection methods presented above have serious drawbacks. Fitness proportionate selection tends to direct the search towards the exploration of genotypes similar to the ones of few fittest individuals, reducing the genotypes diversity throughout the evolution and undermining the global exploration of the search space (Blickle and Thiele 1997, Zitzler and Thiele 1999, Alvarez 2000). Koza himself acknowledged that it is likely to cause *premature convergence* (Koza 1992, p. 103) (the same conclusion is supported by Fogel (1994, p. 6), although for GAs). Truncation selection also drastically reduces the diversity in the population and leads to premature convergence (Blickle and Thiele 1997).

A range of alternative selection methods have been developed to preserve genetic diversity in the population, mainly classified as *rank-based* selection methods and tournament selection methods.

3.1.4.1 Rank-based selection

Rank-based selection methods were firstly suggested to overcome the drawbacks of fitness proportionate selection (Blickle and Thiele 1997). A ranking is introduced sorting first the individuals according to their fitness values, and then assigning rank M to the one with the highest (worst) fitness value, and rank 1 to the one with the lowest (best) fitness value, M being the number of individuals in the population. The selection still requires the generation of a random number as in fitness proportionate selection (roulette wheel sampling), but the probability of an individual to be selected is defined as a function of its rank. In *linear ranking selection*, the selection probability p_i of an individual of rank i is defined by a linear function (Blickle and Thiele 1997):

$$p_i = \frac{1}{M}\left(\eta^+ + (\eta^+ - \eta^-)\frac{i-1}{M-1}\right) \qquad i \in \{1, ..., M\} \tag{3.3}$$

where $\frac{\eta^-}{M}$ is the probability of the worst individual to be selected and $\frac{\eta^+}{M}$ the probability of the best individual to be selected. In statistics η^- and η^+ would be called the (theoretical) *frequencies* of the worst and the best individual respectively (Upton and Cook 1996). If the population size M is constant during the run, the condition $\sum_1^M p_i = 1$ imposes that $\eta^+ + \eta^- = 2$, with $\eta^+ \geq \eta^- \geq 0$.

In *exponential ranking selection* (Montana 1995, Blickle and Thiele 1997) the selection probabilities of the ranked individuals are exponentially weighted. The probability p_i of the individual having rank i is given by:

$$p_i = \frac{c^{M-i}}{\sum_{j=1}^{M} c^{M-j}} \qquad i \in \{1, ..., M\} \tag{3.4}$$

where $0 < c < 1$ is a parameter by which the probability distribution can be tuned. The term under the fraction line ensures that the sum of the p_i probabilities equals 1. As:

$$\sum_{j=1}^{M} c^{M-j} = \frac{c^M - 1}{c - 1} \tag{3.5}$$

the equation (3.4) can be rewritten as:

$$p_i = \frac{c-1}{c^M - 1} c^{M-i} \qquad i \in \{1, ..., M\} \tag{3.6}$$

Blickle and Thiele (1997) showed that exponential ranking selection allows to maximize at the same time variability and the average fitness increase between generations due only to selection. Also, varying the selection pressure through parameter c it is possible to control the time to convergence, as noted in Montana (1995).

3.1.4.2 Tournament selection

In *tournament selection* a subpopulation of t individuals is randomly extracted from the population (the selection can be done with or without replacement of the chosen individual). Such group of individuals, whose size t is defined as the *tournament size*, takes part to a tournament: the individual with the best fitness value wins and is selected. To get μ individuals the process is repeated μ times.

Fitness is the most common performance used in GP tournament selection (Luke and Spector 1997, Luke and Panait 2002a, Van Belle and Ackley 2002). Yet, other scores or parameters can be used to determine the winner of a tournament, for example the size of the syntax tree or Pareto dominance: more details will be given in Section 4.5.4, Chapter 4. Xie et al. (2006) proposed *clustering tournament selection*, in which the population is first divided into different subpopulations according to individuals' outputs on the training data set, and then the t competitors are selected from different clusters and the winner is randomly selected from the cluster that provided the individual with best fitness.

One of the major advantages of tournament selection is that it can be implemented easily and is effective in in preserving diversity in the population (Blickle and Thiele 1997, Xie et al. 2006). Blickle and Thiele (1994) provided a theoretical model to predict how many best-quality individuals are transferred from the old to the new generation as a result of tournament selection of size t. Assuming that in the old population there are $s(f_b)$ individual of fitness f_b, Blickle and Thiele (1994) estimated that the number s' of such individuals selected through tournament are:

$$s'(f_b) = M\left(1 - \left(1 - \frac{s(f_b)}{M}\right)^t\right) \tag{3.7}$$

where M is the size of the population. In case $s(f_b) \ll M$:

$$s'(f_b) = ts(f_b) \tag{3.8}$$

Although the result in Eq. (3.8) is valid only if the number of best individuals $s(f_b)$ is much smaller than the population size, and so accurate only at the beginning of the evolution or if a strategy to delete copies is in place, the equation is important as it shows the influence of tournament size on premature convergence. If the tournament size t is too high, the relentless replication of the best individuals predicted by Eqs. (3.7-3.8) (*takeover* effect) may result in loss of diversity and so premature convergence. In practice, tournament sizes commonly vary from 2 (Luke and Spector 1997 1998, Miller and Thomson 2000), to 7 (Vladislavleva et al. 2010), to 10 (Chellapilla 1997).

According to the analysis carried out in Blickle and Thiele (1997), exponential ranking selection, linear ranking selection and tournament selection are comparable in terms of diversity loss and average fitness increase between generations.

3.1.5 Genetic operators

Genetic operators role is to generate a new population from the privileged set of individuals identified by the selection process. The primary operators commonly used in genetic programming are *reproduction*, *crossover* and *mutation*. In Section 3.1.5.6 a few secondary operators that can be used in specific cases will be briefly described.

3.1.5.1 Reproduction

Reproduction consists in copying a selected individual to the new population without changes to the genotype. The definition of such term given in Chapter 2 may generate some confusion: according to neo-darwinism, sexual reproduction is the process through which two organisms generate offspring, whose genotype is generated by both crossover and mutation from parents' ones. In evolutionary algorithms, reproduction may be used to just indicate the replication of a parent's genotype without changes Luke and Spector (1997), Ferreira (2001), De Jong and Pollack (2003), Munroe (2004). This meaning will be adopted in this work: reproduction is then assumed to require a single parent and to generate a single, identical child.

Reproduction is usually associated with the concept of *elite*. The elite is a group of highly fit individuals that are propagated between generations without alteration, its use being termed *elitism*. In evolutionary algorithms elitism is a common way to keep a memory of the best solutions found during the exploration. It is usually recommended when at each generation extensive variations are introduced in selected individuals' genotypes, to prevent such huge variations from having destructive effect on the good individuals found (Zhang and Mühlenbein 1995, Ferreira 2001, Lew et al. 2006, Vladislavleva 2008). Elitism is usually implemented by maintaining an *archive*, which may be part of the population or a separate population (Teller and Veloso 1996, Sætrom and Hetland 2003, Vladislavleva 2008). In either case, the archive is usually updated with the best individuals at each generation.

Elitism, despite the many benefits brought to the evolution both in GP and GA (Fogel 1994, Zitzler and Thiele 1999, Deb et al. 2002), should be used in a balanced way. If used to massively, it can have a detrimental effect on diversity and pose limits to the design space exploration (premature convergence) (Luke and Panait 2002a, De Jong and Pollack 2003, Affenzeller and Wagner 2004).

3.1.5.2 Crossover methods

Crossover is historically considered the primary search operator in genetic programming (Koza 1992). The most common version of the crossover operator, generally referred to as *standard crossover* or *subtree crossover*, generates two offspring from two parents, swapping the subtrees branching out of two randomly selected nodes in each parent's syntax tree, called *crossover points*. In Fig. 3.4 is shown the crossover between the two following mathematical expressions:

$$\text{PARENT 1:} \quad \frac{\log x^2 + 3}{cos(y)} \qquad \text{PARENT 2:} \quad 2x + sin(z^3)$$

The crossover points are node 3 in parent 1 and node 5 in parent 2. Crossover generates two offspring pruning the subtrees branching out of the parents' crossover points and swapping them:

$$\text{OFFSPRING 1:} \quad \frac{sin(z^3) + 3}{cos(y)} \qquad \text{OFFSPRING 2:} \quad 2x + \log x^2$$

The result of the operation is two new syntax trees that resemble the parents but at the same present new features. The extent of the variation produced by subtree crossover on a syntax tree depends on the choice of the crossover points. In case stochastic node selection is used, the term *hit rate* is adopted to define the sampling frequency of a node (Banzhaf et al. 1998, Vladislavleva 2008). As nodes at higher depths on average outnumber nodes at lower depths, the probability of the nodes with higher depth of being

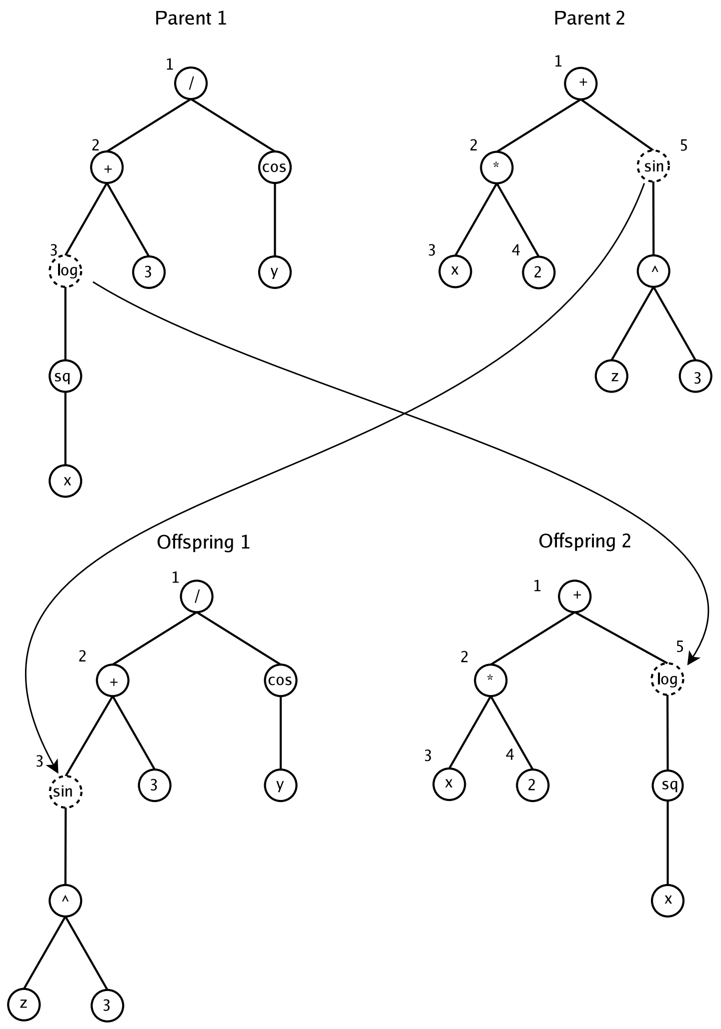


FIGURE 3.4: Standard or subtree crossover

selected is larger, and hence larger will be their hit rate in case of uniform random selection. Subtrees stemming out of nodes at high depths are necessarily smaller than subtrees developing from nodes close to the tree root node. Therefore, if crossover point selection is done through uniform random selection, the extent of variation on parents' genotype will decrease progressively during the evolution, as syntax trees' average size and depth increase (Koza 1992, Banzhaf et al. 1998, Whigham 1995, Van Belle and Ackley 2002, Vladislavleva 2008, Barbosa and Bernardino 2011).

For its simplicity, standard subtree crossover is widely used in GP community (Koza 1992, Nordin et al. 1996, Soule et al. 1996, De Jong and Pollack 2003), usually coupled with a mechanism to prevent the operation from taking place in case offspring exceed the maximum depth or size allowed (Whigham 1995, Montana 1995, Luke and Spector 1996, Richards et al. 2005).

However, standard subtree crossover suffers from a few limitations. Poli and Langdon (1998) defined subtree crossover as a "local" search operator, as it does not introduce extensive modifications in the genotypes and so it cannot explore globally the design space. They also observed that the genetic code that is exchanged is mostly transferred from one parent. Secondly, as explained above, subtree crossover is biased, as in absence of countermeasures the traditional node selection strategy tends to pick nodes close to the terminal ones. Its capability to explore new syntax tree shapes is also limited, as experiments using subtree crossover have shown that during evolution the average tree size to depth ratio remains constant and equal to the value of this ratio observed in the first generation (see Section 3.1.2). Thirdly, subtree crossover mechanism cannot ensure that the portions of genetic material that proved successful in the parents, usually referred to as *building blocks*, retain the same performance in the offspring.

If the bias on crossover point selection can be eliminated easily (in Section 4.5.2, Chapter 4 and Section 5.2.4.2, Chapter 5 a simple strategy will be described), ensuring that portion of successful syntax trees perform well when inserted in a different syntax tree is far more difficult. To study the effect of the location in a syntax tree where new genetic material (syntax subtree) is implanted on offspring fitness value, researchers have turned their attention to the DNA recombination mechanism occurring in nature. In homologous crossover, as briefly introduced in Section 2.2.2.2, Chapter 2, genetic material exchange from DNA regions that are responsible for the same phenotypical trait (called *loci*) is encouraged by the alignment of the DNA chains before recombination (Nordin et al. 1999). Back into the artificial world populated by syntax trees, alignment of this kind is totally absent in standard subtree crossover. Nordin et al. (1999) ascribed standard crossover "brutality" (Nordin et al. 1999, p. 290) and the high probability of generating offspring that are worse than parents to the complete neglection of the importance of the *context*, the location where a syntax subtree is implanted. The role of *context* is paramount in producing higly fit individuals from inherited code: to get the best fitness contribution, it is important that the code portion be surrounded by code that makes it perform well (Langdon 2000, Bleuler et al. 2001). Different strategies have been devised to ensure that the exchanged portions of genetic code are extracted and inserted in regions of the genotypes that are "similar" in the two parents. These attempts resulted in the development of *homologous* or *context-preserving crossover* operators, whose major implementations are detailed in Table 3.1, together with a few other alternative crossover operators commonly used (Banzhaf et al. 1998, Poli et al. 2008).

TABLE 3.1: List of most common context-preserving crossover operators

Crossover name	Effect
Size-fair c.	It generates a single child swapping subtrees selected from two parents. The crossover point in the first parent, the one from which the child inherits its root node, is selected randomly. In the second parent the crossover point is the root node of a subtree having on average the same size as the subtree to be replaced in the first parent (Langdon 2000)
Homologous c.	As size-fair crossover, but the replacing subtree in the second parent is selected to be not only of the same size of the subtree to be replaced, but also of the same shape and rooted at a similar location (Nordin et al. 1999, Langdon 2000, p. 102)
One-point c.	Crossover points are chosen in a common region shared by the two parents (Poli and Langdon 1998, Nordin et al. 1999, Poli et al. 2008, Vladislavleva 2008). The common region is identified traversing the parent genotypes and recording the nodes with the same position and number of arguments (arity).
Uniform c.	Similarly to one-point crossover, crossover points are selected in a common region between parents, so to preserve context. Instead of replacing a single subtree, many nodes or subtrees (if selected nodes are on the boundary of the common region) randomly selected in the common region are swapped (Poli and Langdon 1998). The mechanism is inspired by the uniform crossover for GAs (Poli and Langdon 1998). Uniform crossover is for example used in Kalganova and Miller (1999)
"Genetic c. dissolves"	It is a sort of interpolation between two individuals, according to which two subtrees are not swapped but faded, introduced by Sims (1993). Unfortunately Sims (1993) did not provide many details on the technique.
Self c.	Example of crossover involving a single parent. Randomly chosen subtrees belonging to the *same* individual are swapped (Poli et al. 2008).

The role of a few of the crossover operators presented above on syntax trees' fitness and size will be discussed more in detail in Section 4.4.1, Chapter 4.

3.1.5.3 Mutation methods

The mutation operation generates a single child from a single individual. A range of different strategies can be used to modify the parent genotype: mutation operators can just change a few nodes in a syntax tree or radically change the individual (Van Belle and Ackley 2002). In general the extent of genotypic change caused by mutation in the parent genotype is usually much wider than the one caused by crossover (Chellapilla 1997). This is due to the fact that the variation introduced by mutation does not depend on the variability of the population (Chellapilla 1997, Munroe 2004). Crossover generates offspring shuffling existing genetic material, so it will not introduce radical changes in case a population is composed of similar individuals and its effect will fade by the generation. Mutation instead introduces new genetic material (Munroe 2004). This is the reason why mutation can be effective in reducing the risk of premature convergence.

Mutation can be useful to escape suboptimal individuals, but also lethal, as it may turn a successful individual into a low-quality one in a single generation. Due to mutation unpredictability on fitness value, different methods have been developed to control the amount of variation introduced in the genotype (Chellapilla 1997). All of them require the choice, usually random, of a node, called *mutation point*, where the genotype modification has to take place. *Point mutation* and *subtree mutation* lie at the opposite ends of the spectrum regarding the amount of variation introduced in a genotype.

Point mutation is the mutation operator that produces the smallest variation in a syntax tree. It replaces the content of a node in the selected tree with another primitive of the same kind: a function is replaced by another function having the same number of arguments (arity), a variable or constant is replaced by another element of the terminal set (Sims 1993, Zhang and Mühlenbein 1995, Chellapilla 1997). An example is shown in Fig. 3.5: the mutation point selected in the parent is node 2, which is a functional binary node. The function contained in it is an addition (+). Mutation operator replaces this function with another binary function included in the functional set, here division (/):

$$\text{PARENT:}\quad \frac{\log x^2 + z}{cos(y)} \quad\rightarrow\quad \text{OFFSPRING:}\quad \frac{\log x^2}{z}\frac{1}{cos(y)}$$

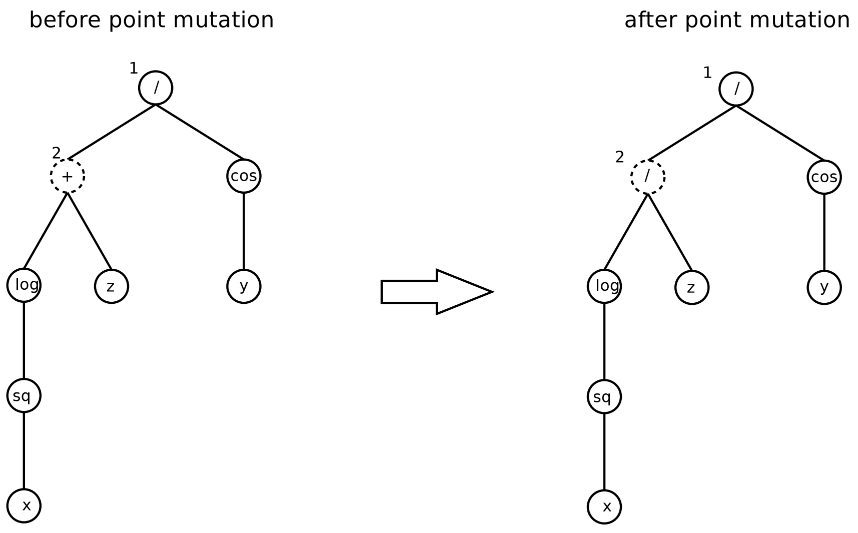


FIGURE 3.5: Point mutation

although only a single node has been modified, the metamodel has undergone a significative change from parent to offspring. To increase the amount of variation produced by point mutation, more than one node per syntax tree can be modified. A way to do it is to assign a threshold of mutation (a number) to each node of the parent: if traversing the tree such threshold is lower than a randomly generate number, the content of the node is replaced with another primitive of the same arity (Van Belle and Ackley 2002). All nodes in a parent tree are instead replaced with another randomly chosen node of the same arity by the operator *AllNodes* presented in Chellapilla (1997). Despite the number of nodes mutated in a tree, the amount of variation introduced by point mutation depends critically on the variability in the functional and terminal sets declared by the user, as no change in syntax tree structure is performed by the operator. If the range of primitives is limited, the amount of change from parent to child will be limited consequently.

Subtree mutation (Koza 1992, p. 106) produces a larger variation on the parent's genotype than point mutation, as it replaces the branch stemming out from the mutation point with a subtree randomly initialised[2]. An example is given in Fig. 3.6. Examples of the application of subtree mutation can be found in Koza (1992), Sims (1993), Langdon and Poli (1998a), Richards et al. (2005).

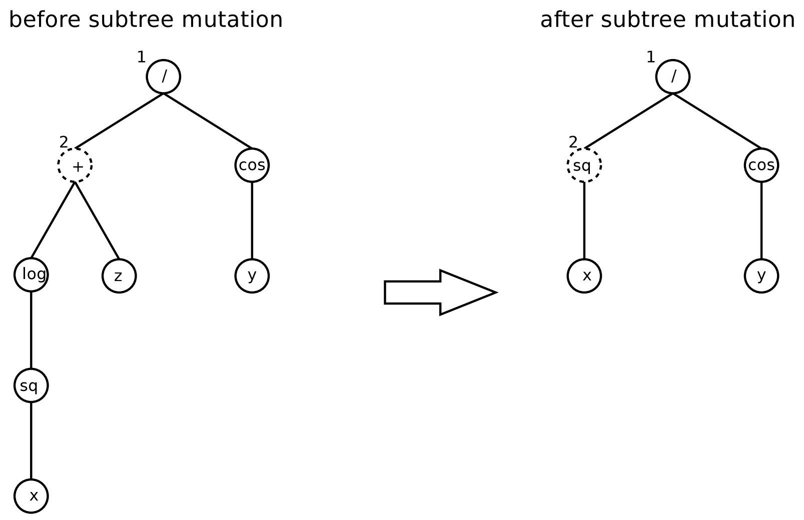


FIGURE 3.6: Subtree mutation

Many other mutation operators have been introduced. A few of them are described in Table 3.2. As seen for crossover, to enhance the effects of the mutation operators on the genotype, different selection strategies can be used to increase the probability of nodes of

[2]grow, full and ramped half-and-half methods are the most common ways to generate the subtrees. See Section 3.1.2.

lower depth to be selected (Brameier et al. 1998, Chellapilla 1997, Langdon et al. 1999, Vladislavleva 2008).

TABLE 3.2: List of most common mutation operators

Mutation name	Effect
Uniform subtree m.	(Van Belle and Ackley 2002). Each node of the parent tree is assigned a mutation threshold. If a node's threshold is lower than a randomly generated number, the node is substituted by a randomly generated subtree.
50%-150% fair m.	(Langdon 1998, Langdon et al. 1999). Subtree mutation in which the size of the replacing subtree is correlated with the size of the subtree to be replaced.
Size-fair subtree m.	(Langdon 1998, Langdon et al. 1999). Subtree mutation in which the size of the replacing subtree is the size of a subtree randomly chosen in the parent tree.
Hoist m.	(Poli et al. 2008). A "new" individual is created copying a randomly selected subtree from a parent (implicitly used by Smits and Kotanchek (2004) and Vladislavleva (2008), as during fitness evaluation they consider all individuals' subtrees as single individuals).
Expansion m.	(Sims 1993, Chellapilla 1997). A new individual is generated replacing a terminal node in the parent tree with a randomly initialized subtree (also called *grow* by Chellapilla (1997)).
Shrink (collapse) m.	(Sims 1993, Chellapilla 1997). A randomly chosen subtree is replaced by a randomly created terminal node (also called *trunc* by Chellapilla (1997).
Mutating constants at random	(Sims 1993, Chellapilla 1997). A new individual is generated adding Gaussian noise to constants in the parent tree (also called *Gaussian mutation* by Chellapilla (1997)).
Selective m.	(Aichour and Lutton 2007). As standard subtree mutation, but the subtree to be replaced is the subtree that contributes the least to the overall individual's fitness. To identify such subtree, each node in a tree is replaced with a neutral constant and the difference between the fitness values of the original tree and of the modified tree is computed. The subtree to be replaced is the one producing the least difference in fitness values.
Permutation	(Koza 1992). The subtrees stemming out of the left and right child nodes of a binary node are swapped. It is not actually a mutation operator, but it can be used to increase variability in the population (used for example in Chellapilla (1997)).

3.1.5.4 Methods to apply crossover and mutation

In GA and GP mutation is usually applied to the offspring genotypes as soon as they have been generated by crossover. This strategy aims at emulating the natural DNA recombination process, in which mutation is the result of errors occurring during genotypes' crossover. This approach was adopted for example by Affenzeller and Wagner (2004, p. 255) for GAs, whereas for GP examples can be found in Alvarez (2000), Topchy and

Punch (2001), Keijzer (2003). In Chellapilla (1997) crossover was not used, but up to six different mutation operator are applied in sequence on the same selected individual.

Alternatively, crossover and mutation can be applied independently (De Jong and Pollack 2003, Vladislavleva 2008, Barbosa and Bernardino 2011). The strategy is in general less disruptive or radical than the previous approach, and it allows for the independent analysis of the effect of the two operators, as done in Banzhaf et al. (1996) (although not using tree-based GP and not for symbolic regression tasks).

3.1.5.5 Role of crossover and mutation

The requirements for a successful evolutionary search are the possibility to explore the search space in its entirety as well as the ability to refine existing good individuals. These concepts are expressed by Koza (1992) using the terms “exploration” and “exploitation”. GP representation and genetic operators should be able to guarantee these two properties. The role of representation will be briefly discussed in Section 4.8, Chapter 4, where genotype-phenotype mappings different from syntax tree will be introduced. In this section the role of genetic operators in ensuring both the exploration of the search space and the exploitation of the best individuals will be analysed.

Poli and Langdon (1998) identified the properties that genetic operators should have to ensure exploration and exploitation. A genetic operator should be *global*, in the sense that it should produce offspring with the highest variety in shape, size and content, to the extent that the chosen representation and legality of the offspring allow it[3]. On the other hand, a search operator should also be *local*, or able to introduce small variations in parents’ genotypes, encouraging the inheritance of “good portions” of genetic material from a suboptimal parent and refining it.

Crossover and mutation are the primary search operators in GP as they are both global and local, depending on the specific implementation and node selection strategy. For example, point mutation or standard subtree crossover with crossover points selected at high depths perform a local, refining action on selected parents (Chellapilla 1997, Poli and Langdon 1998). Subtree mutation or standard crossover with crossover points selected close to the tree root node tend instead to generate offspring that bear little resemblance to the parents, leading the search to unknown regions of the design space. Due to the

[3]quoting Sims (1993, p. 468): “large random changes in genotype usually result in large jumps in phenotype that are less likely to be improvements, but are necessary for extending the expression to more complex forms”.

coexistence of local and global features in both crossover and mutation, the role of these operators in GP evolution has been debated since the birth of genetic programming.

Researchers tried to address the problem observing crossover and mutation roles in GAs. In genetic algorithms crossover is the primary generator of new trial solutions, but it is not able to reintroduce primitives, also called *alleles*, once these have been lost (Holland 1992, pp. 110-111). Mutation has therefore the important role of reintroducing fresh genetic material (lost alleles) in the evolution. Helping mantain variability in and among the chromosomes, it increases the search robustness to local suboptima and promotes local refinement. In this sense mutation helps GAs avoid premature convergence.

In principle, the beneficial effect that mutation has in GAs holds for GP as well (Chellapilla 1997, Munroe 2004, Winkler et al. 2007). However, there are a few practical issues that brought researchers to consider mutation less effective than crossover, at least in the early 1990s. Koza (1992) argued that subtree mutation is either "not needed" or it has "little utility" in GP, limited to occasional genotypes perturbations to lead the evolutionary search out of regions of local optima. According to Koza (1992), design space exploration is effectively performed by crossover through the aggregation of successful portions of genetic material, called *building blocks*, which would allow to find the right evolutionary path to the global optimum. The building block theory, relying on the assumption that portions of successful genetic code can be identified and exchanged by crossover, was originally formulated for GAs and extended by Koza (1992) to GP. Mutation, on the other hand, was considered not able to propagate building blocks. Also, its function of reintroducing lost alleles and restoring lost diversity in the population is not as critical as in GAs, where alleles are likely to be lost. Koza (1992) argued indeed that the probability to lose alleles in a GP evolution is far smaller than in GAs, as primitives are usually far fewer than the nodes of a generic GP tree. As a result, for a few years many researchers hardly used mutation in their experiments (Montana 1995, Langdon and Poli 1998b, Luke and Spector 1997, Soule and Foster 1998ba, Langdon et al. 1999, Langdon 2000, Van Belle and Ackley 2002, De Jong and Pollack 2003).

The attitude towards mutation changed from the end of the 1990s (Banzhaf et al. 1996, Luke and Spector 1997, Van Belle and Ackley 2002). On the one hand, the application of building block theory to GP was questioned (Luke and Spector 1997, Chellapilla 1997), and on the other new experiments provided evidence against crossover supremacy as search operator both in GP and GAs (Banzhaf et al. (1996) and in particular Luke and

Spector (1998, p. 210): “where and why one is preferable to the other is strongly dependent on domain and parameter settings”). Many other studies have shown that mutation has an effective role in GP evolution (Banzhaf et al. 1996, Chellapilla 1997, Whitley et al. 2006, Winkler et al. 2007, Poli et al. 2008). Theoretically, the new interest in mutation can be explained by the fact that such operator can increase the sampling frequency of lost primitives and so boost genetic diversity and reduce the risk of premature convergence (Luke and Spector 1997, Poli and Langdon 1998). Furthermore, the local or refining action they perform on individuals is key in reducing convergence time and it is not consistently performed by crossover (Chellapilla 1997, Poli and Langdon 1998).

The renewed interest in mutation has brought a few researchers to completely rely on mutation for the generation of trial solutions, abandoning crossover (“the results clearly indicated that recombination operators (e.g., subtree crossover) are not indispensable and that the use of mutation operators alone is capable of generating appropriate programs” Chellapilla (1997, p. 216)). Despite Chellapilla’s conclusions, today the belief that crossover and mutation are both needed in GP is widely accepted (Munroe 2004, Lew et al. 2006, Vladislavleva 2008).

Extensive research has been dedicated to discover the optimal (constant) rates at which crossover and mutation should be used to maximise the quality of the evolved individuals (Nordin et al. 1996, Banzhaf et al. 1996, Poli and Langdon 1998, Lew et al. 2006). Munroe (2004) introduced a further element, showing that genetic operator importance varies during the evolution and so crossover and mutation optimal rates cannot be constant. He proved that mutation is more effective in increasing individual fitness value than crossover during the early stages of GP evolution. Crossover regains its importance in generating trial solutions only after an intermediate stage where both genetic operators are equally effective. These conclusions are consistent with the observations reported by Nordin et al. (1996), who performed symbolic regression of a second order polynomial with linear[4] and tree-based GP implementations using subtree crossover and point mutation. Nordin et al. (1996) observed that crossover starts to generate individuals with a better fitness value than their parents (*constructive* crossover) only after a few generations, and after a period of exploration where it is predominantly destructive (offspring have a worse fitness value than parents), crossover becomes responsible of exploring alternative solutions having the same fitness but of increased size (neutral crossover). Unfortunately Nordin et al. (1996) limited their analysis to crossover, so a

[4]more details on the linear GP representation will be given in Section 3.3.

more precise match with Munroe's observations cannot be established. However, the results are important as they allow to formulate the hypothesis that GP evolution can be split in three stages: *youth*, *maturity* and *stagnation*. Further research on this aspect of evolution might lead to interesting discoveries.

In conclusion, twenty years after Koza's book, it seems that more than the single use of either crossover or mutation, finding a correct blend of the two is more effective for ensuring GP success. For example, Lew et al. (2006) applied cross validation and hold out methods (see Section 1.2.2, Chapter 1) to optimise reproduction, crossover and mutation rates.

In the next chapter, in Section 4.4.1, a further element, the presence of neutral code in syntax trees, will cast a new light on the role of crossover and mutation.

3.1.5.6 Secondary genetic operators

Although GP mainly relies on crossover and mutation for search purposes, Koza (1992, p. 105-7) introduced a range of operators that are occasionally used:

- *Editing*: simplification of syntax subtrees during evolution according to predefined rules. For example a subtree composed of only numerical constants and functions can be replaced by the result of the operations. Editing can also be used to remove subtrees that have no effect on the fitness value of the syntax tree (see Section 4.5.1, Chapter 4).

- *Encapsulation*: operation through which a subtree is converted into a primitive, and so it can be reused to generate new syntax trees (Koza 1992, pag. 110). Encapsulation is the most common way to reuse valid portions of evolved code (Angeline and Pollack 1993, Whigham 1995).

- *Decimation*: deletion of a certain percentage of the population, which is regenerated using the initialisation methods. It is a strategy used to restart the evolution afresh, or to abruptly divert the exploration to new regions of the design space. It was used for example by Smits and Kotanchek (2004) and Vladislavleva (2008) under the name *cascade* to periodically delete the entire population. To avoid loss of precious genetic material, an external archive storing the best individuals was maintained.

3.1.6 Termination criteria

Termination criteria define the conditions under which the evolutionary search is stopped (see Section 2.3, Chapter 2). The simplest and most commonly used criterion is to terminate the evolution when the quality of the best individual, measured by fitness (for example RMSE) or by the number of *hits* (Koza 1992), reach a certain threshold. A *hit* is a score associated to an individual, ranging from 0 to the number of fitness cases in the building data set. A hit is scored when the response produced by a syntax tree for a fitness case is within a given tolerance from the corresponding actual response (Banzhaf 1994, Chellapilla 1997).

Criteria based on the measurement of the current quality of the best individual evolved, like the ones described above, fail in detecting if the population as a whole still has the potential to further improve the quality of the best individual. As a result, they may prematurely terminate the evolution, in case scarce knowledge of the problem leads the user to define a low threshold. A few termination criteria have been developed to stop the evolution automatically when the whole potentiality of the evolution is considered exploited. These approaches are based on different definitions of population "hidden potential":

- *hidden potential as impossibility to improve the population* (Affenzeller and Wagner 2004, Yin et al. 2007). This approach assumes that convergence is reached when it is not possible to generate a certain number of individuals that have better fitness value than their parents.

- *hidden potential as impossibility to perturb the population* (Banzhaf et al. 1996). In Section 3.1.5.5 it was pointed out that in the final stage of the evolution (stagnation stage), the dramatic decrease of both destructive and constructive crossover rates determines the impossibility to improve or degrade the quality of the best individuals (Nordin et al. 1996). The proportion of destructive crossovers performed each generation can then be used as a signal of the impossibility to further perturbate the population, and so of convergence. This strategy was used in Banzhaf et al. (1996): the rate of destructive crossover was monitored throughout the evolution, and when that rate fell under 10% the evolution was stopped. It is interesting to note that this approach is slightly different from the previous one, as it is based on the idea that convergence is a condition in which the fitness of the best individuals of the population can neither be increased nor decreased using crossover.

Termination criteria also have the practical duty to stop the evolution if it gets too computationally expensive, regardless if it has converged or not. Different parameters can be used to impose a computational budget to a GP evolution:

- maximum number of generations (Koza 1992);
- maximum number of fitness evaluations (number of individuals evaluated) (used for example in Vladislavleva (2008));
- maximum number of node evaluations (De Jong and Pollack 2003)

Checks based on the first two parameters are really common and easy to implement. According to De Jong and Pollack (2003), monitoring the number of node evaluations provides a more accurate approximation of the computational effort required by the evolution of variable length structures as syntax trees, as such parameter directly depends on the size of the individuals.

In real-life GP applications, a criterion that allows to monitor both quality and cost of the evolved individuals is often opted for. An example can be found in Section 5.2.6, Chapter 5.

3.2 Data generation and results analysis

The selection of the building data set DoE is critical for the generation of high-quality metamodels, as seen in Chapter 1. Surprisingly, the issue has been exhaustively addressed in GP only by a limited number of researchers, among which Vladislavleva (2008). In Section 3.2.1 her main conclusions will be reported. Moreover, the analysis of the results produced by GP, and in particular the comparison of the metamodels produced by two different GP implementations, require specific statistical techniques, due to the stochastic nature of GP algorithm. In Section 3.2.2 a brief survey of such techniques will be presented.

3.2.1 Design of experiment: requirements for GP

As for any metamodelling technique, the distribution and the size of the building data set is critical for the quality of the metamodels returned by GP. In particular, what is the optimal DoE for GP? And how many points should be used to allow for the generation of a high-quality metamodel? How does the size of the DoE affect the computational cost of the evolution?

It is extremely hard to answer these questions using mathematical theory, mainly because GP does not assume any mathematical structure for the solution, the mechanisms used by GP to explore the design space are stochastic and the solution of the symbolic regression problem is not unique. Surprisingly few GP researchers paid attention to the data generation stage, as observed by Giacobini et al. (2002) and Vladislavleva et al. (2010), opting for randomly generated building data sets. This is the case for all the GP symbolic regression examples described in Koza (1992, Chapter 10). Zhang and Mühlenbein (1995) also used a randomly selected building data set. In Ferreira (2002) and De Jong and Pollack (2003) two symbolic regression problems were performed using a dataset randomly chosen in the interval $[-1, 1]$. In Banzhaf (1994) no details are provided regarding how the fitness cases were selected.

The lack of interest in the DoE selection may be explained perhaps by the practical simplicity of randomly generating a set of points. A similar attitude is often observed in the applications where data are already available: their direct use as training data set, without performing any control on their distribution, is common (Smits and Kotanchek 2004) and may be explained by the belief that having more data, regardless their ditribution, is always better than having fewer data. On top of that, probably some overconfidence on GP ability should be taken into account.

The metamodelling background provided in Chapter 1 gives some guidelines to identify the optimal DoE for GP. It was there reported that the use of space-filling designs is a good practice "in the early stages of design, when the form of the metamodel cannot be specified" (Simpson et al. 2001). This the typical scenario in which GP works, as it searches for the optimal mathematical structure fitting the input data. A space-filling design provides GP with information regarding the behaviour of the underlying function reducing the average extension of unsampled subregions of the design space. As the aim of GP as a metamodelling technique is to generate global metamodels, a uniform, uniformly weighted, space-filling DoE will be assumed as the ideal DoE for symbolic regression through GP.

Among the space-filling DoE available, the class of Latin Hypercube DoEs represents a good compromise between space-filling properties and number of points, which can be chosen by the user according to the computational cost of experimentation (see Section 1.2.5.4, Chapter 1). Alvarez (2000) for example used Audze-Eglais Latin Hypercube

DoEs generated by a permutational genetic algorithm (Bates et al. 2004). Full factorial DoEs have superior space-filling properties than Latin Hypercube, but, as noted in Section 1.2.5.1, Chapter 1 their cost in terms of data gathering may be unfeasible for high-dimensional and expensive systems.

The research done by Vladislavleva (2008) is based on the different assumption that data are already available, but it is important as it indirectly confirms that space-filling designs are optimal for GP. She proved on a few test cases the benefits on GP symbolic regression of reducing a large and almost randomly generated input data set to a smaller and uniformly distributed (balanced) subset. *Proximity*, *surrounding* and *non-linearity* weights (Harmeling et al. 2006) based on the distance of each point to the closest points (*neighbours*) and on the corresponding output were used to measure the degree of uniformity of the design space sampling and to assess the importance of each point of a randomly generated input data set. A ranking based on the weights was used to reduce the data set to a compressed and almost uniformly distributed (*balanced*) one, which was then used as training data set for GP[5]. The comparison of the best metamodels produced using the original and the compressed data sets (Vladislavleva 2008) showed that not lower or even better accuracy (on the original data set) was reached when the smaller but approximately uniformly distributed DoE subset was used. Particularly good metamodels were obtained when the data set compression was done using the weight capturing the rate of variation (non-linearity weight) of the output in a neighbourhood centred in each sample point. Vladislavleva's results are important as they confirm the previous assumption on the beneficial effects of space-filling design on metamodelling through GP. In particular they prove that:

1. having more data is not necessarily better than having fewer data;
2. data distribution is critical to GP metamodel accuracy. For the same number of DoE points having a uniform distribution increases GP metamodel accuracy;
3. to improve GP metamodel accuracy, it is better to increase the population rather than enlarging in an unplanned way the input data set.

The last question to answer concerns the optimal size of the DoE. The non-parametric nature of GP does not allow to identify the minimum number of points required to generate a metamodel. For parametric techniques this number is a function of the number

[5]the GP implementation used in Vladislavleva (2008) is Pareto GP: a brief description will be given in Section 4.5.4.4, Chapter 4.

of unknowns in the model, depending if the fitting is performed in a standard way or according to a least-square approach (see for example response surface methodologies or polynomial chaos expansion (Eldred et al. 2008)). Giacobini et al. (2002) showed that it exists a minimum number of points from which a boolean function can be completely reconstructed, but no recommendations on how to select such minimal building data set were given. In the absence of any conclusive answer to this question, generally it is recommended to use as many points as possible, to fight the curse of dimensionality. In Chapters 5 and 6 experiments on benchmarks and real-life applications will give an idea of the dependence between the accuracy that can be expected from a GP metamodel, the dimensionality of the problem and the number of points in the training data set.

3.2.2 Results analysis

GP metamodels' accuracy and generalisation ability is generally checked evaluating root mean squared error ($RMSE$), coefficient of determination (R^2) (Montgomery et al. 2006, Ramu et al. 2010), maximum absolute error, maximum percentual error on an additional (different) *validation* data set, better if uniformly sampled on the design space and larger in size than the building data set, as suggested in Simpson et al. (2001, pag. 142). A validation methodology tailored to GP metamodels is described in Vladislavleva (2008).

As GP is a stochastic technique, each single GP evolution, also called *run*, may in theory generate a different metamodel: some may terminate stuck in a suboptimal region of the design space, whereas others may escape these regions and provide an outstanding metamodel. To increase the probability of obtaining a globally optimal metamodel, generally in a GP *experiment* many runs are launched, typically from 10 to 50 (Schoenauer et al. 1996, Luke and Spector 1997, Chellapilla 1997, Luke and Spector 1998, Vladislavleva 2008), out of which the best metamodel per run is extracted through the validation process above described.

The set of validation errors associated to the best metamodels generated by each run of a GP experiment, also called validation error *distributions*, are generally used to compare the exploratory power of two GP implementations, provided that the same computational budget (see Section 3.1.6) is granted to all runs.

A first way to compare GP experiments is to compare the average of the validation error distributions (Soule et al. 1996, Chellapilla 1997). Nordin et al. (1999, pag. 295-6) compared two radically different GP implementations, based on different representations (machine code linear GP and tree-based GP) using average performance. Luke and Spector (1997 1998) and Soule and Foster (1998b) used student's two-sample *t*-test (Upton and Cook 1996) to compare mean standardised fitness.

A comparison based on average parameters may however not be reliable. The averaging process can obscure characteristic features of the evolutions (Blickle 1996), as they mix data produced by successful runs and failed runs, and could bring to misleading conclusions. Soule and Foster (1998a) termed the difference in individual performance due to failed or successful exploration as *bimodal distribution*, also suggesting that analysis should be done on successful runs only, and (badly) failed runs filtered out. Soule and Foster (1998a) considered the failed runs useful only to understand why the run failed. Also Vladislavleva (2008) recognised the problem. She defined a threshold on the validation error and considered all the generated metamodels whose error was beyond that threshold as outcome of failed runs (called *bad* runs). Failed runs were then treated differently from successful runs in the analysis of GP performance.

Many researchers preferred to use statistical methods to compare the validation error distributions produced by different GP implementations. The most used statistical methods are ANOVA test and rank-based, non parametric tests. ANOVA test is used to assess if two sets of measurements, normally distributed, are the realisation of the same normal distribution (Upton and Cook 1996). This test was used for example by Luke and Panait (2002ab). ANOVA test however assumes that the measurement sets are normally distributed. This assumption may not be true for GP results, as validation error sets may be strongly skewed. Rank-based non parametric methods are used to test if there is significant evidence of a difference between the medians of two data sets. They do not require that the data sets to be compared be the realisation of any particular distribution, but it is generally harder to get conclusive results from them than from ANOVA tests. Wilcoxon rank sum test (also called Mann-Whitney U test) is generally used to compare two validation error sets, whereas Kruskall-Wallis test is used to spot median differences in more than two error sets (Upton and Cook 1996). These tests were used for example in Hollander and Wolfe (1999), De Jong and Pollack (2003), Hornby (2006), Vladislavleva (2008) and Vladislavleva et al. (2010).

A tool that is often used to graphically represent the distribution of the metamodels' errors on training or building data sets is the *boxplot* (Upton and Cook 1996). A few examples of its use can be found in Luke and Panait (2002ab), Helton and Davis (2003), Zitzler and Thiele (1999), Whitley et al. (2006) and Vladislavleva (2008). A boxplot is a diagram that represent the median, the first quartile (25% percentile), the third quartile (75% percentile) and extreme values of a given data set, as shown in Fig. 3.7. The distance between the first and the third quartile is called *interquartile range*, and corresponds to the width of the rectangular box in Fig. 3.7. The lines extending out of the rectangle represent the samples belonging to the first fourth and the last fourth of the data: by convention their maximum length is one and a half times the interquartile range. Samples represented by plus sign (+) are considered outliers (Harmeling et al. 2006). Boxplots will be used frequently throughout this dissertation to represent the errors of the best metamodels produced by a GP experiment (see Chapters 5, 6).

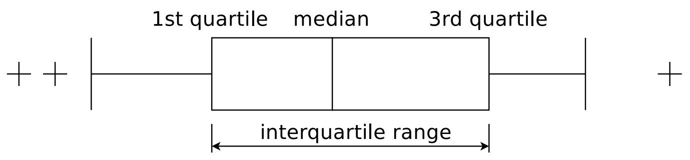


FIGURE 3.7: Boxplot

3.2.3 Computational effort analysis

The method originally used by J. R. Koza to measure and compare the exploratory power or performance of GP implementations was based on the probability $P(M, i)$ of a run using a population of size M to find a *correct* solution within the first i generations. This probability was called *probability of success* and was defined as (Koza 1992):

$$P(M, i) = \frac{\text{number of successful runs at generation } i}{\text{total number of runs}} \tag{3.9}$$

J. R. Koza's *computational effort analysis* aims at computing the number of independent runs $R(z)$ that need to be launched and the total number of individuals $I(M, i, z)$ that need to be processed to solve a problem with a probability of success z at generation i (Koza 1992, Chellapilla 1997, Van Belle and Ackley 2002). Computational effort is a useful parameter to assess GP potentiality (Koza 1992, Whigham 1995, Chellapilla 1997, Miller and Thomson 2000, Bleuler et al. 2001, Van Belle and Ackley 2002) but its application to symbolic regression problems is hindered by some practical issues. First of all, the definition of successful metamodel is tricky. Referring to a zero or reduced value of the error on the building data set can be misleading, as generalisation has to be taken into account. Even measuring the error on the the validation data set, it is not easy to define a threshold under which the metamodel can be considered successful, as the actual GP search efficacy is not known a priori. Furthermore, often the selection of the best metamodel depends on parameters other than error, like compactness of the expression and smoothness, and so it requires human intervention (Kordon and Lue 2004) and most of the times ends up being a trade-off analysis among equally optimal solutions. The difficulty of unambiguously defining a success predicate in symbolic regression makes definition (3.9) unusable most of the times. In Bleuler et al. (2001) computational effort was indeed used to compare different GP implementations on a boolean problem (even-k-parity function), not on a metamodelling one. Secondly, a reliable computation of the probabilistic model $P(M, i)$ on which computational effort analysis is based requires the analysis of a statistically meaningful number of runs, far larger than the number normally used in a GP experiment. As a result, computational effort analysis is not widely used for symbolic regression tasks.

3.3 Alternative GP representations

Tree-based GP is the most common type of GP found in literature nowadays (Barbosa and Bernardino 2011). Although historically genetic programming has been associated to a tree-based representation (Cramer 1985, Koza 1992), other representations have been introduced to improve GP performances and to allow the structure of the individuals to better reflect the nature of the problem under study.

Representation has indeed an important part in ensuring the exploration capability of GP and its quick execution. As emphasised by Vladislavleva (2008, p. 83-4), a good representation should allow fast evaluation speed and guarantee the legality of the offspring produced by the genetic operations. If legal individuals cannot be generated a priori, the representation should not introduce excessive computational overhead to check the legality of individuals' nodes and to fix the illegal individuals. Furthermore, it should allow for both radical changes and small variations on the individuals genotype through the application of genetic operations, so to foster exploration but at the same time preserve important subtrees.

The major GP representations alternative to syntax tree are the *linear* (Nordin et al. 1996, Banzhaf et al. 1996, Nordin et al. 1999) and *graph* structures (Teller and Veloso 1996, Brameier et al. 1998, Kantschik and Banzhaf 2001a, Brameier and Banzhaf 2007, Poli et al. 2008, Withall et al. 2009).

The linear genotype is the simplest and most versatile representation that can be used to implement a GP code, as made of strings of instructions acting on computer registers. Its use is however discouraged by some practical limitations. In the applications where execution speed is critical, the use of machine code is recommended but may raise further issues related to portability (see Section A.1.1.1, Appendix A). Furthermore, although the linear approach is versatile and can be used to perform different tasks, like algorithm and control law generation, it does not appear suited for symbolic regression problems, at least if an explicit metamodel is expected. Linear GP paradigm is based on a so called *imperative approach*: this means that the metamodel evolved by a linear GP implementation is not a symbolic expression but a program that behaves like such (Nordin et al. 1999, p. 281, 293-4). So linear GP is not able to perform symbolic regression, at least not in the literal sense. Although an explicit metamodel can be in principle obtained processing a linear GP individual, this is in practice not done as the conversion has to be performed manually and so it may be hindered by the size of the individual[6].

The graph is a generalisation of the syntax tree and so it is generally more versatile. On the one hand this means that more complex relationship can be represented with graphs than it is possible with linear and tree structures. On the other hand, thanks to the freedom in establishing multiple connections between nodes, a graph is able to represent a complex program with few nodes and arcs (Schmidt and Lipson 2009a). As a result, in graph-based GP code can be reused more extensively than in tree-based GP, leading to a reduction in individual size that, in turn, increases the execution speed and reduces RAM memory usage. The increased versatility of the graph structure implies also that graph-based GP can be used to perform symbolic regression of multiple output functions (vector functions), operation that is not easily achievable in tree-based GP (syntax trees can usually handle only single output programs). Such increased versatility has however a cost. The general complexity of the graph representation implies that the development of genetic operators for graph genotypes and ensuring the legality[7] of the connections between nodes is more difficult than for syntax-trees (Barbosa and Bernardino 2011).

[6]A quote taken from the *Discipulus*™website illustrates the point: "But all versions of Discipulus™output the evolved programs as CODE, not as equations. [. . .] That said, . . . , you can read the code and reduce the code to an equation. But some computer programs Discipulus™can write do NOT reduce to a simple equation. So we cannot do that automatically. It takes a human to make a conversion from code to the simplest and most readable form of an equation." (excerpted from `http://blog.rmltech.com/2012/05/turning-discipulustm-program-outputs.html` on July 11, 2012).

[7]propriety that will be called *closure* in Section 4.1, Chapter 4.

Chapter 4

Main genetic programming challenges

Genetic programming implementation has undergone substantial changes since the introduction of the tree-based formulation by Cramer (1985) and Koza (1992), described in the previous chapter. Alternative representations have been developed, and many basic components, among which fitness function and genetic operators, have been redefined to address a problem that is intrinsically linked to GP variable length representation: *bloat*, defined as "program growth without significant return in term of fitness" (Poli et al. 2008, p. 101).

This chapter is a collection of the solutions provided by researchers to the main GP standard algorithm pitfalls. The survey is not limited to GP implementations for symbolic regression tasks, as it is believed that the same phenomena can be observed to a certain extent in all GP implementations, regardless the task and the (variable length) representation. A wider analysis may therefore help identify solutions that could be successfully applied to symbolic regression tasks.

4.1 Closure property

The correct execution of a GP evolution relies on two main factors: the correct execution of genetic operations, linked to representation and genetic mechanisms, and the correct execution of fitness evaluation procedures, which instead depends on primitives and fitness function definitions. These two properties are usually encompassed in a single term, "closure property". Yet, due to the different GP mechanisms and components they involve, it seems reasonable to give them specific names: *syntactical closure* and *semantical closure*.

4.1.1 Syntactical closure

Syntax is the branch of linguistics that studies the principles and the rules used to build structurally correct sentences, regardless the meaning. As the nodes of a genotype need to satisfy specific structural conditions to form a legal individual, as words do to form a sentence, the expression "syntactical closure" will be used to refer to the correctness of the links between the single nodes of a GP individual.

GP individual syntactical correctness or legality has to be ensured during the first stage of any GP evolution, initilisation, but it may be disrupted by the genotype modifications performed by genetic operations.

The mechanisms that lead to the generation of illegal genotypes depends on the representation and on the specific details of genetic operations, so giving a general solution is not possible. In tree base GP for symbolic regression tasks, for example, illegal genotypes can be generated by point mutation, in case it replaces a node with another one of different arity: the resulting mismatch in the number of arguments introduces a condition that, if not handled by repair strategies, cannot be processed by GP. The issue may be more radical when different data types are used (Montana 1995): in this case not only does the number of arguments have to match, but also the type of data returned by the substituted and replacing node has to be the same.

In general, three main approaches have been followed to ensure syntactical closure:

- deletion: individuals that cannot be interpreted (not legal) are either immediately discarded and replaced by new ones, or penalised by harsh fitness values (Ryan et al. 1998).

- a posteriori correction or repair (*dynamic data typing*): after being generated, illegal individuals are repaired and accepted in the population. Sims (1993) repeatedly applied genetic operators until a legal individual is found. In (Keijzer and Babovic 1999) a deterministic repair mechanisms is used to fix all illegal individuals. In some cases repair algorithms can be very complex and specific to the GP implementation (Kalganova and Miller 1999). In general this approach results in additional computational cost.

- a priori generation of feasible individuals: individuals are created legal.

It has been noted that the the mechanisms used to ensure genotype legality, either "a priori" or "a posteriori", may constrain the exploratory path (Whigham 1995, Banzhaf et al. 1998). For the interested reader, two interesting a priori approaches are strong data typing approach (Montana 1995) and grammatical approaches (Whigham 1995, Ryan et al. 1998).

4.1.2 Semantical closure

Once syntactical correctness of the individuals is achieved there is a second type of closure that has to be satisfied, which could be called *semantical closure*. *Semantics* is the branch of linguistics that studies how meaning is associated to a symbol, so semantical closure is an expression that will be used to describe the condition in which a meaning, or fitness value, can be extracted from the symbols representing a GP individual during the fitness evaluation stage. The independent analysis of semantical closure is motivated by the fact that syntax closure does not necessarily imply that fitness value is defined.

The analysis of semantical closure is strictly related to primitives, fitness function definition and the training data set, so it is not possible to provide general conclusions and recommendations. In GP implementations for symbolic regression tasks, semantical closure is satisfied if the constants, variables and functions return values that once processed by any other function in the functional set, result in a well-defined (real) value. The training data set may affect the semantical closure of a GP individual, as functions may not be defined on the entire real axis. Division for example is not defined when the divisor is zero.

The most common way to guarantee semantical closure is to use *protected operations* (Koza 1992, Montana 1995, Langdon et al. 1999). Whenever a functional primitive is not defined because of the particular value of the arguments, as for example in the following cases:

$$\begin{aligned} \frac{a}{x} \quad & with \quad x = 0 \\ log(x) \quad & with \quad x \leq 0, \end{aligned}$$

the operation is forced to return a predefined real value. For example, Koza (1992) forced undefined division to return 1 to make it easier for GP to generate constants exploiting division neutral property (Koza 1992, Keijzer 2003), but less common values like 0 have been used (Xie et al. 2006).

On the one hand, protected operators ease the generation and the use of constants and then may help the exploration allowing for a local refinement of individuals (see Section 3.1.5.5, Chapter 3). On the other hand, the use of protected operators may lead to unreliable assessment of the metamodels corresponding to a GP individual, as the actual result of an operation, either defined or not, is replaced by a user-defined value (Montana 1995). Keijzer (2003) observed that the effect of a protected operation on a GP individual phenotype (metamodel) is to locally replace the highly non-linear behaviour associated with the presence of asymptotes with a well-defined behaviour of lower non-linearity (Keijzer 2003, p. 71-72). Such correction can result in misleading conclusions regarding the quality of the original metamodel, as during metamodel validation protection mechanisms are not used. To avoid such problems, different strategies from protected operations can be used. The presence of undefined operations can be detected and individuals affected by them can be penalised through fitness value. Nordin et al. (1999) for example marked the undefined GP individuals with a symbol INF and penalised them during the evolution. Alternatively, genotypes containing undefined operations can be deleted from the population as soon as they appear (Keijzer 2003). Both the previous approaches however do not protect against the risk of producing a metamodel that turns out to be undefined on some regions of the validation set or the design space. A more conservative approach could be to use only the functional primitives that give birth to metamodels defined on the entire real axis (Keijzer 2003), like polynomials, although this may result in mathematical expressions of excessive size and poor accuracy (see for example experiments reported in Section 5.5.3.1, Chapter 5).

4.2 Sufficiency property

The problem faced by GP is to search for a suitable program in the space defined by all possible programs that can be built using the primitives defined by the user. The first requirement for GP success is that the search space implicitly defined by the selected primitives contains the solution that is looked for, or at least an acceptable approximation thereof (Langdon et al. 1999). If that happens, the *sufficiency* property is satisfied.

In the case of symbolic regression, sufficiency means that at least a metamodel of acceptable accuracy can be built as a combination of the available terminals and functions.

In some cases, a reduced range of primitives may be useful to produce alternative formulations of known solutions (Ryan et al. 1998). An example will clarify this point. Let's suppose that the objective function (the solution) is $sin(y)$: if the terminal set is y and the functional set includes $sin()$, a GP implementation is likely to find the solution in its original formulation, $sin(y)$, during the first generation as a result of random aggregation of primitives. But if the functional set included only algebraic operations (addition, summation, multiplication, division) and power, neglecting for now how constant values can be found, a run would most likely generate an expression like the following:

$$y - 0.16667\,y^3 + 0.00833\,y^5 \simeq y - \frac{y^3}{3!} + \frac{y^5}{5!} \tag{4.1}$$

which is the Taylor expansion of $sin(y)$ around $y = 0$ limited to the first three terms.

Expression (4.1) is undoubtedly a valid approximation of the solution $sin(y)$ in a region centred in $y = 0$ and it is somewhat simpler than the original expression, as it does not use trigonometric operations. A similar example is reported in Ryan et al. (1998), where Grammatical Evolution (see Section 4.8) used for symbolic regression of the polynomial $x^4 + x^3 + x^2 + x$ on $[-1, 1]$ returned expressions containing sin or exp.

However, if the range of primitives is reduced up to the point that no combination of them represents a function that reasonably fits the data, GP search is doomed to failure. A wrong selection of primitives can also dramatically increase the difficulty of the search (as noted in Bleuler et al. (2001) for the generation of the even-k-parity function problem).

4.3 Multiobjective fitness function

The aim of any metamodelling technique is to generate a metamodel that is a "good" approximation of the true underlying model that accounts for response variance, as introduced in Section 1.1, Chapter 1. Once a metamodel is generated, the assessment of its quality does not require but the evaluation of the integral error I (Eq. 1.5), in one of its formulation ($RMSE$, R^2, max absolute error, etc ...), on the validation data sets. A "good" metamodel can then be recognised by a reduced integral error.

The main challenge that GP has to confront is to evolve a metamodel with an acceptable integral error I on the entire design space from the limited amount of information provided by the training data set (Zhang and Mühlenbein 1995). The original GP paradigm (Koza 1992), described in the previous chapter, implicitly assumes that the minimisation of the error on the training data set is sufficient to lead the evolution towards "good" metamodels. Although this approach may in some cases be successful, it is flawed by the fact that a zero error on the training data does not imply that a metamodel has good *generalisation* ability, in other words that it provides an acceptable approximation of the true underlying function on the whole design space.

Fig. 4.1 illustrates the point: the noisy function is the metamodel generated by a GP implementation purely based on error using the black points as training data set, while the smooth line represents the underlying true function ($z\,sin(z)$ in $[0, 3]$). It is clear that GP evolution was successful, as a metamodel with zero error on the training data set was found, but the generated metamodel is practically unusable for its poor generalisation ability.

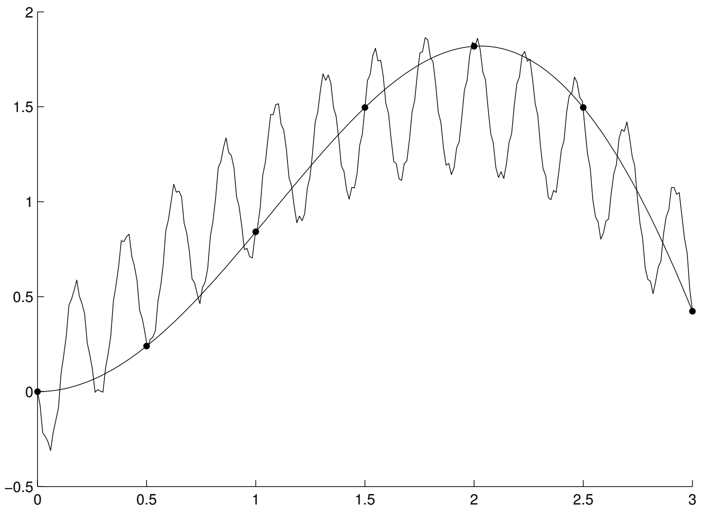


FIGURE 4.1: Example of GP metamodel with poor generalisation ability. The function to be approximated (smooth) and the metamodel (noisy) return the same response on the training data set (black dots)

Increasing the density of the DoE may help improve metamodel generalisation ability, but it does not represent a conclusive solution to the generalisation problem, let alone the computational overhead associated with it: there will always be a chance for GP to generate a metamodel noisy enough to interpolate the given data without providing a reliable approximation of the true underlying function on the whole design space. Runge's phenomenon proves indeed that even using regular polynomials the integral error of the metamodel evaluated on the whole design space can grow even if the number of (uniformly distributed) input samples is increased (Quarteroni et al. 2000, p. 239).

The tendency of GP to generate a metamodel that is accurate on the training data set but has poor generalisation ability is called *overfitting* (Langdon and Poli 1998a, De Jong and Pollack 2003, MacLean et al. 2005, Vladislavleva 2008). Although an appropriate selection of primitives may reduce the problem (Blickle 1996), overfitting is mainly the result of GP attempts to minimise metamodel error metric on the training data set.

The previous examples have shown that fitness function based purely on error does not lead GP to the discovery of the true underlying function. Yet, this is not an issue related to any intrinsic limit of GP potentiality, but a theoretical one. The problem of finding a mathematical expression of any mathematical structure minimising the error on a given input data is not well-defined: there are multiple models that could be assumed to be true underlying functions (see Fig. 4.1). In optimisation terms, the search performed by GP, when led uniquely by training error minimisation, has infinite solutions.

In order to use GP for metamodelling purposes, it is then necessary to bias the evolution towards the set of (finite) solutions that have the features that the true underlying function is assumed, expected or *desired* to have. A smooth behaviour for example is usually assumed for the true underlying function describing some natural phenomena, as done in the application shown in Section 6.4, Chapter 6. Keeping non-linearities to a minimum may be required for practical purposes, for example to ease metamodel analysis and optimisation. When instead metamodels are used to infer knowledge or to perform sensitivity analysis a reduced size (compactness) and interpretability are important (Sætrom and Hetland 2003, De Jong and Pollack 2003, Schmidt and Lipson 2009a).

A way to direct GP search to solutions of the desired form is to codify the desired properties through mathematical quantities, called *objectives*, and reformulate fitness function as a function of them as well as of training error. This idea is the basis of *multiobjective* GP.

4.3.1 Objective identification

A major challenge in the definition of GP objectives is to identify which genotypical property determines the appearance of a desired phenotypical trait (metamodel property). Similarly to what happens in nature, the relation between genotype and phenotypical traits in GP is not biijective and extremely complex, due to pleiotropy and polygeny (see Section 2.2.2.3, Chapter 2). Researchers have tried to identify GP genotype features that are linked to important properties in the corresponding metamodel.

Generalisation ability has been observed to be correlated with the size of GP individuals, with shorter expression having usually better generalisation ability than longer ones with the same training error (Zhang and Mühlenbein 1995, Nordin and Banzhaf 1995, Blickle 1996, Soule and Foster 1998ba, Iba and Terao 2000). According to Langdon et al. (1999, p. 168), there is no correlation between fitness and the size of a GP individual, so keeping GP trees' size as small as possible is a good practical rule to enhance generalisation ability. Smoothness is another appreciated property, but unfortunatley shorter solutions are neither necessarily smoother than longer ones nor exempt from the risk of over-fitting (Vladislavleva 2008).

Smoothness, low non-linearity, compactness and a reduced number of nested operations are also important to increase metamodel reliability and interpretability (Vladislavleva 2008, Schmidt and Lipson 2009a) and so, in the end, determine the success of a GP metamodel. These properties contribute to what Vladislavleva (2008) called *metamodel complexity*. Smits and Kotanchek (2004) favoured trees with lower depth to minimise the number of nested functions through the definition of the *expressional complexity*, sum of the number of nodes in all subtrees of a given tree. The first parameter aimed at measuring the order of non-linearity of a metamodel was introduced by Garshina and Vladislavleva (2004). Its definition takes into account the minimum order of the polynomial approximating a GP metamodel with a given precision. This approach however introduces a further metamodelling problem, the approximation of a metamodel generated by GP through a RSM technique, and considering that such further approximation has to be done for all the individuals in the population, it may dramatically slow down evolution (Vladislavleva 2008). A more efficient strategy was developed by Vladislavleva (2008). She introduced a set of rules to recursively compute a GP tree "order of non-linearity" from the order of non-linearity of its terminal and functional nodes. The approach, based on approximation through univariate Chebyshev polynomials, is more efficient than building a multivariate polynomial response.

A compact, interpretable and "physically grounded" mathematical structure is the most desired feature in a GP metamodel, at least when GP is required to provide theoretical insight into an unknown physical system (Keijzer et al. 2005, Schmidt and Lipson 2009a). With the aim of increasing interpretability and physical consistency of GP metamodels, Keijzer and Babovic (1999) introduced the concept of "dimensionally awareness" in GP. In Keijzer and Babovic's implementation physical units are assigned to each GP terminal. Newly generated trees featuring operations that are not physically consistent, for example the sum or subtraction of terminals having different physical dimensions, are penalised through the definition of an objective called *goodness of dimension*. Metamodels generated by Keijzer and Babovic's *dimensionally aware genetic programming* were reported to feature improved interpretability and to be less affected by excessive tree growth than standard GP (Keijzer and Babovic 1999, Keijzer et al. 2005).

4.3.2 Objective handling

The set of objectives used to bias GP evolution define a sort of metamodel "aesthetics", as they define more or less accurately the phenotypical traits desired in the solution. A typical issue in multiobjective GP is to find a way to harmonise the optimisation of each desired trait during the evolution, as it often happens that such features are not independent (pleiotropy and polygeny) and sometimes even conflicting. Penalising GP individual size excessively for example may reduce evaluation time and so speed up the evolution, but it may also limit genotype expressivity and compromise accuracy.

The strategies used in multiobjective GP to deal with the concurrent optimisation of multiple objectives are based on the typical multiobjective methods used in optimisation, especially in evolutionary algorithms (Fonseca and Fleming 1995, Zitzler and Thiele 1999, Laumanns et al. 2002, Deb et al. 2002, Marler and Arora 2004, Bonte et al. 2005). Common approaches are:

- bounded objective function method (the main objective is identified and the others turned into constraints);
- weighted-sum method (also called parametric approach);
- lexicographic method;
- Pareto approaches.

An introduction to the methods listed above can be found in Marler and Arora (2004). In Section 4.5 a few applications of these methods to GP using specific objectives will be reviewed. For the interest Pareto optimality has attracted in the whole class of evolutionary algorithms, a general introduction to Pareto approaches will be presented in the next section.

4.3.2.1 Introduction to Pareto approaches

Pareto approaches are used when it is not possible to determine “the relative importance of the objective functions or desired goals before running the optimization algorithm” (Marler and Arora 2004, p. 370). Pareto-based multiobjective optimisation aims at finding a subset of points or vectors, called *Pareto set*, that represents the best compromise in terms of objectives, considered all equally important. The Pareto set is composed of mathematically equivalent solutions, so the best point(s) cannot be selected until the final user gives some indication on the relative importance of the objectives (*a posteriori articulation of preferences* (Marler and Arora 2004)). The advantage of Pareto approach is that no additional optimisation runs have to be performed to identify new optimal points objective preferences change.

Pareto optimality has been used extensively in the field of evolutionary computation, GP included, as it is able to deal with heterogeneous objectives, non necessarily comparable using standard inequalities, and to identify in a population of vectors a set of equally optimal but diverse individuals. In EAs, maintaining a set of equally good individuals that span all the possible combinations of the objectives helps increase genotype variability, which in turn reduces the risk of premature convergence, as seen in Chapter 3. This explains why many selection methods and genetic operators based on Pareto dominance (Fonseca and Fleming 1995, Zitzler and Thiele 1999, Keijzer and Babovic 1999, Vladislavleva 2008) as well as on other Pareto performance metrics (Fonseca and Fleming 1995, Smits and Kotanchek 2004) have been developed in GAs and GP.

A common challenge in evolutionary algorithms is to ensure the Pareto front is actually reached and explored uniformly. The evolution may indeed disrupt the uniformity of the distribution along the trade off surface, hindering the exploration of some potentially interesting combinations of objectives or converging to a single region of the Pareto front (*genetic drift* Fonseca and Fleming (1995, p. 7), De Jong and Pollack (2003)). Strategies as *sharing*, *crowding*, *niching* based on Pareto dominance and *restricted mating* have therefore been developed for this aim (Fonseca and Fleming 1995, Zitzler and Thiele 1999, Deb et al. 2002, Laumanns et al. 2002, Kroo 2004).

In Section 4.5.4.4 more details will be given on Pareto GP implementations. First, however, it is worth to identify the most important objectives in determining GP evolution success through the analysis of the causes of premature convergence.

4.4 Premature convergence, bloat and neutral code

The problem of premature convergence has been introduced in the previous chapter, where it was described as a state when evolution has reached a local optimum of the fitness landscape, and further evolution is not able to improve that fitness. In Section 3.1.4, Chapter 3, loss of population diversity caused by unbalanced selection strategies was identified as a major cause of premature convergence, both in GA and GP (Koza 1992, Banzhaf et al. 1996, Affenzeller and Wagner 2004). On the other hand, as introduced in Section 3.1.5.5, Chapter 3, crossover and mutation also have a major role in preventing GP evolution from being stuck in suboptimal solutions, through the exploration of new and diverse trial solutions. Selection strategies and genetic operators appear therefore intricately related to the problem of premature convergence.

The elements described so far do not allow to formulate any hypotheses regarding how the selection and the reproduction stages may interact to cause a progressive reduction in GP exploratory power. In the next section it will be shown that the consideration of further GP phenomena called *bloat* and *neutral code* are required to reach a deeper understanding of the causes of premature convergence.

4.4.1 Bloat

Bloat is an important phenomenon in genetic programming. One of the clearest definition is provided by Poli et al. (2008, p. 101), who define bloat as "program growth without (significant) return in terms of fitness". Blickle (1996) and Langdon (1998) refer to bloat as a generic increase in individual size from one generation to the other not linked to any performance improvement in the population. Other similar definitions are provided by Soule and Foster (1998a), Langdon et al. (1999), Banzhaf and Langdon (2002), De Jong and Pollack (2003) and Haeri et al. (2012), all sharing the concept that bloat is a condition in which despite a growth in GP individual size fitness values slightly improves or it does not at all. The expression "fitness stagnation" is usually associated with the bloat phenomenon. It is important to note that code growth is often necessary to improve fitness, in particular during the first generations of a GP run (Soule and Foster 1998b). Zhang and Mühlenbein (1995), Langdon and Poli (1998a) and Schmidt and

Lipson (2009a) posed the interesting problem of computing the "appropriate" size of a solution of a problem, beyond which any further increase is bloat[1].

Bloat has been observed to affect GP regardless the implementation or representation used and independently from the task GP has been applied to. Bloat however is not specific to genetic programming, as it was also reported in artificial neural networks (Luke and Panait 2002a). Langdon (1998), Langdon and Poli (1998b), Soule and Foster (1998a) and Langdon et al. (1999) suggested that code growth without substantial fitness improvement is inherent in any search technique that uses a variable-length representation. Bloat can be seen as the result of the increased redundancy due to the absence of limits on the size of the individuals (Langdon and Poli 1998a). Redundancy is the possibility to build many genotypes with the same phenotype (Banzhaf 1994): although an intrinsic feature of GP, it is enhanced by the possibility of extending individual genotypes with portions of genetic code that have no effect on fitness.

If there is a general consensus on what bloat generally implies, no agreement has been reached on how to recognise bloat quantitatively. In other words, to the best of the author's knowledge, a parameter that could be used to detect whether a GP run is "bloating" or not has not yet been defined. The most common approach is to monitor average individual size and compare it with the average population fitness. De Jong and Pollack (2003) observed an exponential increase in the average number of node evaluations with generations in absence of bloat countermeasures in a multiobjective GP not using mutation. In Soule and Foster (1998b) and Soule and Foster (1998a) the average tree size in a GP implementation not using mutation was reported to increase linearly with generations. In Bleuler et al. (2001), average tree size was also observed to grow approximately linearly with the generations (standard GP with subtree crossover, point mutation for even 5-parity problem). In the symbolic regression of quintic and sextic polynomials performed by Langdon et al. (1999, p. 184) and Langdon (2000) using a tree-based GP with tournament selection, one-child crossover, no mutation, a linear increase in the average depth of binary trees was observed, at about one level per generation. In both cases it was predicted that a quadratic or sub-quadratic growth in size may be expected. Although all the mentioned experiments were performed with GP using only crossover, bloat can occur even with other genetic operators, like subtree mutation (Langdon 1998).

[1]In this sense, referring to code growth *with* return in fitness as to "structural bloat", as done in Gelly et al. (2006), or "fitness independent bloat", as done in Langdon (1998), seems unappropriate given the negative undertone the word has in GP community. The expressions "necessary code growth" or "healthy code growth" or "code growth due to training" appear to be more correct.

Bloat describes a state of fitness stagnation of a GP evolution and is, necessarily, a condition to be retarded as much as possible. However, also a disproportionate rate of growth, either in tree depth or size, with respect to fitness improvement rate is detrimental to GP exploration. Langdon et al. (1999) and MacLean et al. (2005) observed that if size is not kept under control (for example through a *length* bias or penalisation), GP best individual may suffer from a sudden corruption of generalisation ability, in other words overfitting, if the evolution is let to continue after the individual has reached its "maturity". These findings back the idea expressed in Zhang and Mühlenbein (1995), Langdon and Poli (1998a) and Schmidt and Lipson (2009a) that for a certain problem an appropriate (optimal) size of the individuals may exist.

A disproportionate rate of growth has also serious practical implications. De Jong and Pollack (2003) showed that, in the absence of countermeasures, the number of GP tree node evaluations on average increase exponentially with generations. This rate of growth may put to the test even the most powerful computers, or at least the ones commonly used in universities and laboratories, considering that the computational overhead due to selection, genetic operations and evaluation is a function of tree size (Van Belle and Ackley 2002, De Jong and Pollack 2003). A typical problem when running GP experiments with large GP populations is to provide enough RAM memory to allow for individuals storage and evaluation (Soule et al. 1996, Soule and Foster 1998b, Langdon and Poli 1998a, Iba and Terao 2000, Bleuler et al. 2001, Luke and Panait 2002a, Van Belle and Ackley 2002). To avoid incurring computational costs that cannot be sustained, a computational budget can be assigned to the evolution, but in this case disproportionate code growth can rapidly squander it evaluating individuals larger than average but with poor fitness value, limiting GP search efficiency.

Finally, excessive code growth may raise problems in using or implementing the solution generated by GP. In symbolic regression a large model usually has a reduced interpretability (Van Belle and Ackley 2002). An excessive solution size can represent an issue also in other applications. A large logic circuit generated by GP, for example, may be hard to implement in hardware. In general, larger solutions are more expensive to produce (Zhang and Mühlenbein 1995).

In conclusion, fighting bloat and excessive code growth is paramount for GP efficiency so size control should be considered as one of the main objective in multiobjective GP. Different approaches can be used to reduce size growth. For example, through a simple size penalisation, or, assuming that an appropriate size of the solution can be identified, penalising size if it is beyond this assumed threshold. These strategies however do not address the causes of bloat, they simply try to limit its effects. In the next section it will be shown that the causes of bloat have to be sought in the behaviour of portions of genotype that have no effect on fitness, or *introns*.

4.4.2 Introns

In the natural world there are many examples of "amounts of apparently unexpressed pseudogene DNA" (Van Belle and Ackley 2002, p. 152) or "genetic code that does not apparently express itself in the individual produced by the genome", as reported by Nordin et al. (1996) referring to the studies of J. D. Watson, co-discoverer of the structure of DNA in 1953. Segments of organisms' DNA that do not explicitly contribute to protein synthesis are usually referred to as *introns* (Langdon et al. 1999). Opposed to introns are *exons*, which are portions of genetic code that do contribute to the generation of proteins (Langdon et al. 1999).

Introns are mostly found in high complexity organisms, as eukaryotes, whereas in simple ones, as prokaryotes, non functional genetic code is almost absent (Nordin et al. 1996, Soule et al. 1996). Soule et al. (1996, p. 216) report that "80% - 90% of human genome does not code for functional proteins, even though some of this DNA has a structural function". Neutral parts of genetic code are therefore apparently useful in nature and serve a specific purpose.

Researchers in evolutionary computation have borrowed the terminology used in biology and genetics and they applied the terms *intron* and *exon* to describe portions of computer code (Soule and Foster 1998a). In EAs an *intron* or *neutral code* is defined as a portion of an individual genotype that has no effect on the individual fitness value, whereas an *exon* (or *effective code*) is a portion that contributes to it (Nordin et al. 1996). Introns have been observed in ES, EP and GA (Soule and Foster 1998a), but in GP their generation and growth is sustained by the variable-size representation (Altenberg 1994, Nordin et al. 1996, Langdon and Poli 1998a, Miller and Thomson 2000, Brameier and Banzhaf 2007, Poli et al. 2008). To understand how introns affect GP evolution, it is important to make a distinction between portions of genetic code that can possibly contribute to an individual's fitness as opposed to portions that cannot under any circumstances (Soule and Foster 1998a). An intron classification is introduced in the next section.

4.4.2.1 Classification

The representation chosen, the building data set and the primitives available to GP contribute to the range of introns that GP can spontaneously generate during the evolution. As seen in Table 1.2, GP can be used for many tasks, in which the primitives or the presentation used may encourage the formation of specific kinds of introns that do not exist in GP applied to symbolic regression. However, in author's opinion, knowing both what *can* and what *cannot* happen depending on the specific parameters of a GP implementation may be beneficial for the development of a GP implementation for symbolic regression tasks.

The classification that is suggested in the following has been obtained merging the definitions and observations reported in Iba and Terao (2000) for tree-based GP and Brameier and Banzhaf (2007) for linear GP[2]. Three main different types of introns can then be defined, divided in two classes:

- *non-executed code*: genotype portions that are not executed or evaluated. Their existence is determined by the availability of functional primitives that allow to skip the evaluation of some part of the genotype, like conditional statements (for example the functional primitive $if \dots then$) or other logical operations or control statements. For this reason, this kind of introns are often encountered in GP applications to control law or algorithm synthesis. Two subclasses of non-executed code can be defined. *Structural* or *syntactic introns* (Brameier and Banzhaf 2007, Iba and Terao 2000) are parts that are *never* executed, regardless the building data set used. *Effective introns* (Soule et al. 1996, Iba and Terao 2000) are instead code segments that could theoretically be executed but they are not due to the specific selection of the building data set (during individual evaluation the operating conditions to activate them are not encountered).

- *non-functional code*: genotype portions that are executed but do not affect fitness value. They are also called *semantical introns* (Brameier and Banzhaf 2007). Semantical introns are the only intron type that can be encountered in GP used for symbolic regression tasks, as control statements are not used. The appearance of

[2]the linear GP representation will be introduced only in Section 3.3.

semantical introns is made possible by the neutral property of the functional primitives used. An example is provided by the following expression:

$$5x + (3 - 3) \tag{4.2}$$

whose response does not depend on the subtree $(3-3)$ due to the neutrality property of addition. Nordin et al. (1996) observed that some genotype portions can happen to be neutral on specific building data sets, but are not neutral once evaluated on a different input data set. An example of this behaviour is illustrated by the GP individual:

$$5x + (sin(x)) \tag{4.3}$$

in which the subtree $(sin(x))$ may appear as a semantical intron if the building data set is $[k\pi \mid k \in \mathbb{Z}]$. This type of "apparent" introns are a threat to generalisation ability: a possible strategy to counter them will be shown in Section 5.3.0.6, Chapter 5. Further classifications of semantical introns have been put forward. Nordin et al. (1996) considered the number of instructions/operations required to produce a neutral code portion (*first order* and *second order* introns). Soule and Foster (1998b), Langdon et al. (1999) referred to *viable* (neutral fragment which by modification of a single subtree become effective on fitness - for example the portion $(1-(4-3))$ in $X+(1-(4-3)))$ or *inviable* nodes (neutral fragment which cannot become effective on fitness even by its modification - for example Z in $Y+(0*Z)$) and *operative* and *inoperative* nodes to describe different categories of neutral code. In this thesis this classification is not used.

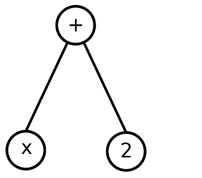
The classification introduced so far refers only to the introns generated spontaneously during a GP evolution. Nordin et al. (1996) called them "implicit" introns to distinguish them from "explicitly defined" introns, which are generated and inserted in the population by an external agent. A brief explanation on how this kind of introns has been used to improve GP search will be given in Section 4.5.3.

4.4.3 Introns role in evolution: interaction with crossover and mutation

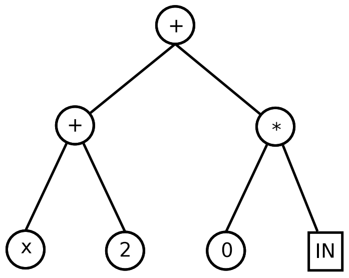
Introns provide a mechanism to explain bloat, as they make GP individuals bigger without changing their fitness value. In this sense, introns cause bloat and bloat is strictly linked with the appearance of introns, regardless the GP representation used (Blickle and

Thiele 1994, Nordin and Banzhaf 1995, Langdon and Poli 1998a, Soule and Foster 1998a, Bleuler et al. 2001, Banzhaf and Langdon 2002, De Jong and Pollack 2003). Although this mechanism has been identified, it has not yet been explained how introns are generated by GP and how they should be dealt with to improve GP performance. More specifically, if bloat is caused by introns, is intron elimination beneficial for a GP evolution? Is it possible to indiscriminately remove them? If not, when and how to remove them? These questions are not trivial and address a fundamental issue, whether introns are useless or have a function in genetic programming.

Research has shown that introns function changes during a GP run. To study intron dynamic behaviour, researchers have used the adjective *constructive*, *destructive* or *neutral* to describe genetic operations that produce offspring of respectively better, worse or same fitness as the parents (Nordin et al. 1996, p. 6). The idea that crossover could be a mostly destructive operator and introns could provide a defense mechanism against GP individuals disruption was put forward not long after the appearance of GP (Altenberg 1994). The symbolic regression experiments on a second order polynomial performed by Nordin et al. (1996) with both linear and tree-based GP confirmed the theory, which is today widely accepted (Langdon and Poli 1998a, Bleuler et al. 2001, Brameier and Banzhaf 2007). As seen in Section 3.1.5.2, Chapter 3, the disruptive effect can be explained by the fact that crossover, at least the standard version, does not take into account the context in which swapped subtrees are inserted (Bleuler et al. 2001). Introns provide a defense against this destructive effect modifying crossover hit rate[3] of the nodes in a GP individual (Altenberg 1994, Nordin et al. 1996). Fig. 4.2 helps explain how the defense mechanism works: if the crossover point is selected by uniform selection among the nodes excluding



(A) original individual



(B) individual protected by an intron

FIGURE 4.2: Example of GP individual protected by neutral code

the root, the probability of disrupting the individual in Fig. 4.2A is 100%, as for sure

[3]see Section 3.1.5.2, Chapter 3.

one of the two connections to the root node will be disrupted. If instead the same tree is implanted in a larger tree containing an intron (IN) as in Fig. 4.2B, the probability of disruption decreases to a maximum of 83.3% (5/6), as the intron can be replaced without consequences to the rest of the individual (and the intron may be composed of other nodes). The same protection mechanism, based on hit rate modification, substantially holds for subtree and point mutation (Langdon 1998, Langdon et al. 1999). Results in Banzhaf and Langdon (2002) suggest that introns protect individuals from deleterious changes due to point mutation, although a simplified representationless GP model was used to reach this conclusion. Other evolutionary algorithms, like Grammatical Evolution (Ryan et al. 1998), also benefit from the protection provided by introns.

A major drawback of intron presence is that they could become too protective, constrasting therefore exploration in favour of exploitation (Langdon et al. 1999). In other words, introns can protect GP individuals from crossover and mutation even when they may be constructive (Langdon 1998), encouraging instead neutral crossover. A correlation between the increase of the average intron size during the last part of a GP run and the rise in neutral crossover frequency, and a concurrent drop in both destructive and constructive crossover, is well documented in Nordin et al. (1996), who observed it in both linear and tree-based GP used for symbolic regression of a second order polynomial (fitness proportionate selection). It can be inferred that introns, if no parsimony measures are put in place to counter them, grow to a point that they lock the optimal individual found making it harder by the generation for genetic operations to perturb the evolution, eventually leading to fitness stagnation. Therefore, as soon as the intron conservative protection starts to take over, the training stage has to be considered concluded (Nordin et al. 1996, Banzhaf et al. 1996, Iba and Terao 2000). The same excessive protection mechanism was observed by Blickle and Thiele (1994). They concluded that "the probability to escape a potential local optimum decreases with time" (Blickle and Thiele 1994, p. 37), as promising new-born individuals with low proportion of neutral code are less likely to survive than individuals with high proportion of neutral code. Similar conclusions were reached by Langdon et al. (1999) who used GP to solve the artificial ant problem. So, recurring to a mechanical similitude, the effect of neutral code on evolution can be compared with the behaviour of a flywheel: the bigger the neutral code portion, the harder is to improve fitness (Langdon et al. 1999, p. 186), as the bigger the flywheel is the harder is to change its angular speed. Introns can be considered as a measure of the "inertia" or resistance to change in a GP population.

Despite the long-term detrimental effect, protection offered by introns may be needed in particular stages of the evolution. As already hypothesised in Section 3.1.5.5, Chapter 3, a GP run can be roughly split in three stages, which were called youth, maturity and stagnation. The results described by Nordin et al. (1996) confirm this idea, and provide quantitative means to define these stages. It has been shown indeed that stagnation can be recognised by a sudden fall in destructive and constructive crossover frequency and an abrupt increase in intron average size. Youth instead can be characterised as a stage in which constructive and destructive crossover frequencies have an irregular behaviour but are on average higher than during stagnation and the average intron size slowly increases, experiencing a jump when the best individual is found (Nordin et al. 1996, p. 10). Such jump in intron average size may be the evidence that introns are useful in the early generations of a GP run, protecting as sort of computational "incubators" promising but small and not fit individuals from the harsh environment. The beneficial role of introns on GP evolution is also indirectly suggested by Banzhaf (1994). Recalling Kimura's neutrality theory of evolution, Banzhaf (1994) support the idea that *any* form of redundancy, or the possibility to generate many different genotypes corresponding to the same phenotype, is key to enhance variability, as selection pressure is not able to destroy different individuals if they have the same fitness. According to Banzhaf (1994) introns, increasing GP redundancy, are therefore beneficial to global exploration: increasing variability in the population they provide an escape route from locally optimal region of the design space.

In conclusion, introns role on GP evolution is dynamic: introns can increase the probability of the individual to survive but at the same time reduce the efficiency of GP as optimisation process (Blickle and Thiele 1994). The effect of neutral code appears also to depend on its diffusion, that is the ratio between effective and neutral code in the population. The mechanisms that bring to the formation of introns become therefore important to describe the relation between bloat and introns. In this regard, according to Banzhaf et al. (1996) (for linear GP) and Soule et al. (1996) (for tree-based GP), introns are mainly generated by crossover and subtree mutation, due to the statistical advantage for the longer of two individuals to have the same fitness value as the parents. Point mutation has been identified as a "switch" operator that can turn an intron into working code and viceversa (Soule and Foster 1998a). Point mutation appears to prevalently turn neutral part of genetic code into effective code or to break blocks of neutral code (Banzhaf et al. 1996). This functionality adds to the already important function of reintroducing lost alleles (Section 3.1.5.5, Chapter 3).

4.4.4 Bloat theories

Three main theories regarding how introns are generated and result in bloat have been formulated in the GP community (Soule and Foster 1998ba, Langdon et al. 1999, De Jong and Pollack 2003, Gelly et al. 2006) and are indicated by the following expressions:

- protective role of introns;
- fitness causes bloat;
- removal bias theory.

4.4.4.1 Protective role of introns

The theory (Altenberg 1994, Nordin and Banzhaf 1995) does not address the causes of intron formation, but emphasises the fact that intron growth is a phenomen that sustains itself. Precedently it has been shown how the presence of introns reduces the probability of the individual to be disrupted, as the crossover point may be selected in the neutral region (Nordin and Banzhaf 1995, Ryan et al. 1998). The probability of disruption then decreases as the size of the neutral part increases (Soule and Foster 1998b). This observation finds a mathematical proof in Blickle and Thiele (1994, p. 34), where it is shown that the probability that crossover leaves unchanged the effective region of a given GP tree containing neutral code (called *redundant* code) is linear with the proportion of neutral nodes in the parent individual. Concurrently, it has been observed that in case no length bias are introduced in the evolution, on average crossover and subtree mutation produce a size increase in the offspring (Langdon and Poli 1998a). These two elements combined make population vulnerable to bloat and unable to stop it if measures to curb size growth are not in place.

The appearance of the smallest intron is sufficient to trigger bloat: the individual that contains it, which will be called “carrier”, will be more likely to survive genetic modifications untouched in its effective region than other individuals that do not have introns. Its neutral part on average will be expanded by crossover and subtree mutation, so the carrier will keep its original fitness value and the probability that its effective region is disrupted will decrease as its overall size increases. This mechanism explains the typical exponential growth in average introns size, observed for example in Nordin et al. (1996).

4.4.4.2 “Fitness causes bloat”

The theory is based on the general assumption that, if a variable length representation is chosen, long genotypes have a statistical advantage on short genotypes of the same fitness, as there are (far) more longer programs than shorter ones that result in the same fitness (Langdon and Poli 1998a, Langdon 1998 2000, Banzhaf and Langdon 2002). So the evolution is steered towards larger versions of the same solution simply because they are more abundant (Soule and Foster 1998b, Langdon et al. 1999).

Specifically, neutral code further expands the number of syntactically different but semantically equivalent individuals (Soule and Foster 1998b). As a result, in the absence of size bias, individuals composed of larger regions of neutral code are more likely to be sampled by GP than individuals with smaller introns (Langdon et al. 1999, Langdon 2000).

As noted in Soule and Foster (1998ba), this theory holds for unbiased search techniques using a discrete variable-length representation. It does not assume neither the evolutionary mechanisms (genetic operators) nor the existence of introns, being based only on the distribution of the solutions in the search space. In this sense, it is more general than the other two bloat theories (Langdon et al. 1999, pag. 170). It also predicts the occurence of bloat in search techniques not based on populations (Langdon 1998, Langdon et al. 1999).

4.4.4.3 Removal bias theory

The removal bias theory (Altenberg 1994, Soule and Foster 1998b, Langdon et al. 1999) was developed for tree-based GP and it assumes the mostly destructive effect of subtree crossover and subtree mutation. In this sense is less general than the other two theories.

Inviable code, or neutral code that even if modified cannot change fitness value, forms subtrees that are mostly concentrated around the leaves (terminals) in a typical tree (Soule and Foster 1998b). Assuming that the nodes have all the same probability to be selected as crossover or mutation points (this is not always the case, as the selection may be biased on purpose), nodes close to the terminals are more likely to be selected as crossover or subtree mutation points. At the same time, the small subtrees rooted in these nodes are more likely to be inviable code than larger subtrees rooted at lower depths.

As a result, the crossover and mutation point selection is biased towards the small and inviable subtrees rooted near the terminals. The replacement of larger subtrees would be penalised, as larger subtrees are more likely to be made of viable code (Soule and Foster 1998b) and on average altering viable code has a destructive effect on individual fitness (context issues). So on the one hand the choice of the subtree to be replaced is biased towards small and inviable subtrees. On the other hand, the replacing subtree is not subjected to similar size constraints. As for crossover, a large subtree is most likely to be implanted in an inviable node close to the parent's terminals, otherwise it would most probably penalise the offspring. In subtree mutation the generation process is not biased towards the creation of small replacing subtrees.

The overall effect of the asymmetry between the size of the subtree to be replaced and the replacing subtree (Altenberg 1994) is an average tree size increase with no change in fitness, as replacing subtrees are implanted in inviable regions of the parents. The presence of inviable code then triggers the accumulation of further inviable code, and the mostly disruptive crossover produces an evolutionary disadvantage for new-born smaller trees.

The previous theories highlight different causes of bloat. Given the complexity of phenomenon, there is a general consensus in the GP community that the described mechanisms coexist in a typical GP run (Soule and Foster 1998ba, Langdon and Poli 1998b, Langdon et al. 1999, Banzhaf and Langdon 2002).

4.5 Strategies to fight bloat

So far it has been shown that the control of introns size throughout a GP run is paramount to increase GP efficiency, as depending on their proportion in a GP population introns can provide a protection of promising individuals against disruption (introns as *incubators* of new solution) or prevent further evolution and determine premature convergence (introns as population "inertia" against improvement). The positive effect of reducing introns is acknowledged by many researchers (Blickle and Thiele 1994, Nordin et al. 1996, Banzhaf et al. 1996, Soule et al. 1996, Iba and Terao 2000). As expected, keeping individual size under control not only reduces the computational effort associate to a GP run but also increase the accuracy of the best individuals and GP success rate (Soule et al. 1996, Soule and Foster 1998a, Bleuler et al. 2001, Van Belle and Ackley 2002).

Bloat theories allow to identify in the following mechanisms the main causes of intron formation:

1. predominantly destructive character of crossover and subtree mutation (protective role of introns and removal bias theory);
2. size bias in subtrees to be replaced but not in the replacing subtree (removal bias theory);
3. equal selection probability of all nodes in a tree, favouring the selection of nodes close to the terminals (protective role of introns and removal bias theory);
4. probabilistic advantage of larger individuals ("fitness causes bloat" theory).

Consequently, subverting these basic mechanisms can then lead to the development of strategies to prevent introns appearance. In the following sections a survey of the main anti-bloat strategies is presented. Some approaches contrast the appearance of introns altering the basic mechanisms described by bloat theories, others instead focus on the direct elimination of introns or penalisation of bloated individuals. More specifically, the strategies that will be described can be divided in the following classes:

- direct elimination through code editing
- genetic operator adjustment
- explicitly defined introns
- multiobjective approaches
- avoidance of destructive crossover (or mutation)

4.5.1 Direct elimination through code editing

The most direct way to reduce intron size is to remove them from GP individual through editing (Soule et al. 1996, Iba and Terao 2000, Blickle and Thiele 1994, Blickle 1996, Ryan et al. 1998). Intron direct removal requires as a first step their identification. Iba and Terao (2000) and Blickle (1996) used a strategy based on the *marking* algorithm (Blickle and Thiele 1994, Blickle 1996): they assigned a counter variable or flag to each node of a tree. If during tree evaluation the value stored in the counter is not updated it means that the corresponding node is not used, so the subtree rooted in that node is a structural intron. The neutral subtree can then be removed or replaced by a randomly generated terminal node (*deleting crossover* - Blickle (1996)).

Intron elimination through code removal presents however a few limitations. Firstly, identifying neutral code implies an additional computational cost (Nordin et al. 1996, Collet et al. 2000, Aichour and Lutton 2007), which is proportional to individual size and to the number of individuals in the population. Moreover, not all introns can be easily detected (Blickle 1996). Soule et al. (1996) and Langdon and Poli (1998a) acknowledge the difficulty of reliably detecting neutral code. More specifically, the strategy used by Iba and Terao (2000) and Blickle (1996) to identify introns is able to find only structural and effective introns, so it appears of no use in GP used for symbolic regression, as affected only by semantical introns. Secondly, removing introns every generation partially reduces introns growth rate but, as noted by Soule et al. (1996), this rate remains exponential. Interestingly, Soule et al. (1996) also observed that introns growth rate retains a memory of the introns size at the beginning of the evolution: editing the first generation slightly reduces introns growth rate.

4.5.2 Genetic operator adjustment

The following approaches have mainly been inspired by the removal bias theory (Section 4.4.4.3).

4.5.2.1 Altering subtree crossover/point mutation balance

If bloat is caused by the asymmetry of subtree crossover and subtree mutation, as assumed by the removal bias theory, then increasing point mutation rate will reduce bloat. Experiments on linear GP on classification tasks performed by Banzhaf et al. (1996) show that increasing point mutation rate indeed helps reduce the percentage of introns in the population, retard fitness stagnation, prolong evolution and increase the accuracy of the best individual. These results prove the validity of the theory that considers introns as population "inertia" towards variation. They also support the theory that mutation primary role is to break blocks of neutral code and to introduce alleles that may have been lost due to selection pressure.

4.5.2.2 Constraining crossover point selection to effective regions of the genotype

The asymmetry of subtree crossover and subtree mutation assumed by the removal bias theory can be reduced if the possibility of selecting neutral (inviable) nodes as crossover points is eliminated. In Blickle and Thiele (1994) a *marking* algorithm is proposed to identify the nodes that are not executed/evaluated and crossover point selection is then restricted to the nodes that are effective, with the aim of increasing the probability of a constructive crossover (*marking crossover*). As this technique relies on the same algorithm to detect introns described in Section 4.5.1, the same drawbacks there highlighted hold (only structural introns are detected). In Collet et al. (2000) the idea of constraining the selection of the subtree to be replaced is further developed. Each subtree composing the parent is evaluated and the subtree giving the worst contribution to parent's fitness value[4] is selected for replacement with a randomly generated tree.

Constraining the crossover point selection to non neutral regions of the genotypes is an attempt to increase the probability of performing a constructive crossover. However, these approaches are only able to avoid neutral crossover. There is no guarantee that swapping active subtrees and implanting them in active regions of the parents leads to a constructive crossover. Probably it is for this reason that this class of methods is seldom used (Langdon and Poli 1998a, p. 45).

4.5.2.3 Adjusting hit rates

Bloat according to the removal bias theory and the "protective role of introns" theory is caused by the higher probability of selecting terminal nodes as crossover points if a random uniform sampling strategy is used. Biasing the selection is then a simple solution to contrast bloat. One of the most commonly used approaches is to assign different hit rates to inner (functional) nodes and terminal nodes (Luke and Spector 1996 1997 1998, Brameier et al. 1998, Langdon et al. 1999, Langdon 2000, Vladislavleva 2008). For example Koza (1992) suggests setting the hit rate of functional nodes to 90%, using instead 10% for terminal nodes.

Adjusting hit rates reduces the protection mechanism provided by introns, making the population more "reactive", or less resistant to change. Nonethless, as in the previous approach, reducing the probability to select a neutral node as crossover point is no guarantee that the offspring have better fitness than the parents: swapping two effective subtree most of the times results in worse offspring.

[4]through this strategy is therefore possible to detect semantical introns.

4.5.2.4 50%-150% fair mutation and size-fair subtree mutation

According to the removal bias theory, subtree mutation tends to increase the size of the parent as the selection of the tree to be replaced is biased towards small subtrees, whereas the random subtree generation is not. The introduction of a dependency between the sizes of the subtree to be replaced and the subtree to be randomly generated can then tackle bloat. This strategy also reduces the probabilistic advantage of longer programs assumed by the “fitness causes bloat” theory. The correlation between the size of the subtree to be replaced and the subtree to be inserted assumed by *50%-150% fair mutation* (Langdon 1998, Langdon et al. 1999) and *size-fair mutation* (Langdon et al. 1999) is based on these concepts.

4.5.2.5 Size-fair and homologous subtree crossovers

Size-fair crossover and *homologous crossover* (Langdon (2000) for tree-based GP, Nordin et al. (1999) for linear GP) are improved versions of the standard subtree crossover, able to recreate in the offspring, to some extent, the context that swapped subtrees had in parents. The introduction of a size and depth correlation between parents and offspring, which is implicit in the concept of *context*, reduces the probabilistic advantage of larger solutions (“fitness causes bloat” theory) and the asymmetry of the crossover operator (removal bias theory).

Both strategies were reported to consistently curb bloat in a symbolic regression problem performed using a tree-based GP, to the extent that the rate of growth of the average individual size is reduced to an approximately linear trend with generations instead of the quadratic of sub-quadratic behaviour observed with standard crossover (Langdon 2000). Langdon (2000) however reported that the search success rate (number of correct solutions found out of the number of runs) was however not improved with respect to standard GP, in some symbolic regression cases even slightly reduced (Langdon 2000, p. 106). Probably the reduced efficiency can be explained by the excessive aggressiveness of the two new operators. Imposing a strong bias on size may hinder the formation of “incubator” introns (see Section 4.4.1) in the early generations of a GP run (*youth* stage), reducing therefore the protection towards small but promising individuals which could boost search efficiency. This effect is strengthened in case the initial population has a small average size.

4.5.2.6 Different mutation operators

The use of many different mutation operators introducing a size bias in the offspring is another possible countermeasure to bloat. Sims (1993) used for example more than five kinds of mutation. Mutation frequencies were adjusted to make size or complexity reduction slightly more likely than code growth, reducing bloat.

A general drawback of the previous approaches, which attempt to reduce the asymmentry of genetic operators and to affect the interaction between introns and genetic operators, is that they necessarily have to be developed for the specific representation used (Bleuler et al. 2001). This means that these solutions are not general countermeasures to bloat (Langdon et al. 1999).

4.5.3 Explicitly defined introns

Introns provide GP individuals with protection against disrupting genetic operations ("Protective role of introns" theory). As it has been shown, such protection can increase but also decrease the efficiency of the search, according to the proportion of neutral code in the population. Artificial introns, not generated spontaneously by GP, can then be used to vary this proportion with the aim of retarding premature convergence.

Artificial introns were first used in GAs, according to Altenberg (1994). Nordin et al. (1996) extended the approach to linear GP, calling this kind of introns *explicitly defined introns* (EDIs). Each EDI is given a weight, which correponds to the probability of its root node to be selected during crossover. Changing the weight is then possible to increase or loosen the protection offered by EDIs to the individual that hosts them.

The experiments performed by Nordin et al. (1996) and Blickle (1996) on symbolic regression problems did not produce conclusive results. Moreover, the complexity of the strategy may discourage the practical use of EDIs (Blickle 1996).

4.5.4 Multiobjective approaches

In Section 4.3 it has been shown that using a fitness function based uniquely on an error metric (defined on the building data set) is not a good criterion to direct a GP evolution, as it most probably leads to a solution with poor generalisation ability. To increase the probability of finding a "good" solution, the additional features that the ideal solution is expected to have need to be converted into quantitative objectives and fitness function changed to a multiobjective formulation.

The analysis presented in the previous sections has shown that reducing bloat is determinant for improving GP efficiency. A few stategies derived from the main bloat theories have been described but, due to the specific mechanisms they tackle, they result in countemeasures of limited validity (see Section 4.5.2).

In this section a different class of anti-bloat approaches are presented, called multi-objective approaches. They curb disproportionate code growth a posteriori, as they do not aim at tackling the mechanisms that lead to bloat, but penalise bloated individuals instead. They hence represent a more general solution to bloat, independent from GP representation and genetic operators, and they are effective against both structural and semantical introns.

The simplest indicator of bloat is individual size. Hystorically, size has always been used as a second objective, after the error metric, in multiobjective GP implementations to control size growth in GP evolution. There are no theoretical bounds on the number of objectives, however. In the following a survey of the application to genetic programming of the general multiobjective optimisation methods introduced in Section 4.3.2 is presented.

4.5.4.1 Transforming objectives into constraints

One of the most commonly used strategies to tackle disproportionate code growth is to impose an upper bound on the maximum size (Blickle 1996, Langdon and Poli 1998a) or depth (*maximal depth restriction* - Koza (1992), Whigham (1995), Soule et al. (1996), Luke and Spector (1997 1998), Van Belle and Ackley (2002)) of the individuals produced by genetic operations. Koza (1992) for example in his experiments imposed that crossover could not produce offspring of depth larger than 17.

This strategy poses a few problems. To begin with, before the size or depth limit is reached, code growth occurs unsuppressed (De Jong and Pollack 2003). Secondly, the maximum size or depth has to be selected by the user. The existence of an "appropriate" size of the solution has been previously assumed (see Section 4.4.1), but it is in general difficult to guess (Zhang and Mühlenbein 1995). A guess about such appropriate size introduces therefore a bias: a size limit too tight undermines expressivity and diversity in the population (Blickle 1996), whereas a limit too loose does not curb code growth. Thirdly, additional strategies are required to handle individuals that do not respect the limits: they could be discarded and the genetic operation repeated or correction stategies could be applied to reduce their size/depth.

Maximal size/depth restriction is anyway really simple and computationally inexpensive and in some cases has proved successful. Luke and Panait (2002a) strongly support depth limiting ("when it comes to fitness, plain depth limiting is hard to beat" Luke and Panait (2002a, p. 420)). Particularly good performance of this approach in symbolic regression for tree-based GP are reported in Luke and Panait (2002b). In many cases depth/size limiting is successfully used in combination with other parsimony pressure techniques (Blickle 1996, Luke and Panait 2002b). Often it is used to impose a limit to the computational cost of the evolution (Smits and Kotanchek 2004).

4.5.4.2 Parametric approach

Parsimony pressure is an alternative expression to refer to the use of size penalisation. Parsimony pressure is said to be *parametric* when size (or depth) is included explicitly in the fitness function as an objective (Nordin et al. 1996, Bleuler et al. 2001, Luke and Panait 2002b). In symbolic regression tasks, a general parametric approach consists in adding to the main objective (error metric) a term depending on the individual size, depth or any correlated parameters[5]:

$$F_i = E_i + p(s_i) \tag{4.4}$$

where E_i is the error metric of individual i on the building data set and p a function of the size s_i of the individual or correlated parameter. A linear relationship for p is generally used (Nordin et al. 1996, Blickle 1996, Bleuler et al. 2001, Alvarez 2000, Luke and Panait 2002a):

$$F_i = E_i + \alpha s_i \tag{4.5}$$

α being a weight that can be used to tune the relative importance of the size (or depth) with respect to error E_i. Eq. (4.5) follows the general formulation of the weighted-sum method in multiobjective optimisation (Marler and Arora 2004). Soule et al. (1996) used GP with linear parsimony pressure to evolve robot guidance laws: he activated the size penalisation defined by Eq. (4.5) only if the size of the program goes beyond a predefined threshold.

[5]see for example the *expressional complexity* defined in Section 4.3.1, which is used to both penalise size and nested operations.

Linear parsimony pressure is one of the most effective techniques for size control in GP due to the generality of the approach (Soule and Foster 1998b). However, the "benefits and costs of parametric parsimony pressure" are strictly related to the amount of pressure used (Soule and Foster 1998a, p. 299). Excessive parsimony pressure reduces individual size degrading however average individual fitness, and viceversa (Blickle 1996, Soule and Foster 1998a). Guessing a priori the appropriate value of the weight α that introduces a balanced amount of parsimony pressure is extremely difficult as its optimal value depends on the problem and on the definition of the error metric E_i (Blickle and Thiele 1994, Zhang and Mühlenbein 1995, De Jong and Pollack 2003, Vladislavleva 2008). If the fitness formulation features more than two objectives, it is also not trivial to guess the correct relative importance of size (or depth) with respect to the other objectives in generating a high-quality individual (Smits and Kotanchek 2004). Performing a few preliminary runs is therefore usually suggested to optimise as much as possible the benefits of linear parsimony pressure (Blickle 1996, Soule et al. 1996, Nordin et al. 1996, Bleuler et al. 2001). Alternatively, cross-validation strategies (see Section 1.2.2, Chapter 1) can be used to tune the objective weights, which following the terminology used in the machine learning community can be called *hyperparameters*: a useful example is provided by Lew et al. (2006).

Luke and Panait (2002a) and Luke and Panait (2002b) observed that the importance of the different objectives may change during evolution, so assigning them a constant value may negatively affect GP search, favouring small size over error reduction. The experiments performed by Soule and Foster (1998a) show that this is indeed the case.

Adaptive parsimony pressure The analysis performed by Soule and Foster (1998a) proves that a variable parsimony pressure that can adapt to the different stages of evolution increases the efficiency of GP. Adaptive parsimony pressure has been implemented in different ways.

In Kalganova and Miller (1999) a "two-stage" approach is followed to optimise fitness and minimise size. Parsimony pressure is activated through a second objective linked to the size of the individual only when a given level of the main objective (performance, raw fitness) is reached. The approach can be generalised by the fitness function definition proposed in Bleuler et al. (2001, p. 537) (minimisation of fitness assumed):

$$F_i = E_i + 1 \quad \text{if } E_i > \epsilon \tag{4.6}$$

$$F_i = 1 - \frac{1}{N_i} \quad \text{if } E_i \leq \epsilon \tag{4.7}$$

where F_i is the fitness of individual i, ϵ is the error threshold, E_i is the error of individual i and N_i is the corresponding size. A two-stage fitness function lets the population evolve an individual of acceptable quality first and then tries to reduce the size of such individual. The successive application of the two penalisations prevents parsimony pressure from conflicting with the main objective (error) when the best individuals are still being looked for, main drawback of linear parametric pressure. The idea was already put forward in Zhang and Mühlenbein (1995), although not specifically applied to GP. A main drawback of the two-stage fitness formulation is that the error threshold ϵ has to be defined by the user, who may not have enough knowledge to guess its optimal value. Parsimony pressure may then never be activated and bloat occur undisturbed (Bleuler et al. 2001).

A two-stage but also smoothly adaptive parsimony pressure approach is used in Zhang and Mühlenbein (1995). The main idea is to exploit the two-stage approach and for each stage introducing an adaptive parsimony weight $\alpha(g)$ which depends on the history of the correlation between the error and the size of the optimal individual in the population. In mathematical terms:

$$F_i(g) = E_i(g) + \alpha(g)C_i(g) \tag{4.8}$$

where $F_i(g)$ is the fitness value of individual i at generation g, $E_i(g)$ is the error and $C_i(g)$ is a complexity metric (size of the individual for example (Bleuler et al. 2001)). Further details on the definition of $\alpha(g)$ can be found in Zhang and Mühlenbein (1995).

The two-stage fitness functions presented require the definition of an error threshold ϵ, so a few preliminary tests are required to optimise this parameter.

4.5.4.3 Lexicographic (rank-based) approaches

Two strategies for the optimisation of accuracy and size of GP individuals that do not require tuning parameters have been proposed by S. Luke and L. Panait. The common feature of these approaches is that they compare objective of the same nature through a non-parametric approach.

Lexicographic parsimony pressure (Luke and Panait 2002b) is introduced during the selection stage through two comparison stages. Individuals are first compared with respect to their raw fitness/error values. If the values are different, the individual with best quality is selected. Otherwise, a second comparison is done with respect to size: the smallest individuals is then chosen. In the rare case that individuals have same fitness and same size, the selection is random. The approach is based only on rank, so no tuning parameters are needed to articulate a preference between two objectives of different nature. The strategy was developed with the aim of reducing introns in populations with a lot of individuals with identical raw fitness/error, encouraging the selection of the smallest individuals among the fittest. However, Luke and Panait (2002b) acknowledge that the strategy is not effective for symbolic regression, where the slight fitness improvements due to overfitting drastically reduce the probability of execution of the second comparison, so impairing bloat and overfitting reduction. The idea underlying lexicographic parsimony pressure was already proposed by Zhang and Mühlenbein (1995), who also acknowledged its similarity with the two-stage strategy used in Kalganova and Miller (1999).

In Luke and Panait (2002a) a parsimony pressure based on "double tournament" selection is described. The selected individual is the winner of a final tournament based on size performed among individuals that have already passed a first qualifying tournament based on raw fitness/error (or viceversa). Compared to maximal depth restriction (Section 4.5.4.1), the strategy appears to better curb bloat, but not in a statistically significant manner.

Although not more effective in reducing code bloat than maximal depth/size restriction if used by themselves, lexicographic parsimony pressure and double tournament performances drastically improve when applied together with maximal depth/size restriction. The main effect of a combined used of the previous techniques is a dramatic reduction in GP individual average size (Luke and Panait 2002b), in some cases halved (Luke and Panait 2002a), with respect to the average size of the solutions produced using maximal depth/size restriction. No beneficial effect on best raw fitness/error is however recorded by Luke and Panait (2002ab).

4.5.4.4 Pareto approaches

Pareto approaches, introduced in Section 4.3.2.1, have been used to introduce parsimony pressure in GP. Similarly to rank-based approaches, Pareto approaches do not compare objectives of different nature and do not require user-defined parameters to articulate a preference among the objectives.

In its simplest form, Pareto parsimony pressure is introduced considering raw fitness/error as first objective and size, depth, or any other related parameter as second. Pareto parsimony pressure biases the evolution towards compact and accurate individuals introducing a ranking among individuals that is used in selection strategies as an alternative to more common fitness-based rankings, for example in tournament selection (Fonseca and Fleming 1995, Vladislavleva 2008). Different dominance-based criteria have been used to rank individuals in a Pareto sense to promote the convergence towards the actual Pareto front and to encourage its uniform exploration. These criteria are mainly based on three metrics defined in the Pareto objective space, which result in different exploration behaviours of such space (Fonseca and Fleming 1995, Smits and Kotanchek 2004):

- *non domination level*: ranking built recursively eliminating from the objective space the individuals belonging to the Pareto front (for example *fast non-dominated sorting algorithm* in Deb et al. (2002), used by Vladislavleva (2008, p. 95)).
- *domination*: number of individuals that dominate the given individual. It encourages exploration at the edges of the known Pareto front or in new regions.
- *dominance*: number of individuals the given individual dominates (Zitzler and Thiele 1999). It encourages the exploitation in the middle of the known Pareto front (used for example in Kroo (2004)).

The above described criteria have been used to implement many Pareto strategies for evolutionary algorithms: PAES (Knowles and Corne 1999), SPEA (Zitzler and Thiele 1999), SPEA2 (Zitzler et al. 2001) and NSGA-II (Deb et al. 2002) are among the most known. As Pareto selection strategies require only the definition of a metric on the objective space, regardless the individual representation or the task of the optimisation, they are easily adaptable to GP. Some useful recommendations about their optimal use in GP can then be obtained from the analysis of Pareto EAs and GAs. Elitism is reported by Zitzler and Thiele (1999), Deb et al. (2002) to improve accuracy and evolution speed in GAs, so its use is usually suggested in Pareto GP implementations (Teller and Veloso 1996, Bleuler et al. 2001, Sætrom and Hetland 2003, Smits and Kotanchek 2004, Vladislavleva 2008). In this case special attention has to be paid to ensure that the archive is updated correctly with the non-dominated individuals, as in general the number of individuals composing the Pareto front is different from the size of the archive. Different clustering techniques can be used to select a given number of uniformly distributed individuals on the Pareto front (for example *crowding* (Deb et al. 2002), *average linkage method* (Zitzler and Thiele 1999)). In the opposite scenario, different Pareto layers can be involved in the selection process if the individuals on the Pareto front are fewer than the archive size (Vladislavleva 2008).

4.5.5 Avoiding destructive crossover

Forcing crossover (and subtree mutation) to be constructive is an indirect way to fight disproportionate code growth and fitness stagnation, as it dissolves the evolutionary advantage of individuals containing a large amount of neutral code (see Section 4.4.4). This approach not necessarily reduces all neutral code, limits instead introns to the extent that they do not hinder fitness improvement. It can also be used in combination with other anti-bloat strategies, to boost fitness once the protection against variation ("population inertia") has been weakened by the partial elimination of introns. Indeed, the simple reduction of neutral code percentage in the individuals does not necessarily ensure that the likelihood of constructive crossover increases (Ryan et al. 1998, p. 91).

The following strategies are common ways used in GP to ensure that crossover is constructive or at least not able to destroy the best solutions found:

Pseudo-hillclimbing. An offspring is accepted only if its fitness is equal (mild version) or better (rigorous version) than the fitness of the parent which supplied the offspring's root node. Otherwise, a copy of the parent is taken (Soule and Foster 1998b, Langdon et al. 1999). The tests performed by Soule and Foster (1998b) on a maze navigation problem and on the even-7-parity problem show that when mild pseudo-hillclimbing is used, code growth is reduced but still present. The rigorous pseudo-hillclimbing instead affects so much bloat dynamics that the dependency between tree size and generations is reduced to a sub-linear trend. Soule and Foster (1998b) explained this behaviour observing that the strong requirement posed by the rigorous pseudo-hillclimbing strategy eliminates the individuals containing introns whose existence is predicted by both the removal bias and "protective role of introns" theories. Only code growth correlated to fitness improvement occurs, as predicted by the "fitness causes bloat" theory, and the rate of growth is far slower than the quadratic or sub-quadratic one observed when introns are allowed (see Section 4.4.1).

The two versions of pseudo-hill climbing increase convergence speed and reduce the use of computational resources due to the reduced amount of introns in the individuals. However, they do not significatively improve average individual fitness (Soule and Foster 1998b). This may imply that “a certain amount of neutral and/or destructive crossovers are necessary for an effective search” (Soule and Foster 1998b, p. 785), indirectly confirming the important role of introns in ensuring a global exploration of the design space in the early stages of a GP run (introns as “incubators” of promising subtrees - see Section 4.4.3). A drawback of the pseudo-hilclimbing method is its computational cost: a lot of individuals have to be generated and evaluated to select just a few.

Self-adaptive selection pressure steering. Self-adaptive selection pressure steering was originally developed for GAs (Affenzeller and Wagner 2004). It is basically a variation of the pseudo-hillclimbing strategy: during the evolution, an increasing percentage of the population has to be made of individuals that have better fitness value than their worse parent (or than an average of the parents’ fitness values). The remaining part of the population is filled with randomly selected individuals from the pool of offspring that did not satisfy the success criterion.

The application of this approach to GP, as done in Winkler et al. (2007), is heavily penalised by the computational costs of generating and evaluating a number of individuals much larger than the population, given the fact that crossover is mostly destructive (see Section 4.4.4.3). If in GAs this is possible due to the fast genotype evaluation (in Affenzeller and Wagner (2004) the size of the evaluated individual pool reaches up to 11 times the population size), in GP such process is computationally unfeasible. As the possibility to find individuals that are better than the parent is related to evolution convergence, the self-adaptive selection pressure steering method has also inspired a termination criterion (see Section 3.1.6, Chapter 3).

Plagiarism penalty Plagiarism penalty (Langdon and Poli 1998b) is a parametric approach that penalises the fitness of an individual if it is equal to its best parent’s. This strategy reduces the evolutionary advantage of offspring containing introns, which by definition have the same fitness of their parents. Penalising them breaks the protection mechanism provided by introns and reduces “population inertia”, so plagiarism penalty is expected to extend the training or youth stage of a GP run and to retard premature convergence (see Section 4.4.3). The experiments performed by Langdon and Poli (1998b) on the artificial ant on Santa Fe trail problem show that plagiarism penalty is able to slow code growth without penalising the performance of the best individuals. Similarly to what occurs with pseudo-hillclimbing, bloat is not stopped because the bloat mechanisms due to the removal bias theory and the "fitness causes bloat" theory are not targeted. A second important effect observed by Langdon and Poli (1998b) is that plagiarism penalty can increase genotype diversity varying the magnitude of the penalisation, which in turn may reduce the risk of premature convergence. As a result of the reduced introns protection and the enhanced variability, GP runs benefit from an extended training or youth stage. Unfortunately in Langdon and Poli (1998b) the plagiarism penalty was not tested on symbolic regression problems, but it is reasonable to assume that similar effects can be expected due to the general validity of the bloat theories.

4.6 Boosting variability: distributed genetic programming

Introns and lack of variability are the main causes of premature convergence in GP and GA (Affenzeller and Wagner 2004). Mutation has been traditionally trusted with the duty of boosting variability and converting neutrons into effective code to reduce the "population inertia" to retard premature convergence (see Section 4.4.3). Common strategies to boost variability are the *epoch replacement* operator (Whigham 1995), which consists in generating from scratch a certain number of individuals each generation, and the *cascade* operator (Section 3.1.5.6, Chapter 3), through which the population is periodically destroyed and regenerated.

A radically new approach to increase variability is exploited in *distributed genetic programming* (Koza 1992, Luke and Spector 1996, Smits and Kotanchek 2004, Affenzeller and Wagner 2004, Poli et al. 2008): several subpopulations, or *demes*, are evolved simultaneously and from time to time a few individuals migrate from a deme to another to exchange fresh genetic material. The stochastic nature of genetic drift increases the likelihood that different demes converge to different optimal solutions (Affenzeller and Wagner 2004, p. 250), so the inclusion of the migrating individuals is likely to reduce the risk of premature convergence.

Many different strategies have been adopted to manage the flow of migrating individuals from deme to deme. A distributed approach is used in the GA-GP algorithm developed by Affenzeller and Wagner (2004), called SASEGASA. In SASEGASA the initial population is split into many demes, which are let to evolve independently. Self-adaptive selection pressure steering is used to detect premature convergence of the demes: when it is not possible to generate a certain number of offspring outperforming the parents, a deme is is merged with other converged demes. Progressively the number of demes reduces, while their population size increases. At the end of a SASEGASA experiment a all demes are absorbed into a single population having the size set at the beginning. The distributed approach featured by SASEGASA allows to enhance variability in the subpopulation and so reduce the risk of premature convergence. Although the computational cost of SASEGASA is penalised by the expensive self-adaptive selection pressure steering method, it can be said that the use of computational resources in SASEGASA is more efficient than in a standard GA. The evolutions in the single demes are indeed stopped soon after the training stage has terminated, and converged demes are merged. In this way the computational power that would not have produced significative improvements during the stagnation stage of the demes' evolution is used to train a new, bigger and heterogeneous population with a higher potential to find the global optimum/a. SASEGASA was originally developed for GA but it has also been applied to GP for metamodelling purposes (Winkler et al. 2007). In GP the self-adaptive selection pressure steering method used in SASEGASA acquires futher importance, as it reduces introns. A distributed GP was also used by Iba and Terao (2000) to evolve robot navigation strategies.

It is worth to remind that if distributed GP implies the parallel execution of many GP runs, a parallel GP implementation does not necessarily adopt a distributed approach. More details on parallel GP are given in Appendix A.

4.7 Reducing fitness evaluation cost

Genetic programming is a computationally expensive technique. Much research effort has been dedicated to the reduction of fitness evaluation, which is the most time consuming operation in genetic programming (Sims 1993, Schoenauer et al. 1996, Giacobini et al. 2002, Xie et al. 2006, Vladislavleva 2008).

If elitism is used and if the building data set does not change throughout the evolution, the simplest strategy to reduce the evaluation cost is to evaluate elite individuals only once, when they are inserted into the archive (Koza 1992). The savings in term of reduction of the number of evaluated individuals are proportional to the size of the elite.

More effective strategies to reduce fitness evaluation costs have been developed. They can be classified according to the part of the evaluation process they attempt to simplify: some reduce the number of fitness cases, others the number of individuals evaluated, a third group simplifies the individual to be evaluated and/or the fitness function to make fitness evaluation cheaper.

4.7.1 Reducing the number of fitness cases

Fitness evaluation cost can be reduced limiting the number of fitness cases considered in the evaluation process, although modifying the building data set in size or distribution may potentially affect the quality of the evolved individuals (Section 3.2.1, Chapter 3). Giacobini et al. (2002) proved that it exists a minimum number of fitness cases from which a boolean function can be reconstructed by GP. It appears therefore that reducing the number of fitness cases may reduce computational cost without affecting the quality of the best individuals returned by GP, at least for the symbolic regression of functions defined on a discrete domain.

The research done by Vladislavleva (2008) and Vladislavleva et al. (2010) on balancing techniques (Section 3.2.1, Chapter 3), confirm that also in GP used for the symbolic regression of continuous functions the computational cost associated to fitness evaluation can, under certain conditions, be cut reducing the number of fitness cases without degrading the quality of the regression.

Goal softening and the ESSENCE algorithm (Vladislavleva 2008) are techniques that allow for reduction in evaluation costs or, equivalently, allow to improve the quality of the evolved metamodels for the same computational budget They consist of increasing progressively (linearly) the size of the subset used to evaluate the individuals throughout the evolution, starting from a subset considerably smaller than the original building data set (Vladislavleva (2008) used 10% of the original building data set size). GP individuals generalisation ability is increased if the increasing subsets are selected randomly (Vladislavleva 2008). Similar approaches were used also by Angeline and Pollack (1993) (GAs) and by Schoenauer et al. (1996) (GP).

4.7.2 Reducing the number of individuals evaluated

Evaluating only a percentage of the individuals in a GP population is a way to reduce the total computational cost of evaluation. The *population clustering* strategy (Xie et al. 2006) for GP for symbolic regression tasks aims at reducing the number of individuals actually evaluated by clustering *fitness-case-equivalent* individuals. Fitness-case-equivalent individuals are defined as metamodels that produce the same output in two successive training samples, randomly selected. Once all the individuals in the population are assigned to a cluster, only the fitness value of the smallest individual in each cluster is evaluated and is then assigned to all the other individuals in the cluster. Such fitness value is then used to establish a ranking among clusters, and then in tournament selection.

Population clustering considerably increases success rate and convergence speed in symbolic regression. However, the computational cost of the additional operations that clustering requires is not negligible and dissolves much of the savings obtained (Xie et al. 2006).

4.7.3 Simplifying the evaluation procedure

The main advantage offered by metamodels is that they are inexpensive to evaluate, so considerable reduction in the fitness evaluation cost may be expected from the use of a fitness function metamodel. This principle was followed by Ziegler and Banzhaf (2003), who used machine code GP (Nordin et al. 1999) to evolve a classifier able to determine the winner of a tournament selection from the genotypical features of the individuals taking part to it, this way eliminating the need to perform fitness evaluation of each individual. Despite using a metamodel of the fitness function is a powerful way to reduce fitness evaluation cost in GP, the assessment of the actual savings has to take into account that the evolution through GP of such metamodel may also be expensive (Ziegler and Banzhaf 2003).

4.8 Multiple genotype-phenotype mappings

The representation used by genetic programming, be it linear, tree or graph, is intrinsically redundant. *Redundancy* or *neutrality* (Banzhaf 1994) is defined in GP as the possibility to generate different genotypes, even radically different, having the same phenotype (measured by the fitness value). The variable-length nature of GP individuals further increases the number of ways individuals with the same observable properties can be built.

As introduced in Section 4.4.3, redundancy is fundamental to increasing the probability of evolving high-quality solutions for two main reasons. The first is that it multiplies the number of evolutionary paths that GP can follow to reach the globally optimal phenotype (Miller and Thomson 2000, Ferreira 2001). Secondly, it makes neutral variations of the genotypes possible, enhancing variability in the population and favouring the exploration of the design space (Banzhaf 1994).

A few mechanisms that indirectly enhance GP redundancy have been analysed in this chapter. A first one is the use of many functional primitives (see sufficiency property in Section 4.2). Introns can also be used to effectively increase the level of redundancy in GP (see Section 4.4.1), although their excessive and uncontrolled growth may produce collateral effects (see Section 4.4.3).

A third way to increase redundancy in GP is to use a *double genotype-phenotype mapping*. The standard GP paradigm, as the one described by Koza (1992) and analysed in Chapter 3, adopts a single genotype-phenotype mapping: tree-based genotypes are modified by genetic operators and the corresponding phenotypes, for example mathematical expressions, can be extracted from them using a mapping algorithm. A double genotype-phenotype GP implementation instead use two orders of genotypes: genotypes of the first order (inner) undergo genetic operations but they cannot be directly mapped to the final program (for example a mathematical expression). They have to be converted in genotypes of the second order, for example to syntax trees, and then transformed to their final shape (mathematical expression)[6]. This way the redundancy intrinsic in the representations used in each genotype order can be exploited, increasing the overall redundancy of the GP implementation.

Binary Genetic Programming (BGP) (Banzhaf 1994), Grammatical Evolution (GE) (Ryan et al. 1998), Cartesian Genetic Programming (CGP) (Miller and Thomson 2000) and Gene Expression Programming (GEP) (Ferreira 2001) are different examples of double genotype-phenotype GP. They all feature fixed-length GA chromosomes as first order genotypes, mapped to a syntax tree or a directed graph for evaluation purposes.

[6]Some attention has to be paid to the different and sometimes ambiguous usage of terms in standard GP and double-mapping GP. For example in Vladislavleva (2008, p. 81) the syntax tree is called “genotype” and the corresponding response surface is the “phenotype”. For double-mapping GP, for example in Miller and Thomson (2000), the syntax tree is the “phenotype” and the linear chromosome which is mapped into is referred to as the “genotype”. In this case it does not appear clear how the final program, for example the response surface in a symbolic regression problem, has to be called. As a result, the naming conventions introduced by Vladislavleva (2008) are here followed.

Chapter 5

Hybrid genetic programming

Previous chapters have provided an extensive background on genetic programming theoretical assumptions and described many of the different implementations that have spawned from Cramer's and Koza's work. This chapter is dedicated to the illustration of the GP implementation, which will be named HyGP, developed as part of the research activity described in this thesis.

After a brief description of hybrid approaches in genetic programming (Section 5.1), the improved implementation is described in detail in Section 5.2, while in Sections 5.3 and 5.4 a few strategies to boost its performances are presented.

The last part of the chapter is dedicated to the comparison of the enhanced HyGP with other metamodelling techniques. In Section 5.5 tests carried out with a parametric metamodelling technique, polynomial chaos expansion (PCE), are described. The optimisation of a 10-bar truss performed using metamodels generated by HyGP and moving least squares method (MLSM) is described in Section 5.6.

5.1 Hybrid (or memetic) techniques

In Chapter 1 it has been acknowledged that there are substantially two kinds of exploratory algorithms, gradient-based and stochastic ones. Their features are contrasting, as the exploitation of information provided by derivatives allows for an efficient but local exploration, whereas stochastic searches are generally characterised by a slower but global exploration of the design space, be it a vectorial space (as in GA) or a function space (as in GP for symbolic regression tasks).

The integration of traditional, deterministic search techniques in population-based algorithms seem therefore a natural way to achieve speed and to avoid local optima. This basic idea has lead to the birth of the so-called *hybrid* or *memetic* approaches. One of the first indications on the potential benefits of hybrid approaches in GA, EP and ES can be found in Fogel (1994).

The idea of merging the two search strategies spread also to the genetic programming community. Deterministic algorithms have been used differently according to the specific nature of the program to be evolved. In particular, many researchers developing GP for symbolic regression tasks (Collet et al. 2000, Topchy and Punch 2001, Keijzer 2003) have argued that the search for the right value of the parameters can actually spoil the search, as a potentially good or optimal mathematical structure can be penalised excessively by a wrong value of just one constant, leading to the disappearance of the precious individual from the population. As a result, hybrid GP implementations have focused in particular on the use of deterministic algorithms to optimise the allocation and the tuning of the numerical constants that may be generated during individual initialisation and evolution. In order to appreciate the new perspective offered by hybrid GP approaches is necessary to plunge into the way numerical constants have traditionally been handled in genetic programming, with particular attention to symbolic regression tasks.

In the original Koza's GP formulation (Koza 1992) numerical constants, called *random ephemeral constants*, are randomly generated and inserted in the individuals at the beginning of the evolution, so that the tuning process relies uniquely on the ability of crossover and mutation to cluster or group the numerical coefficients to generate the final numerical values (see also Lew et al. (2006)). In this sense, parameters tuning is performed by evolutionary mechanisms. Not all researchers have however agreed on the advantages of randomly generating the numerical constants only at the beginning of the evolution. In Binary Genetic Programming (see Section 4.8, Chapter 4) Koza's assumption is dropped and the numerical parameters are randomly generated at each generation as a result of a modification of the portion of genotype coding the parameters. On the other hand, Miller and Thomson (2000) report that for CGP (see Section 4.8) the search improves using a fixed value (1.0) instead of the ephemeral random constant introduced by Koza (1992).

Adventuring a bit further, some researchers have criticised the very use of explicit numerical constants. Ferreira (2002) shows how not using numerical terminals for the intialisation of the individuals can drastically improve GEP success rate in symbolic regression and other tasks with respect to standard GEP implementation where constants are part of the individuals. Experiments show that GEP is able to generate the numerical value exploiting properties as, for example, the identity element of multiplication and addition (if $x * 1 = x$, then x/x generates 1)[1].

Hybrid approaches feature strategies to deal with numerical constants rather different from all the ones described above. A deterministic optimiser is generally used to tune the GP individuals' numerical values, often fed with initial guesses provided by a stochastic optimiser, able to perform a global search for roughly optimal parameters values. This mechanism allows to refine locally inaccurate models, leaving to the evolutionary mechanisms the global exploration for an acceptable mathematical model. Hybrid approach may be considered as performing deterministically a local refinement (exploitation) that is otherwise performed by crossover and mutation operators in conventional GP (Poli and Langdon (1998), see also Section 3.1.5.5).

The main advantages of hybrid approaches can then be syntetically formulated as:

- higher confidence in individuals' fitness value. Optimising the values of the parameters allows to reduce fitness variance due to bad values of them and so it provides a more reliable assessment of the mathematical structure of the individual. The more parameters' values optimisations are performed starting from different initial guesses, the more reliable is the assessment. As a result, the risk of losing individuals with a good mathematical structure but "bad" parameters' values is minimised;

- reduction of the size of the trees during evolution, as clusters of nodes are not needed to produce constants (Schoenauer et al. 1996). As a result, increased search efficiency, reduced RAM usage and more efficient use of computational power are to be expected from the adoption of hybrid approaches in GP;

- faster tuning of the numerical parameters;

- increase of the evolution speed, as the search for a good mathematical structure is not hindered by the process of aggregating blocks of code to tune numerical parameters by evolutionary mechanisms (Zhang and Mühlenbein 1995);

- reduction of the search space (Ferreira 2002);

- reduction of the number of generations required to reach the same level of accuracy.

[1]Ferreira (2002) however does not mention any problems related to the non definition of the operations generating the numerical constant: $x/x = 1$ but if $x = 0$ a measure to make the operation legal must be implemented to ensure semantical closure (see Section 4.1.2, Chapter 4).

Aware of the benefits, researchers have come up with many different hybrid GP implementations, featuring a similar core GP algorithm but different optimisers for numerical coefficients tuning. Probably the first attempt to use an additional tuning algorithm can be ascribed to Zhang and Mühlenbein (1995), who perturbed GP numerical parameters by a series of random variations, eventually selecting the set of numerical coefficients resulting in the best fitness value according to a hill-climbing strategy. Blickle (1996) used a similar approach, changing parameters sequentially by a fixed amount for a specified maximum number of optimisation steps. Inspired by evolution strategies and evolutionary programming (see Section 2.3.1, Chapter 2), Schoenauer et al. (1996) and Chellapilla (1997) performed tuning adding Gaussian noise to numerical parameters (in Chellapilla (1997) this procedure is referred to as Gaussian mutation). Gray et al. (1996 1997) used for tuning purposes a Nelder-Simplex algorithm and simulated annealing (simulated annealing was also suggested by Lew et al. (2006)).

Strategies as the ones just outlined represent initial attempts to make parameter tuning independent from evolution. Nontheless, they do not benefit from the quick and fine tuning achievable by deterministic optimisers, although stochastic optimisers are effective in providing a rough guess of the optimum set of numerical coefficients.

An example of the integration of a deterministic optimiser in tree-based GP was presented in Topchy and Punch (2001), where local tuning is performed by a gradient-based optimiser. Linear scaling was instead used by Keijzer (2003). The same approach was used in the Pareto-based GP implementation described in Smits and Kotanchek (2004), Smits et al. (2005), Vladislavleva (2008), Vladislavleva et al. (2010) (see Section 4.5.4.4, Chapter 4).

Still, all the described implementations have missed a fundamental issue for the success of hybrid approaches. In fact, the computational cost of tuning the numerical parameters has not been taken into account: if the number of coefficients in the GP individuals are not kept to the bare minimum, so that expressivity and then fitness value are not compromised, the advantages provided by deterministic tuning quickly disappear. Experiments carried out by the author (not shown here) have proved that the natural proliferation of numerical constants in Koza's style GP makes the deterministic tuning of an entire population of individuals computationally unfeasible after the first generations, due principally to bloat (see Section 4.4.1, Chapter 4). A clever strategy to optimise both the number and the location of the coefficients in GP individuals is then key to the success of hybrid approaches.

In this regard, the hybrid tree-based GP implementation developed by Alvarez (2000) (used also in Ashour et al. (2003)) is particularly interesting as numerical nodes are inserted into parameterless individuals in optimal location so that the number of coefficients is minimised, reducing as a result the tuning cost. The parameters insertion algorithm is based on the application of simple algebraic properties, like distributive property for multiplication (more details can be found in Alvarez (2000)). The approach followed in Alvarez (2000) makes GP individuals' coefficients handling completely independent from evolution, as numerical nodes do not take part to genetic operations (although the spontaneous generation of constants can still be expected as a result of the existence of neutral elements for the functional primitives used - ex. $x/x = 1$). The approach also reduces the appearance of semantical introns, reducing the use of computational resources and delaying premature convergence (see Section 4.4.3, Chapter 4).

5.2 HyGP, a hybrid GP implementation

This section is dedicated to the description of a hybrid tree-based GP implementation aimed at symbolic regression tasks, which has been called "HyGP". The software was used to produce the metamodels shown in Chapter 6 and is freely downloadable from Armani (2011).

HyGP evolutionary engine has been developed according to the guidelines provided in Koza (1992) and summarised in Chapters 2 and 3. However, the evolutionary search has been used only to find a set of good mathematical structures, while deterministic algorithms have been opted for for the insertion and optimisation of the numerical values in the evolved parameterless individuals, in order to benefit from the advantages of the hybrid GP approaches described in the previous section. The work done by Alvarez (2000) has been the main inspiration for the efficient use of computational resources (computing power and memory usage). Therefore, although the traditional genetic programming operators - reproduction, mutation, crossover - are still used, they are applied to GP individuals (metamodels) stripped of their numerical terminals. Parameters are inserted in the offspring and optimised only during fitness evaluation and at the end of the run. The whole process is shown schematically in Fig. 5.1.

Some changes have been introduced to the GP algorithm developed by Alvarez (2000) to improve the efficiency of the search and the quality of the evolved individuals. In the next sections HyGP's components are described in detail.

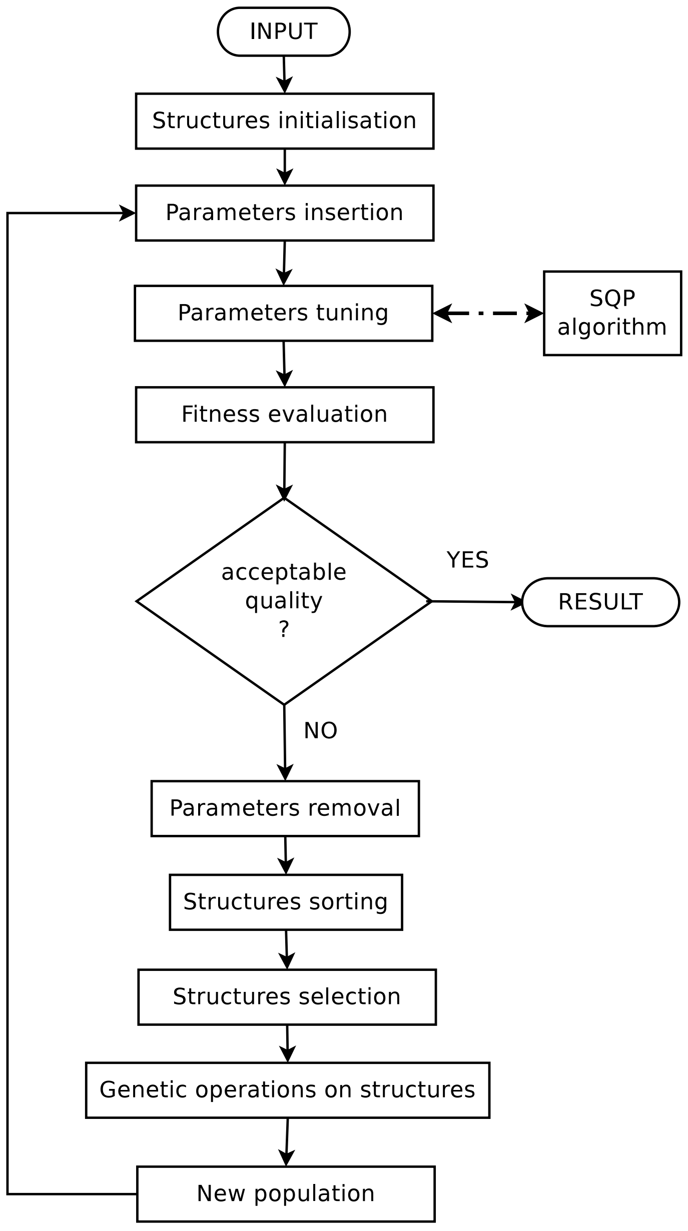


FIGURE 5.1: HyGP's hybrid GP algorithm

5.2.1 Initialisation

A modified version of the *ramped half and half* method is used to initialise the population (see Section 3.1.2, Chapter 3). The relative percentages of the population generated by the "full" and the "grow" method can be specified by the user[2]. As Alvarez's hybrid approach is adopted, nodes containing numerical values *are not* inserted in the trees at this stage. Individuals' structure is defined selecting functional nodes (FU, FB) and variable nodes (TV). At the end of the this step trees do not contain any numerical parameters (terminal constant nodes - TC).

[2]See parameter p_FULL in Section B.1.1, Appendix B.

5.2.2 Parameters insertion and tuning by an optimisation algorithm

The computational overhead required for parameter tuning is kept to the minimum through the optimisation of coefficients location and number. Following Alvarez' hybrid approach, at each new generation parameterless individuals, which will be termed *ancestors*, are first generated using traditional GP operations. Then, a *family* of mathematical expressions is spawned from each ancestor through the insertion of numerical terminals, which are finally optimised using a SQP algorithm (Madsen et al. 2002). The number of individual generated from each ancestor is equal to the number of sets of initial guesses used by the SQP optimiser: the individual having the lowest error after tuning is then chosen to represent the original ancestor[3]. Particular measures have been put in place in order to give the user the possibility to specify the data set used for parameter tuning and to choose the number of random initial guesses for the SQP optimiser (see Appendix B): the effect of the number of initial guesses, or the size of the "families", has been studied and results will be discussed in Section 5.3.0.3. In a standard HyGP experiment the number of initial guesses is 2.

Parameters are inserted recursively taking into account the possible simplifications granted by distributive property between sum/subtraction and multiplication/division, so that the number of parameters is reduced without compromising the expressivity of the individual. The rules applied by the insertion algorithm are described in Table 5.1. The reader will notice the similarity with the set of rules presented in Alvarez (2000, p. 59). Unary nodes containing $sine$ or $cosine$ are however here treated in a different way, in order to provide such primitives with more expressivity. In particular, as a result of rule 2 and 3, a sine or cosine term is provided with a parameter for amplitude (rule 2) and another one for frequency (rule 3).

The flowchart of the parameter insertion algorithm is shown in Fig. 5.2. An example of how the insertion algorithm works is given in Fig. 5.3, where is shown how the ancestor $z + sin(z)$ is turned into the expression $4.3 + 8.1z + 1.2sin(3.5z)$ (complete individual). It is worth reminding that inserting parameters necessarily alters the depth and the size of the tree.

[3]Encouraging the evolution of classes of individuals rather than of single individuals is a technique that has also been used by other researchers. Parallels can be found for example in the context-free grammar genetic programming implementation developed by Whigham (1995) (see Section 4.1.1, Chapter 4).

TABLE 5.1: Rules followed to insert constant nodes

1. Binary node (FB)

   - Multiplication and Division operations only require one tuning parameter:
     $F(\mathbf{x}) = x_1 * x_2 \longrightarrow \tilde{F}(\mathbf{x}, \mathbf{a}) = a_1 * x_1 * x_2$
   - All other operations require two tuning parameters:
     $F(\mathbf{x}) = x_1 + x_2 \longrightarrow \tilde{F}(\mathbf{x}, \mathbf{a}) = a_1 * x_1 + a_2 * x_2$
     $F(\mathbf{x}) = x_1^{x_2} \longrightarrow \tilde{F}(\mathbf{x}, \mathbf{a}) = (a_1 * x_1)^{a_2 * x_2}$
   - When F is a combination of the previous two approaches, tuning parameters are only applied to operations different from multiplication and division:
     $F(\mathbf{x}) = x_1 * (x_2/x_3 + x_4) \longrightarrow \tilde{F}(\mathbf{x}, \mathbf{a}) = x_1 * (a_1 * x_2/x_3 + a_2 * x_4)$
     $F(\mathbf{x}) = (x_1 + x_2)^{x_3 * x_4} \longrightarrow \tilde{F}(\mathbf{x}, \mathbf{a}) = (a_1 * x_1 + a_2 * x_2)^{a_3 * x_3 * x_4}$

2. Unary node (FU)

   - if $sin()$ or $cos()$, one tuning parameter is inserted; otherwise no parameters are added:
     $F(x) = sin(x) \longrightarrow \tilde{F}(x, a) = a * sin(x)$
     $F(x) = cos(x) \longrightarrow \tilde{F}(x, a) = a * cos(x)$

3. Terminal Var node (TV)

   - One tuning parameter is inserted:
     $F(\mathbf{x}) = (x_1) \longrightarrow \tilde{F}(\mathbf{x}, \mathbf{a}) = (a_1 * x_1)$

4. At the end of the process a free parameter is added to the whole expression:

   - $\tilde{F}(\mathbf{x}, \mathbf{a}) \longrightarrow F_{fin}(\mathbf{x}, \mathbf{a}) = a_0 + \tilde{F}(\mathbf{x}, \mathbf{a})$

5.2.3 Selection

The selection of the individuals for genetic operations (reproduction, crossover and mutation) exploits elitism. The main reasons for the adoption of an elitist approach are described in Section 3.1.5.1, Chapter 3.

In this regard, the individuals included in the population *elite* are granted a privileged status. The *elite* is defined as a subset made of a certain percentage of the best individuals in the population at each generation. So before selection ancestors have to be sorted according to the fitness value of the best complete individual they spawn.

In all the experiments that will be described in this chapter and in the next (Chapter 6) the *elite* size was set to 20% of the population, although there is complete freedom for the user to change the percentage defining the elite size (see Appendix B).

For the selection of candidates for crossover, the first parent is chosen as the individual having the lowest fitness value in a pool of three competitors randomly selected in the

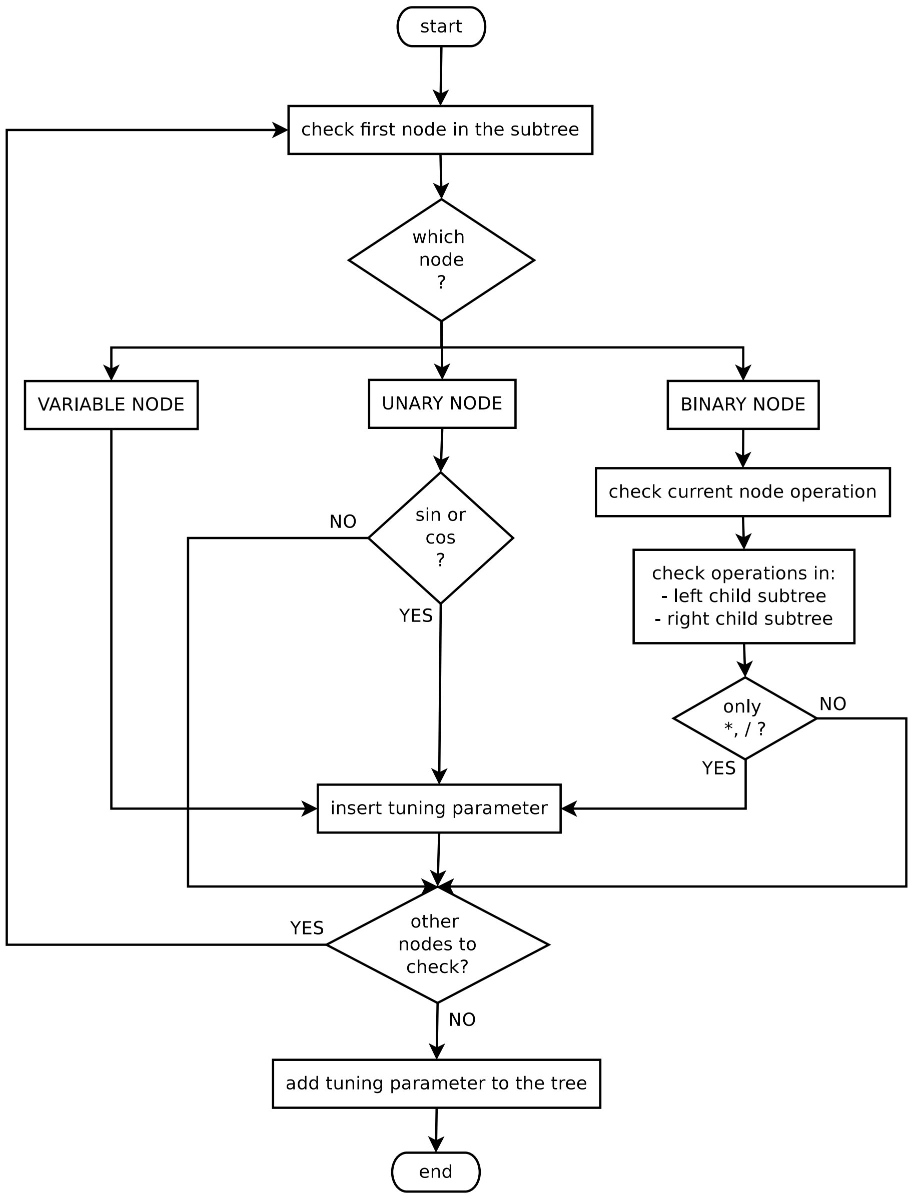


FIGURE 5.2: Algorithm to insert constant nodes in the tree structure

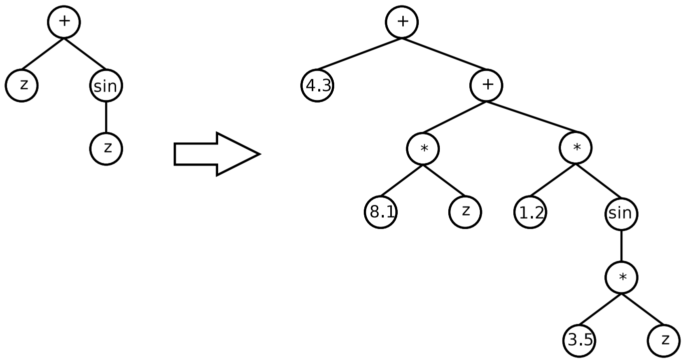


FIGURE 5.3: Tree before (*left*) and after (*right*) parameter insertion: $z + sin(z)$ turns into $4.3 + 8.1z + 1.2sin(3.5z)$

elite (tournament selection of size 3), whereas for the second parent the three competitors are randomly selected from the *whole population*. As for mutation, the individual undergoing mutation is the one having the lowest fitness value in a pool of three competitors randomly selected from the *elite*.

5.2.4 Reproduction, crossover and mutation

Traditional genetic operators are used to evolve ancestors, which are GP syntax trees without numerical parameters. Reproduction, crossover and mutation are applied independently from each other to generate a predefined percentage[4] of the population, according to description provided in Section 3.1.5.4, Chapter 3. In all the experiments shown in this chapter and the next (Chapter 6) the percentage of the new population generated by crossover and mutation has been set to 40%, leaving a percentage of 20% to reproduction, as shown in Fig. 5.4.

The independent application of the genetic operators allows to focus on the single effect of each particular genetic operator. This strategy has been therefore chosen to better understand the effect of crossover and mutation, as well as to make code validation simpler.

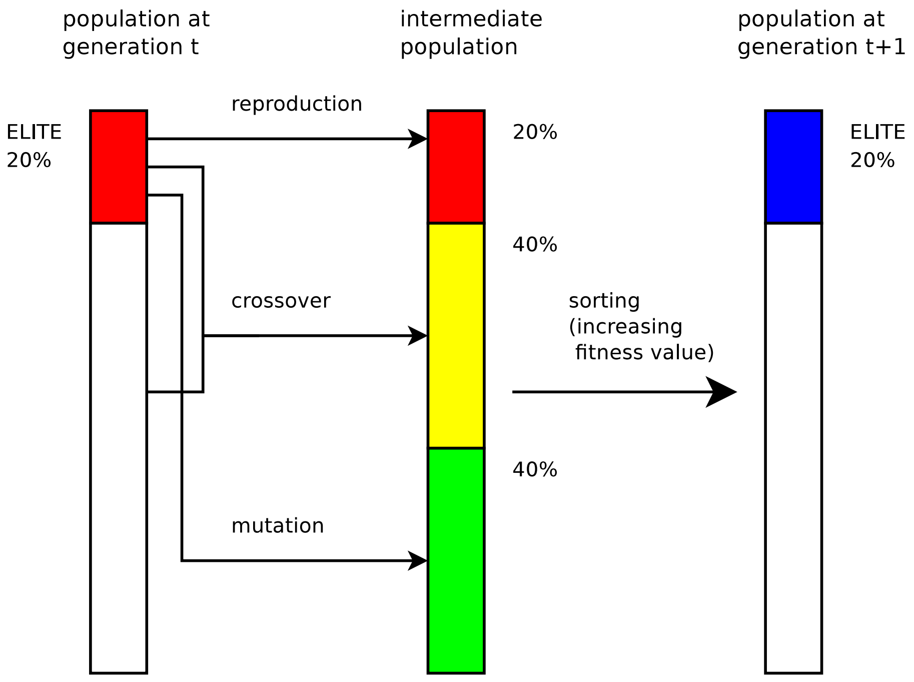

population at generation t
intermediate population
population at generation t+1
reproduction
ELITE 20%
20%
ELITE 20%
40%
crossover
sorting (increasing fitness value)
40%
mutation

FIGURE 5.4: HyGP replacement scheme

[4]These percentages are set through the parameters REPR_RATE, CROSS_RATE and MUT_RATE described in Section B.1.1, Appendix B.

Some modifications have been introduced to the standard versions of reproduction, subtree crossover and subtree mutation to improve the exploration of the search space and the robustness of the generated metamodels. Details are provided in the next sections.

5.2.4.1 Reproduction

All individuals belonging to the elite are transferred unchanged to the new population. If in the elite there are multiple copies of the same ancestor, they are replaced by new individuals, generated randomly using either a "full" or a "grow" algorithm[5] (the selection is done randomly) imposing a maximum depth of 2. This approach has been inspired by the *epoch replacement* operator used by Whigham (1995) (see Section 4.1.1, Chapter 4).

If the same ancestor survives in the elite for more than a generation, a reevaluation procedure is applied to verify its robustness: a new tuning of the coefficients is performed, using as initial guesses a set of values different from the ones obtained in the previous generation. The fitness value of the new individual is compared to the previous fitness value, and the best set of coefficients is chosen[6].

5.2.4.2 Crossover and mutation

Crossover and mutation are applied independently in HyGP. Subtree crossover is used, whereas mutation is alternatively point mutation (odd generations) and subtree mutation (even generations) (see Section 3.1.5, Chapter 3). Point mutation is used to introduce small changes in the individuals, which are likely to cause minimal disruption to the genotype (point mutation can be considered as a local or "delicate" operator). On the other hand, subtree mutation is also used for its potential to radically change the structure of the genotype. The resulting balanced set of mutation operators is then able to introduce slight (or local) as well as dramatic (or global) changes in genotypes, variations that are both needed in a GP search as described in Section 3.1.5.5, Chapter 3). In both cases, mutation is likely to introduce brand new pieces of genetic information, helping maintain genetic diversity.

Candidates nodes for crossover and mutation[7] are not selected with uniform probability, in order not to bias the selection towards terminal nodes, which results in local search

[5]for more details on these algorithms see Section 3.1.2, Chapter 3, .

[6]The number of reevaluations can be increased setting the number of initial guesses for the SQP optimiser to a value bigger than 2 - see parameter N_GUESSES in Section B.1.1, Appendix B, .

[7]see Section 3.1.5.2, Chapter 3

and bloat due to removal bias[8]. However, instead of imposing a separate hit rate for functional and terminal nodes[9], a two-step selection is used. The depth of the candidate node is chosen first, giving all depths in the individual the same probability to be selected. The node is then selected among all the nodes at the same depth, still all having the same probability to be selected. Such strategy appears to be a really simple solution to reduce the effect of removal bias.

To further reduce bloat, an upper bound on program depth is introduced, following the approach called *maximal depth restriction* described in Section 4.5.4.1, Chapter 3. It has been already reported on the benefits of this approach for symbolic regression[10]. The depth of the (parameterless) offspring generated by crossover or subtree mutation is checked: if it is lower or equal to the imposed limit, the offspring is accepted, otherwise the genetic operator is repeated until a child with an acceptable depth is generated. This is a variation of the traditional maximal depth restriction used in Koza (1992), according to which one of the parents is chosen if the child exceeds the limit depth.

5.2.5 Fitness function

In order to encourage the evolution of accurate, smooth and compact mathematical expressions, as well as to avoid bloat, the fitness function is defined as a weighted sum of different terms or objectives, as outlined in Section 4.5.4, Chapter 4.

The fitness value $F\left(i,t\right)$ of individual i at generation t is computed as:

$$F\left(i,t\right) = a_1 F_1(i,t) + a_2 F_2(i,t) + a_3\, 10^6\, F_3(i,t) + a_4 F_4(i,t) \tag{5.1}$$

$$a_1 + a_2 + a_3 + a_4 = 1 \tag{5.2}$$

where:

- F_1 is the root mean square error (RMSE) of individual i at generation t evaluated on the building data set, divided by the average RMSE of the *elite* individuals at the previous generation:
$$F_1 = \frac{RMSE(i,t)}{\overline{RMSE}(t-1)} \tag{5.3}$$
- F_2 is the square of the number of numerical terminals (or coefficients) present in the individual, after the parameter insertion has taken place.

[8] see Section 4.4.4, Chapter 3
[9] See Section 4.5.2, Chapter 3
[10] See Section 4.5.4.1, Chapter 3

- F_3 is the number of not legal operations that are encountered during the individual RMSE evaluation, defined as the total number of fitness cases in which any operation in the syntax tree is not legal (for example, division by zero).
- F_4 is the number of nodes the individual is made of.

The weighted approach defined by (5.1) has been chosen as the simplest way to optimise concurrently different objectives. Accuracy, parameter tuning cost reduction, smoothness and generalisation ability, bloat repression have been considered as the main factors contributing to a successful evolution[11]. As noted in Section 4.5.4.2, Chapter 4, some preliminary GP trials are required to tune the weights in Eq. 5.2.

Although not original in its structure, the fitness function defined in Eq. (5.1) presents some innovative features regarding the way objectives are defined. In first place, the traditional raw fitness defined as sum of the absolute errors or root mean square error[12] is replaced by a normalised error value (F_1). Initial GP tests, not shown here, confirmed the validity of the approach, in particular for a multiobjective fitness function formulation.

Secondly, the integration of a "smoothing" term that penalises undefined operations (F_3) is a novelty with respect to the fitness function formulation described by Alvarez (2000), whose hybrid approach has been an important source of inspiration for HyGP. This extra term, which was already used by Blickle (1996), has proved effective in improving smoothness and generalisation ability.

Finally, generalisation ability and computational efficiency are boosted whereas premature convergence and overfitting are contrasted penalising the size of the individuals through the terms F_2 and F_4. Reducing parameter tuning cost is key to the success of hybrid GP approaches (see Section 5.1), so the term F_2 has been introduced to bias the evolution towards individuals with fewer numerical coefficients, as also done in Alvarez (2000). However, this penalisation is not able to prevent in any way the progressive accumulation of nested multiplications and divisions, which, applying the rules in Table 5.1, require just one numerical coefficient. Therefore, a size penalisation has been added through the last term F_4 to prevent such eventuality, which can quickly disrupt the evolution process. The size penalisation expressed by term F_4 is widely recognised as an effective general strategy to fight bloat[13]. As already introduced in the previous section, the *maximal depth restriction* strategy is also used as an additional measure to

[11]for a general introduction to GP evolution as a multiobjective problem see Section 4.3, Chapter 4.
[12]see Section 3.1.3, Chapter 3.
[13]see parametric parsimony pressure strategies in Section 4.5.4.2, Chapter 3.

curb bloat[14], although due to the particular nature of the hybrid approach some bloat reduction is likely: in particular a reduction of semantical introns[15] can be expected.

Curiously, Eq. (5.1) can be considered a synthesis of the fitness functions used in Blickle (1996), where the number of tuning parameters is not penalised, and in Alvarez (2000), where illegal syntax trees are not discouraged.

5.2.6 Termination criteria

The evolution is terminated if the root mean square error of the best-so-far individual goes below a predefined threshold[16]. The evolution in any case ends when the number of generations reaches the limit specified by the user[17].

5.3 Enhancements to hybrid GP algorithm

The hybrid GP algorithm described in the previous sections is substantially similar to the implementation developed by Alvarez (2000), although some important features have been introduced to improve metamodels accuracy and to reduce bloat. Such implementation, as it has been presented so far, will be termed *"reference"* in the following sections.

Main focus of the research following HyGP development has been the exploration of new strategies to improve the quality of the evolved metamodels in terms of accuracy and computational cost. A variety of issues undermining the efficiency of the evolutionary search have been addressed, resulting in a set of different enhanced versions of the HyGP code. Such implementations are described in the following sections.

5.3.0.1 Influence of copies in the population

The first issue that has been analysed is the role of copies on HyGP evolution. The proliferation of copies of the same individual increases the risk of fitness stagnation and premature convergence due to loss of diversity, as observed by Koza (1992), Srinivas and Patnaik (1994) and Chellapilla (1997). Whereas a few researchers have allowed identical individuals in their GP populations (Lew et al. 2006), others have implemented strategies for the removal of "clones", both in GP (De Jong and Pollack 2003) and in GA (Affenzeller

[14]see Section 4.5.4.1, Chapter 3.
[15]see Section 4.4.2.1, Chapter 3.
[16]See parameter THRESHOLD described in Section B.1.1, Appendix B.
[17]See parameter G described in Section B.1.1, Appendix B.

and Wagner 2004). In HyGP reference version, copies are replaced by newly generated individuals, as described in Section 5.2.4.1.

In order to study the effect of removing copies from the elite on the accuracy of the evolved models, HyGP code has been modified so that during reproduction copies are not deleted. This alternative HyGP implementation will be referred to in the experiments shown in Section 5.3.4 as *"with COPIES"* version.

5.3.0.2 Effect of periodic population regeneration

A second enhanced HyGP implementation has been developed to maximise population variability through the periodical deletion and regeneration of part of the population. Evolution is modified so that every six generations, the common genetic operations used in HyGP (reproduction, crossover, mutation) are not applied. Instead, a large percentage of the population (set to 60%) is deleted and the empty places are filled with new individuals. Such individuals are generated either by random initialisation or by joining existing structures using a binary function randomly selected among the available primitives.

The periodic deletion of part of the population has been inspired by the ALPS approach used by Hornby (2006) to improve evolutionary algorithms and by the *cascade* operator presented in Vladislavleva (2008) (see Section 3.1.5.6, Chapter 3). The strategies used for the generation of new individuals, instead, have been adapted from the *epoch replacement* operator described in Whigham (1995) (see Section 4.1.1, Chapter 4) and from the *multigenic chromosome approach* used by Ferreira (2001) (see Section 4.8, Chapter 4).

This enhanced HyGP version will be referred to as *"KILLandFILL"* in Section 5.3.5.

5.3.0.3 Sensitivity to the number of initial guesses

As described in the opening sections of the chapter, the reduction of the computational cost associated with parameter tuning is key to the success of hybrid GP approaches. In a HyGP experiment the user has the possibility to set the number of tuning processes each ancestor undergoes[18]. Increasing the number of initial guesses for the SQP optimiser improves fitness evaluation robustness for each individual, but it can lead to excessive computational overhead, considering that populations are typically made of hundreds of ancestors[19].

[18]see Section 5.2.2
[19]see Chapter 6 for examples.

In order to assess the influence of deterministic tuning on the accuracy of the generated metamodels, the number of SQP random initial guesses has been increased from 2, the default number of initial guesses, to 10. The HyGP implementation featuring this extra tuning effort will be referred to as *"10guesses"* in Section 5.3.5.

5.3.0.4 Role of redundancy

Sufficiency and redundancy have paramount importance in genetic programming. GP is able to generate good quality models even though important primitives are not made available by the user, as seen in Section 4.2, Chapter 4. Moreover, in Section 4.8, Chapter 4 it has been shown how the availability of multiple evolutionary paths to generate the solution increases the probability of evolutionary search success.

In non-hybrid GP the availability of multiple evolutionary paths is reduced by the premature disappearance of numerical coefficients from individuals (Vladislavleva 2008, pag. 83). A decrease in the numerical coefficients in fact reduces the number of genotypes (or syntax trees) that map to the same phenotype (or mathematical model).

The use of unary functional primitives that insert extra numerical coefficient is a simple but effective solution to this problem. For example, the unary primitives $Shift_C$ and $Scale_C$ defined by Vladislavleva (2008), which encapsulate an addition and a multiplication by a numerical coefficient, have been reported to successfully tackle the premature disappearance of numerical nodes (Vladislavleva 2008, pag. 83).

An enhanced HyGP implementation has therefore been developed to explore the benefits of a new unary function called "SHIFT", inspired by Vladislavleva's $Shift_C$ and $Scale_C$ operators. The SHIFT primitive modifies the parameters insertion algorithm[20], introducing an extra addition in the node of the syntax tree where it is located. For example, if we consider the structure $Z1 * Z2$, parameter insertion without SHIFT would produce the individual:

$$Z1 * Z2 \Longrightarrow a_0 + a_1 * Z1 * Z2 \tag{5.4}$$

whereas if the ancestor was $Z1 * SHIFT(Z2)$, the complete individual would be:

$$Z1 * SHIFT(Z2) \Longrightarrow a_0 + Z1 * (a_1 + a_2 * Z2) \tag{5.5}$$

[20]see Section 5.2.2.

Although simple, the SHIFT operator is an effective way to increase the number of numerical coefficients associated to an ancestor, and so the number of primitives combinations that are mapped onto the same model or submodel.

The HyGP implementation featuring the SHIFT primitive will be referred to as *"SHIFT"* in Section 5.3.5.

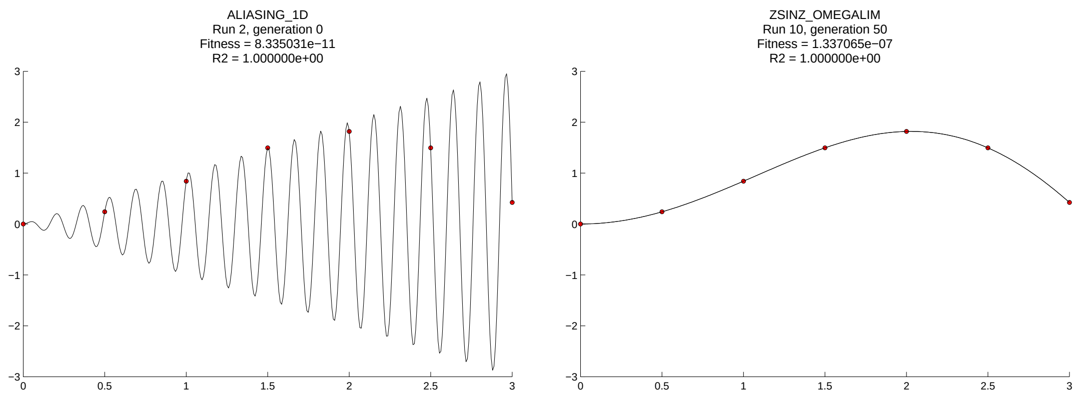
5.3.0.5 Effect of cross-validation strategy on generalisation ability

Cross-validation is a well known class of strategies used to increase the generalisation ability of a model, as seen in Section 1.2.2, Chapter 1. A fifth HyGP implementation has then been developed to assess if the use of the *hold out* method, that is splitting the building data set and using two different subsets for model tuning and model evaluation, improves the generalisation ability of the evolved models.

The building data set was generated by a genetic algorithm called *permutational GA* (Bates et al. 2004) and then split to extract two different subsets: of the two, one subset (*tuning* data subset) was used for tuning, the other (*evaluation* data subset) was used for the final model fitness evaluation. Permutational GA was specifically used as it guarantees not only an optimal or near-optimal latin hypercube distribution in each single subset (tuning and evaluation subsets) but also in the merged data set (complete building data set). The HyGP implementation enhanced with the *hold out* strategy will be referred to as *"NestedDOE"* in Section 5.3.5. In all the experiments that have been performed the size of the evaluation data subset is a third of the overall size of the (merged) building data set.

5.3.0.6 Effect of additional measures to counter overfitting

Differently from traditional GP (Koza (1992)), hybrid GP implementations allow to tailor the tuning strategies to the particular location of the numerical coefficients in the syntax tree. This feature has deserved a particular attention during HyGP development, as initial experiments highlighted a characteristic phenomenon involving trigonometric primitives. If $sine$ or $cosine$ are unary nodes of a syntax tree, SQP algorithm usually tends to improve individual fitness value increasing the circular frequency and trimming the amplitude of these functions. As this mechanism allows to build highly fit individuals in the span of few generations, $sine$ and $cosine$ are used by GP far too often, even when the solution does not have a periodic behaviour. However, most of the times the corresponding model is affected by overfitting due to high circular frequency oscillations (noise). In Fig. 5.5 the problem is illustrated by the comparison of two different individuals produced by HyGP for the same test problem, the symbolic regression of the expression $Z*sin(Z)$ from seven points uniformly distributed in $[0, \pi]$. The evolved individuals are shown in Fig. 5.5: the function in Fig. 5.5A has the same root mean square error as the function in Fig. 5.5B, which is approximately zero. Although the former clearly has far worse generalisation ability than the latter, HyGP is not able to discriminate between them using the given building data set.



(A) NO circular frequency optimisation (B) WITH circular frequency optimisation

FIGURE 5.5: Symbolic regression of $Z * sin(Z)$. Seven uniformly distributed points in $[0, \pi]$, shown by dots, are used as training set

A HyGP implementation has been developed to tackle overfitting caused by $sine$ and $cosine$. An algorithm has been added to recognise trigonometric terms like $sin(cZ)$ and $cos(cZ)$ and to constrain the value of their arguments c, called circular frequencys. In mathematical terms, a numerical coefficient c is recognised by HyGP as a circular frequency value $a_{i,k}$ only if c appears as argument of $sine$ or $cosine$ and it is multiplied by a variable, as in the following:

$$sin(c * Z_i) \Longrightarrow a_{i,k} = c \tag{5.6}$$

$$cos(c * Z_i) \Longrightarrow a_{i,k} = c \tag{5.7}$$

where Z_i is the symbol of the i-th independent variable and k refers to the number of circular frequencys recognised as circular frequencys for Z_i. If it is assumed that the maximum number of oscillations[21] (periods) due to $sine$ or $cosine$ terms that metamodels can have in each variable's range r_i is P_{max}, then the maximum magnitude of the circular

[21] see parameter MAX_N_PERIODS in Section B.1.1, Appendix B.

frequency $\omega_{i,lim}$ multiplying Z_i is:

$$\omega_{i,lim} = \frac{2\pi P_{max}}{r_i} \tag{5.8}$$

The deterministic search for the best set of numerical parameters is then biased to penalise set of values x_i that feature circular frequencys outside the range $[-\omega_{i,lim}, \omega_{i,lim}]$. The bias has been introduced adding to the traditional SQP error metrics (Madsen et al. 2002) a penalisation made of as many terms $g_{i,k}$ as the number of circular frequencys recognised. The minimisation problem solved using SQP has been reformulated as:

$$find \quad \mathbf{x} \tag{5.9}$$

$$minimising \quad F_{SQP}(\mathbf{x}) = \frac{1}{2}\sum_{j=1}^{m}\left(\hat{f}_j(\mathbf{x}) - f_j\right)^2 + \sum_{i=1}^{n_{var}}\sum_{k=1}^{n_{puls_i}} g_{i,k}(a_{i,k}, \omega_{i,lim}) \tag{5.10}$$

where $\mathbf{x}$ is the set of unknown parameters' values of the GP individual $\hat{f}$ being tuned, f_j is the observed output in sample j, $\hat{f}_j$ is the output produced by the GP individual in sample j, m is the number of samples (size of the building data set), n_{var} is the number of independent variables and n_{puls_i} is the number of circular frequencys found for variable Z_i in the individual undergoing tuning. The penalisation terms $g_{i,k}$ that direct the search for $\mathbf{x}$ away from high circular frequencys are defined as functions of the particular numerical coefficients in $\mathbf{x}$ recognised as circular frequencys $a_{i,k}$:

$$g_{i,k}(a_{i,k}, \omega_{i,lim}) = \begin{cases} 0 & \text{if} \quad a_{i,k} \in [-\omega_{i,lim}, \omega_{i,lim}] \\ e^{\left(|a_{i,k}| - \omega_{i,lim}\right)^2} - 1 & \text{if} \quad a_{i,k} \notin [-\omega_{i,lim}, \omega_{i,lim}] \end{cases} \tag{5.11}$$

The HyGP implementation described above has been used to generate the model plotted in Fig. 5.5B. Such implementation will be referred to as *"Omegalim"* is Section 5.3.5.

5.3.0.7 Effect of different fitness function formulations

Attention has also been paid to the study of alternative formulations of the fitness function and their effect on the quality of the evolved models. As described in Section 4.3, Chapter 4, more than one property or objective contribute to the definition of a high-quality metamodel. In Section 4.5.4, Chapter 4 different ways have been shown to translate these properties into mathematical quantities to either encourage or penalise them during the evolution.

Three HyGP implementations have been developed to analyse the effect of three different fitness functions on metamodels accuracy. These implementations will be termed:

- *normFIT*
- *normFITdiv*
- *MinMax*

In *normFIT* implementation the fitness function is defined as follows:

$$F\,(i,t) = a_1F_1(i,t) + a_2F_2(i,t) + a_3 * 10^6 * F_3(i,t) + a_4F_4(i,t) \tag{5.12}$$

$$a_1 + a_2 + a_3 + a_4 = 1 \tag{5.13}$$

where $F_1(i,t)$ is redefined as a normalised RMSE or RMSNE[22]:

$$F_1(i,t) = RMSNE(i,t) \tag{5.14}$$

$$RMSNE(i,t) = \sqrt{\frac{1}{m}\sum_{j=1}^{m}\left[\frac{\hat{f}_{i,j,t} - f_j}{f_j}\right]^2} \tag{5.15}$$

where $\hat{f}_{i,j,t}$ is the value returned by individual i in generation t at sample point j and f_j is the known output at sample point j. The sum is made on the total number of sample cases m in the building data set.

The fitness function formulation used in *normFITdiv* implementation maintains the general definition shown in Eq. (5.12), but emphasises the variation with respect to the average RMSNE in the elite of the previous generation:

$$F_1(i,t) = \frac{RMSNE(i,t)}{\overline{RMSNE}(t-1)} \tag{5.16}$$

normFIT and *normFITdiv* implementations have been introduced to check whether the evolution benefits from the normalisation of the error.

In *MinMax* implementation the fitness function is defined in a different way, using a MinMax approach, as shown in Eq. (5.17):

$$F = max\left\{a_1 * 10 * F_1(i,t), a_2F_2(i,t), a_3 * 10^6 * F_3(i,t), a_4F_4(i,t)\right\} \tag{5.17}$$

$$a_1 + a_2 + a_3 + a_4 = 1 \tag{5.18}$$

[22]RMSNE as, differently from RMSE, the error is normalised by the known output for each sample point.

where the objectives F_1, F_2, F_3 and F_4 are defined as in the reference HyGP implementation described in Section 5.2.5. The MinMax approach is tested to understand whether optimising the worst objective at a time instead of the linear combination of all four can increase the number of evolutionary paths towards the solution.

5.3.1 Experimental methodology

The nine HyGP implementations presented in the previous section have been tested on five symbolic regression problems, defined in Section 5.3.1.1.

For each implementation and each test problem an *experiment* consisting of 10 independent *runs* was performed (50 experiments, 500 runs in total). The RMSE of the best individual returned at the end of the evolution by each run was computed on building as well as on a validation data set. For each experiment two RMSE samples made of 10 values each (one for building, one for validation) were then produced.

Both RMSE samples produced by model evaluation on building and validation data sets were analysed. Model irregularities gone unnoticed during the evolution may in fact appear during the reevaluation on the validation data set. To avoid any bias in the conclusions due to the failure of a few runs (phenomenon termed “bimodal distribution” by Soule and Foster (1998a) and described in Section 3.2.2, Chapter 3), two kind of “pathologies” have been looked for, as done also by Vladislavleva (2008): *singularities*, related to the presence of asymptotes in particular points of the data set, and *excessive RMSE*. The percentage of individuals affected by singularities will be indicated with the symbol “%∞” in Section 5.3.2. Individuals scoring an RMSE larger then the arbitrarily chosen threshold of 100 were considered suffering from excessive RMSE pathology and classified as "bad". The percentage of “bad” individuals will be indicated with the symbol “%bad” in Section 5.3.2.

RMSE values of “pathologic” individuals were then removed from both RMSE samples and a first analysis was performed on the reduced RMSE samples. At this stage boxplots[23] were used to formulate hypotheses regarding the performances of the different HyGP implementations. Indicators as median and interquartile range were considered more appropriate than mean and variance as RMSE observations could not be assumed to have a normal distribution.

[23]see Section 3.2.2, Chapter 3.

A statistical analysis was then carried out on reduced RMSE samples produced by reevaluation on validation data set to confirm or discard hypotheses emerged from previous analysis. Two-sided Kruskal-Wallis test and Wilcoxon rank sum test[24], both non-parametric, were used. Kruskal-Wallis test was carried out first to check if there was significant evidence of a difference in median among all the RMSE samples for each test problem. In case significant evidence had been detected, Wilcoxon test was used to compare the median between single pairs of RMSE samples. A 5% significance level was assumed in all statistical tests.

Parametric tests as one-way ANOVA (Upton and Cook 1996) were used only in a few cases and only as a reference, as RMSE observations in the samples could not be assumed to be normal.

5.3.1.1 Test functions

The nine HyGP implementations described in Section 5.3 were tested on the following symbolic regression test problems:

$$f_1(z_1, z_2) = \frac{e^{-(z_1-1)^2}}{1.2 + (z_2 - 2.5)^2} \tag{5.19}$$

$$f_2(z) = e^{-z} z^3 cos(z) sin(z) \left[cos(z) sin^2(z) - 1\right] \tag{5.20}$$

$$f_3(z_1, z_2) = \frac{(z_1 - 3)^4 + (z_2 - 3)^3 - (z_2 - 3)}{(z_2 - 2)^4 + 10} \tag{5.21}$$

$$f_4(z_1, z_2) = [30 + z_1 sin(z_1)] \left(4 + e^{-z_2}\right) \tag{5.22}$$

$$f_5(z_1, z_2) = \left(z_2 - 5.1\frac{z_1^2}{4\pi^2} + 5\frac{z_1}{\pi} - 6\right)^2 + 10\left(1 - \frac{\pi}{8}\right) cos(z_1) + 10 \tag{5.23}$$

The test functions listed above were selected as benchmarks for a simple reason: they are compact, they have a highly non linear behaviour in the domain under analysis and they can be easily conceived as being obtained combining non linear mathematical functions. Korns (2011) calls this class of test functions “simple test formulas”. However the very same author acknowledges that this “simple” test functions represent a demanding challenge for GP, as “for these intractable problems state-of-the-art symbolic regression engines fail to return a champion with the correct formula” (Korns 2011, p. 130). If GP is not able to return a correct formula for “simple test formulas” then according to Korns

[24]See Section 3.2.2, Chapter 3.

"user interest and trust in the symbolic regression system wanes", GP "loses its differentiation from other black box machine learning techniques as support vector regression or neural nets" and thirdly, being GP a "technique for returning, not just coefficients, but a correct formula", "if this claim cannot be fulfilled [...] a serious reputational issue will develop and research money will flow in other directions" (Korns 2011, p. 130).

Therefore, the test functions were selected among the ones used by other researchers on symbolic regression through GP. The first three were taken from Vladislavleva (2008), who selected these and others functions as "the most difficult problems" from an original set of thirteen functions introduced by Keijzer (2003): *Kotanchek* function in Eq. (5.19), *Salustowicz* function in Eq. (5.20) and *RatPol2D* function in Eq. (5.21). The fourth function, Eq. (5.22), was taken from Hock and Schittkowski (1981) and it will be called *Hock* function in the following. The last function (Eq. (5.23)), known as *Branin-Hoo* function, was excerpted from Viana and Haftka (2009). The plots of the functions are shown in Fig. 5.6.

5.3.1.2 Building and validation data sets

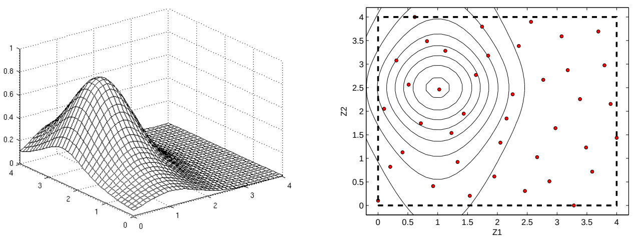
The building data sets of the test functions described in Section 5.3.1.1 were generated coupling points selected by optimal latin hypercube sampling[25] with the corresponding values of the test function. A GA-based code was used to generate the DoEs (Bates et al. 2004). The sampling regions are reported in the second column of Table 5.2 and shown delimited by a dashed line in Fig. 5.6: inside each dashed rectangle, the dots represent the DoE points (for Salustowicz test function the rectangle shrinks to a segment).

The validation data set was instead generated using a full-factorial DoE[26]. For a reliable evaluation of the quality of the individual, a larger number of points was generated with respect to the building data set. The sampling region was also extended for the validation data set, in order to assess model extrapolation ability, as done in Vladislavleva (2008).

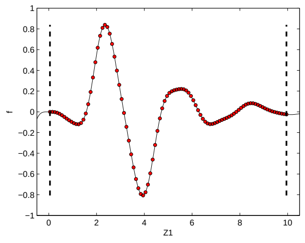
The number of points, the upper and lower bounds and the other characteristics of the validation data sets for each test problem are given in Table 5.2.

[25]see Section 1.2.5, Chapter 1.

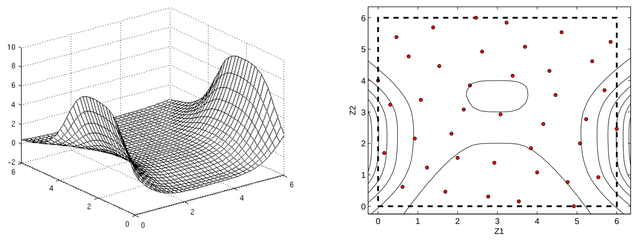
[26]see Section 1.2.5, Chapter 1.



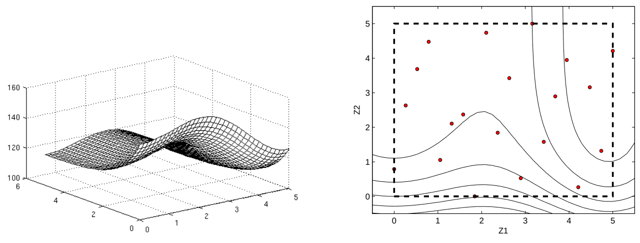
(A) Kotanchek



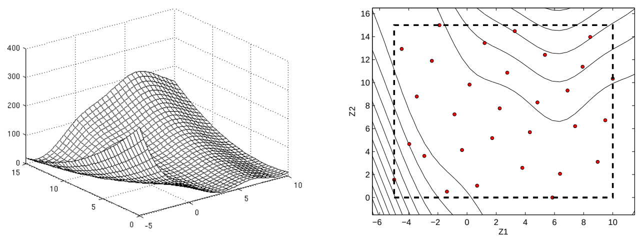
(B) Salustowicz



(C) RatPol2D



(D) Hock



(E) Branin-Hoo

FIGURE 5.6: Shape of the test functions (left) and corresponding regions which building and validation data sets are sampled from (right)

TABLE 5.2: Training and validation data sets for the test functions. In the middle column the domain used for optimal latin hypercube sampling is defined. In third column the validation data set is defined using the notation $Z_i = [a : dx : b]$. This means that values of variable Z_i are sampled from a to b with a step dx.

test function	building data (OLH DoE)	validation data (full factorial DoE)
Kotanchek	40 points $0 \leq Z1 \leq 4$ $0 \leq Z2 \leq 4$	2025 points $Z1, Z2 = [-0.2 : 0.1 : 4.2]$
Salustowicz	100 points $Z1 = [0.05 : 0.1 : 9.95]$ (full factorial DoE)	221 points $Z1 = [-0.5 : 0.05 : 10.5]$
RatPol2D	40 points $0 \leq Z1 \leq 6$ $0 \leq Z2 \leq 6$	1156 points $Z1, Z2 = [-0.25 : 0.2 : 6.35]$
Hock	20 points $0 \leq Z1 \leq 5$ $0 \leq Z2 \leq 5$	441 points $Z1, Z2 = [-0.5 : 0.3 : 5.5]$
Branin-Hoo	30 points $-5 \leq Z1 \leq 10$ $0 \leq Z2 \leq 15$	1369 points $Z1 = [-6.5 : 0.5 : 11.5]$ $Z2 = [-1.5 : 0.5 : 16.5]$

5.3.2 Results and discussion

The results of the experiments described in Section 5.3.1 and its analysis will be detailed in the following sections.

To start with, in Section 5.3.3 standard GP, based on classic genetic programming framework, is put to the test with HyGP on the five test regression problems introduced in Section 5.3.1.1. The results obtained motivate the choice of not progressing further with experimentation on standard GP.

In Section 5.3.4 only two HyGP implementations are compared, the *reference* formulation introduced in Section 5.2, in which copies are deleted from the *elite*, and *with COPIES* implementation, described in Section 5.3.0.1, in which copies are instead left in the elite. This first comparison is performed on all the test functions.

Finally, in Section 5.3.5 all the remaining implementations are compared with the original HyGP formulation (*reference*), analysing separately each test function.

5.3.3 Preliminary tests: standard GP against hybrid GP

The HyGP implementation described in Section 5.2 differs from standard genetic programming mainly for the addition of the deterministic tuning algorithm. Little is known about what effect numerical coefficient tuning has on generalisation ability, robustness and computational cost of model generation once added to a Koza's style genetic programming implementation.

To shed a light on this topic and assess if the main advantages of hybrid approaches generally introduced in Section 5.1 can be expected, the *reference* HyGP implementation has been compared to a similar GP implementation in which tuning was switched off. The results are reported in the following for each test problem.

5.3.3.1 Kotanchek test problem

The GP parameters used for both experiments, *reference* and *standard GP* are listed in Table 5.3. In table 5.4 pathologies on training and validation data sets are reported:

TABLE 5.3: Kotanchek test problem: main parameters used in the experiments

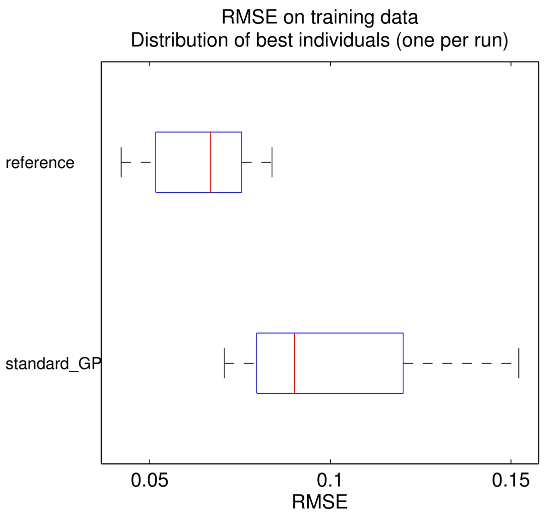
Population size:	200
Generations:	50
Primitives:	+, -, *, / (protected), square, cube, sin, cos, exp
No. of fitness cases:	40
Initialisation Maximum depth initialised trees Maximum depth initialised trees	50% full + 50% grow 4 2
Maximal depth restriction	50
Reproduction rate Crossover rate Mutation rate	20% 40% 40%
a_2 (W_COMPLEXITY) a_3 (W_N_CORRECTIONS) a_4 (W_SIZE)	0.001 0.1 0.0001

reference appears to be more robust than *standard GP*, for which only 4 out of 10 runs produce a defined model on the validation data set. On the other hand, the best individual

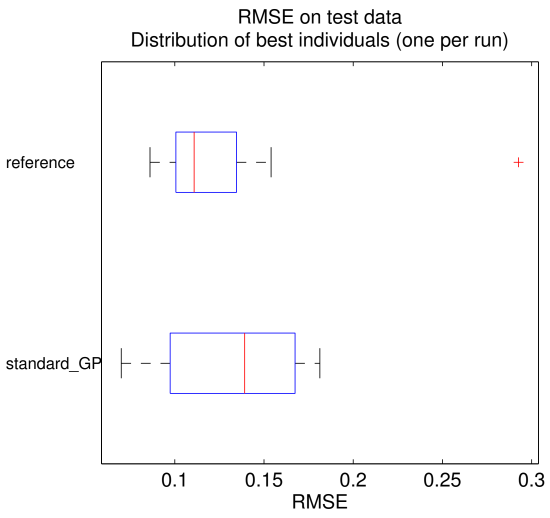
TABLE 5.4: Comparison with standard GP: pathologies on building and validation data sets for Kotanchek test case. All experiments consisted of 10 runs: %∞ is the percentage of individuals not defined, %bad is the percentage of individuals with a RMSE>100 on the particular data set. Median and IQR are computed on the remaining individuals.

Kotanchek	Building data set				Validation data set			
			RMSE				RMSE	
	%∞	%bad	median	IQR	%∞	%bad	median	IQR
standard_GP	0	0	9.003E-02	4.052E-02	60	0	1.392E-01	7.001E-02
reference	0	0	6.676E-02	2.378E-02	10	0	1.109E-01	3.394E-02

is returned by *standard GP* ($RMSE$ = 7.005910e-002 R^2= 8.816135e-001) as shown by RMSE boxplots in Fig. 5.7. The better performance with respect to *reference* is however not statistically significant (p-value = 0.64343). The difference in accuracy on building data set and validation data set for individuals produced by *reference* may be explained by overfitting.



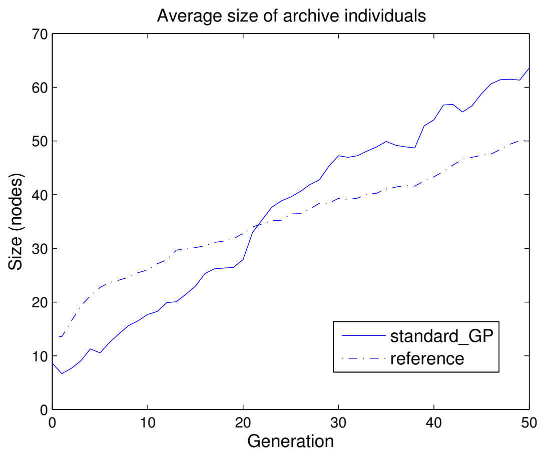
(A) $RMSE$ on building data set



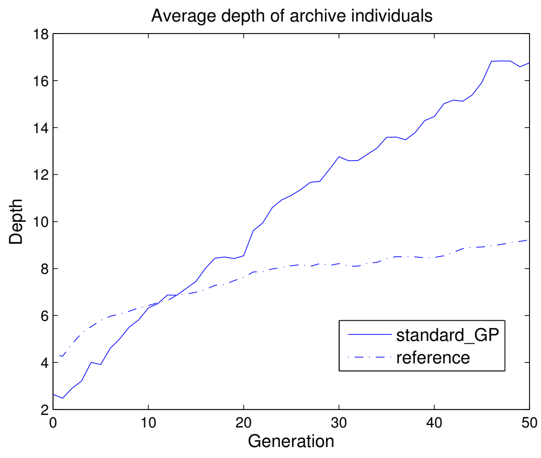
(B) $RMSE$ on validation data set

FIGURE 5.7: Kotanchek test case, comparison between standard GP and reference implementation: $RMSE$ boxplots for building and validation data set. Individuals with RMSE>100 as well as undefined ones are not included

In Fig. 5.8 the average size and depth of the individuals in the archive (40) are plotted against the generations: it can be observed that tuning contributes effectively to limiting the depth of the individuals.



(A) size dynamics



(B) depth dynamics

FIGURE 5.8: Kotanchek test case, comparison between standard GP and reference implementation: average size and depth of the 40 individuals belonging to the archive

5.3.3.2 Salustowicz test problem

The GP parameters used for both experiments, *reference* and *standard GP* are listed in Table 5.5.

TABLE 5.5: Salustowicz test problem: main parameters used in the experiments

Population size:	300
Generations:	50
Primitives:	+, -, *, / (protected), square, cube, sin, cos, exp, negative exp, opposite, reciprocal
Number of fitness cases:	100
Initialisation Maximum depth initialised trees Maximum depth initialised trees	50% full + 50% grow 4 2
Maximal depth restriction	50
Reproduction rate Crossover rate Mutation rate	20% 40% 40%
a_2 (W_COMPLEXITY) a_3 (W_N_CORRECTIONS) a_4 (W_SIZE)	1.0e-6 0.01 1.0e-6

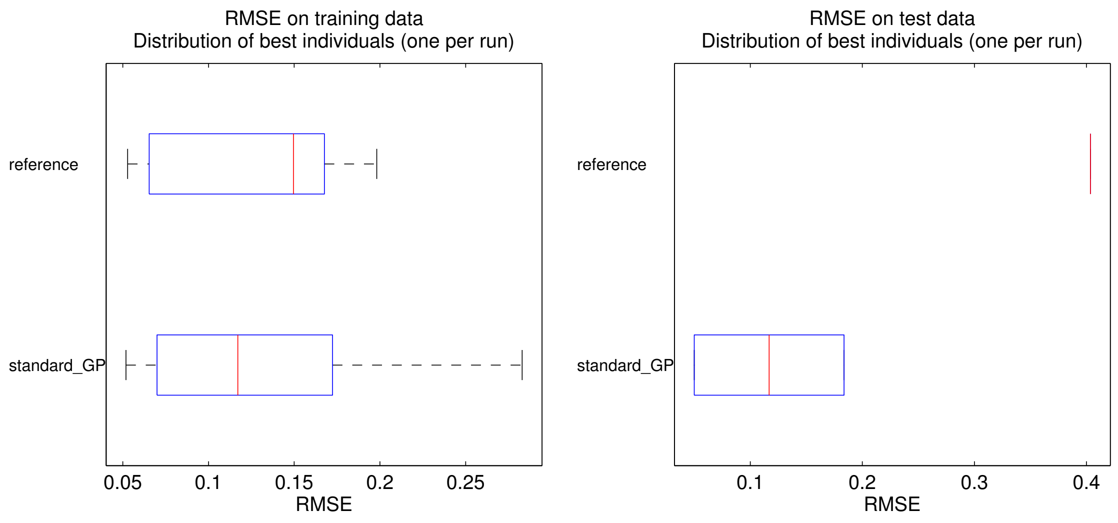
Pathologies on training and validation data sets are shown in table 5.6.

TABLE 5.6: Comparison with standard GP: pathologies on building and validation data sets for Salustowicz test case. All experiments consisted of 10 runs: $\%\infty$ is the percentage of individuals not defined, %bad is the percentage of individuals with a RMSE>100 on the particular data set. Median and IQR are computed on the remaining individuals.

Salustowicz	Building data set				Validation data set			
			RMSE				RMSE	
	$\%\infty$	%bad	median	IQR	$\%\infty$	%bad	median	IQR
standard_GP	0	0	1.171E-01	1.023E-01	80	0	1.168E-01	1.337E-01
reference	0	0	1.496E-01	1.023E-01	90	0	4.036E-01	0.000E+00

The generalisation properties of the individuals produced by both experiments appear really poor: considering the very little proportion of individuals defined on the validation data set, *standard GP* returned the best model (R^2 = 9.72844e-01 , $RMSE$ = 4.99181e-02), as shown by $RMSE$ boxplots in Fig. 5.9. The Salustowicz test case, although a relatively simple univariate function, will prove a hard regression problem also in the experiments shown in the following chapters.

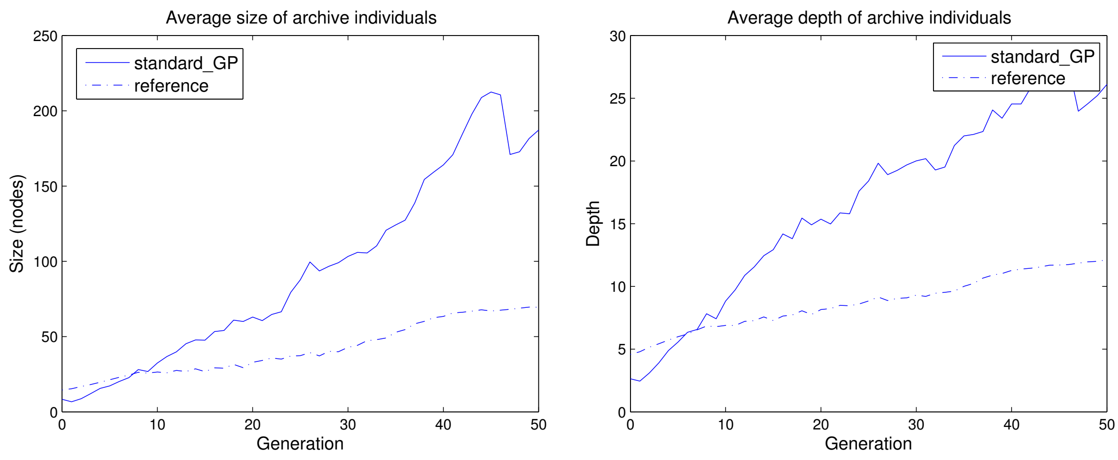
The average size and depth of the individuals in the archive (60) are plotted against the generations in Fig. 5.10. Differently from the Kotanchek test case (Section 5.3.3.1), it



(A) $RMSE$ on building data set (B) $RMSE$ on validation data set

FIGURE 5.9: Salustowicz test case, comparison between standard GP and reference implementation: R^2 boxplots for building data set and validation data set. Individuals with RMSE>100 as well as undefined ones are not included

can be observed that tuning effectively curbs code growth limiting both the size and the depth of the individuals.



(A) size dynamics (B) depth dynamics

FIGURE 5.10: Salustowicz test case, comparison between standard GP and reference implementation: average size and depth of the 10 individuals belonging to the archive

5.3.3.3 RatPol2D test problem

The GP parameters used for both experiments, *reference* and *standard GP* are listed in Table 5.7.

Pathologies on training and validation data sets are shown in Table 5.8.

TABLE 5.7: RatPol2D test problem: main parameters used in the experiments

Population size:	200
Generations:	50
Primitives:	+, -, *, / (protected), square, cube, sin, cos, exp, reciprocal
Number of fitness cases:	40
Initialisation Maximum depth initialised trees Maximum depth initialised trees	50% full + 50% grow 4 2
Maximal depth restriction	50
Reproduction rate Crossover rate Mutation rate	20% 40% 40%
a_2 (W_COMPLEXITY) a_3 (W_N_CORRECTIONS) a_4 (W_SIZE)	0.0001 0.1 0.00001

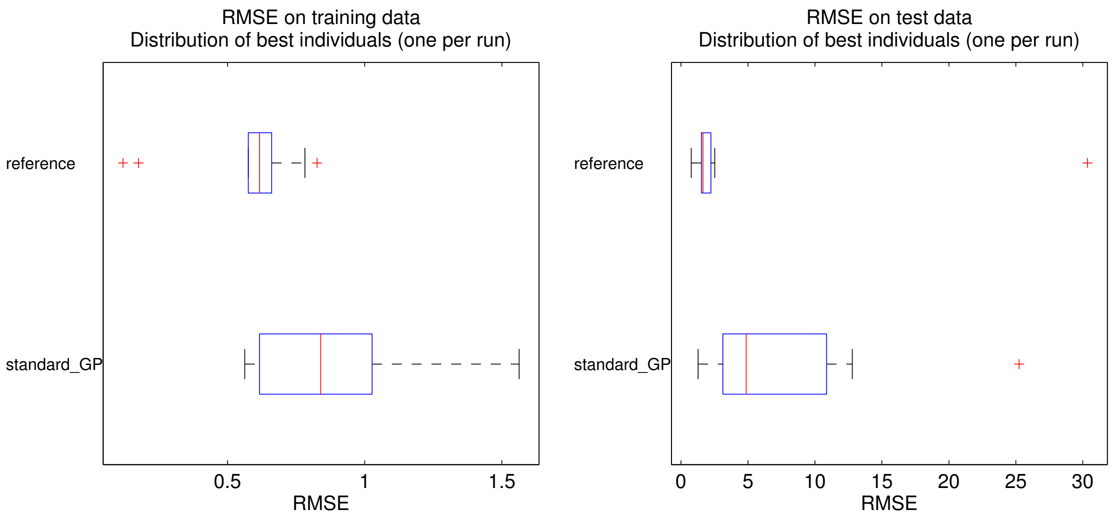
TABLE 5.8: Comparison with standard GP: pathologies on building and validation data sets for RatPol2D test case. All experiments consisted of 10 runs: %∞ is the percentage of individuals not defined, %bad is the percentage of individuals with a RMSE>100 on the particular data set. Median and IQR are computed on the remaining individuals.

RatPol2D	Building data set				Validation data set			
			RMSE				RMSE	
	%∞	%bad	median	IQR	%∞	%bad	median	IQR
standard_GP	0	0	8.397E-01	4.103E-01	30	0	4.870E+00	7.755E+00
reference	0	0	6.165E-01	8.537E-02	0	10	1.634E+00	7.017E-01

Reference turns out to be more robust than *standard GP*, for the larger number of individuals defined on the validation data set, and more consistent, as the $RMSE$ interquartile range is lower than in *standard GP* case.

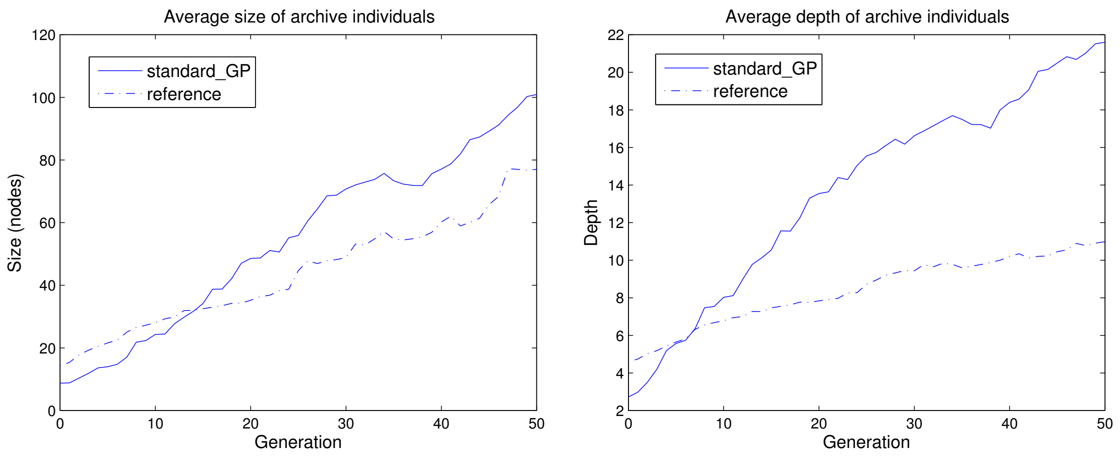
Reference also returns the best model, as shown in $RMSE$ boxplots of Fig. 5.11. The difference in median of the validation set $RMSE$ distributions is not statistically significant according to Wilcoxon rank-sum test (p-value = 0.063969).

In Fig. 5.12 the average size and depth of the individuals in the archive (40) are plotted against the generations: tuning in this case affects mainly depth dynamics.



(A) $RMSE$ on building data set (B) $RMSE$ on validation data set

FIGURE 5.11: RatPol2D test case, comparison between standard GP and reference implementation: R^2 boxplots for building and validation data set. Individuals with RMSE>100 as well as undefined ones are not included



(A) size dynamics (B) depth dynamics

FIGURE 5.12: RatPol2D test case, comparison between standard GP and reference implementation: average size and depth of the 10 individuals belonging to the archive

5.3.3.4 Hock test problem

The GP parameters used for both experiments, *reference* and *standard GP* are listed in Table 5.9.

In Table 5.10 pathologies on training and validation data sets are reported: all the individuals produced by both *reference* and *standard GP* are defined on the validation data set.

TABLE 5.9: Hock test problem: main parameters used in the experiments

Population size:	200
Generations:	50
Primitives:	+, -, *, / (protected), square, cube, sin, cos, exp
Number of fitness cases:	20
Initialisation Maximum depth initialised trees Maximum depth initialised trees	50% full + 50% grow 4 2
Maximal depth restriction	50
Reproduction rate Crossover rate Mutation rate	20% 40% 40%
a_2 (W_COMPLEXITY) a_3 (W_N_CORRECTIONS) a_4 (W_SIZE)	0.01 0.1 0.001

TABLE 5.10: Comparison with standard GP: pathologies on building and validation data sets for Hock test case. All experiments consisted of 10 runs: $\%\infty$ is the percentage of individuals not defined, %bad is the percentage of individuals with a RMSE$>$100 on the particular data set. Median and IQR are computed on the remaining individuals.

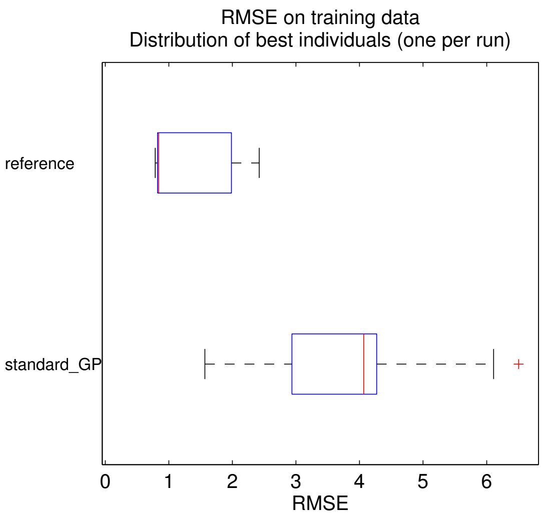
Hock	Building data set				Validation data set			
			RMSE				RMSE	
	$\%\infty$	%bad	median	IQR	$\%\infty$	%bad	median	IQR
standard_GP	0.0	0.0	4.067E+00	1.333E+00	0.0	10.0	9.892E+00	2.485E+01
reference	0.0	0.0	6.165E-01	8.537E-02	0.0	10.0	1.634E+00	7.017E-01

Reference however performs far better than *standard GP*, in terms of both accuracy and consistency, as shown by the $RMSE$ boxplots in Fig. 5.13. The difference in the median of the two validation $RMSE$ distribution is statistically significant, as confirmed by both Wilcoxon rank-sum test (p-value = 0.00064951) and ANOVA test (p-value = 0.0061172).

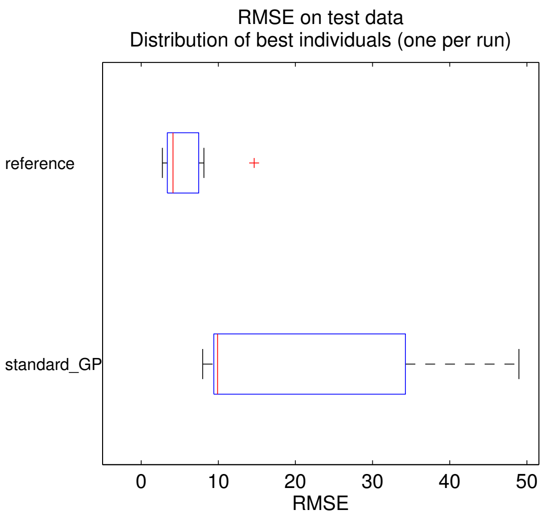
In Fig. 5.12 the average size and depth of the individuals in the archive (40) are plotted against the generations: the effect of the parameter insertion algorithm and tuning on size and depth dynamics is easily recognisable.

5.3.3.5 Branin-Hoo test problem

The GP parameters used for both experiments, *reference* and *standard GP* are listed in Table 5.11.

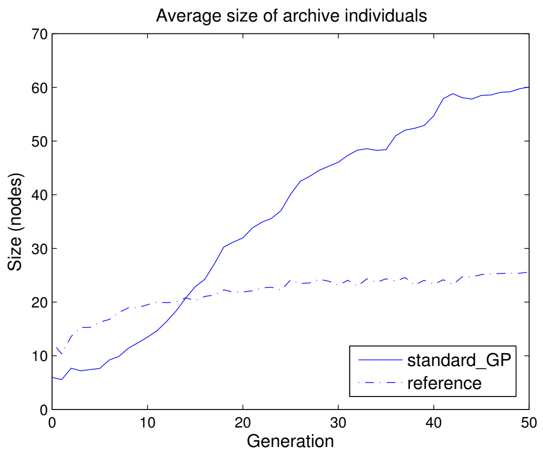


(A) $RMSE$ on building data set

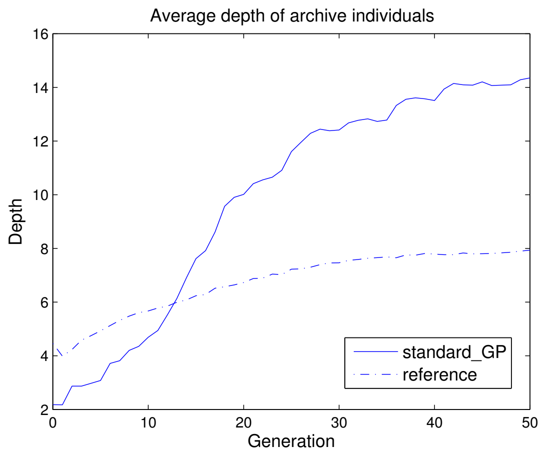


(B) $RMSE$ on validation data set

FIGURE 5.13: Hock test case, comparison between standard GP and reference implementation: R^2 boxplots for building and validation data set. Individuals with RMSE>100 as well as undefined ones are not included



(A) size dynamics



(B) depth dynamics

FIGURE 5.14: Hock test case, comparison between standard GP and reference implementation: average size and depth of the 40 individuals belonging to the archive

In Table 5.12 pathologies on training and validation data sets are reported. *Standard GP* completely failed to return acceptable models, as 9 out of 10 individuals turned out to be undefined on the validation data set and the only one defined has a $RMSE$ larger than 100. Such model was therefore considered "bad" according to the definition given in Section 5.3.1 and excluded from the comparisons.

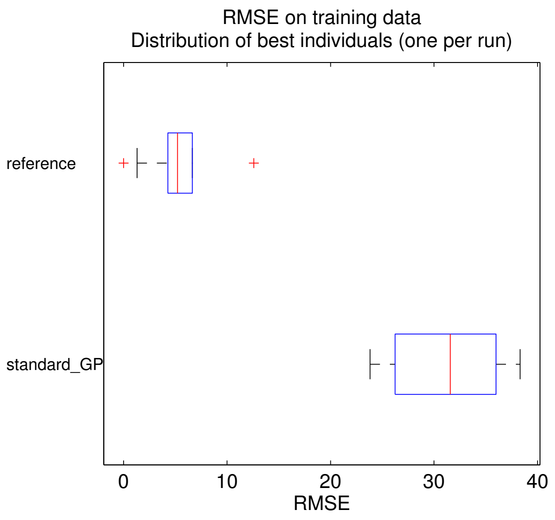
TABLE 5.11: Branin-Hoo test problem: main parameters used in the experiments

Population size:	200
Generations:	50
Primitives:	+, -, *, / (protected), square, cube, sin, cos, exp
Number of fitness cases:	30
Initialisation Maximum depth initialised trees Maximum depth initialised trees	50% full + 50% grow 4 2
Maximal depth restriction	50
Reproduction rate Crossover rate Mutation rate	20% 40% 40%
a_2 (W_COMPLEXITY) a_3 (W_N_CORRECTIONS) a_4 (W_SIZE)	0.001 0.1 0.0001

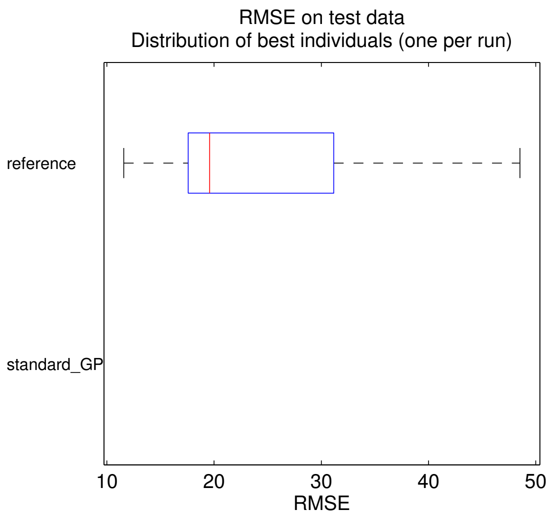
TABLE 5.12: Comparison with standard GP: pathologies on building and validation data sets for Branin-Hoo test case. All experiments consisted of 10 runs: $\%\infty$ is the percentage of individuals not defined, %bad is the percentage of individuals with a RMSE>100 on the particular data set. Median and IQR are computed on the remaining individuals.

Branin-Hoo	Building data set				Validation data set			
			RMSE				RMSE	
	$\%\infty$	%bad	median	IQR	$\%\infty$	%bad	median	IQR
standard_GP	0.0	0.0	3.156E+01	9.752E+00	90.0	10.0	N/A	N/A
reference	0.0	0.0	5.220E+00	2.368E+00	50.0	0.0	1.959E+01	1.357E+01

Reference appears in this case far superior to *standard GP*. Its individuals' $RMSE$ distribution on the validation data set is shown in Fig. 5.15B.



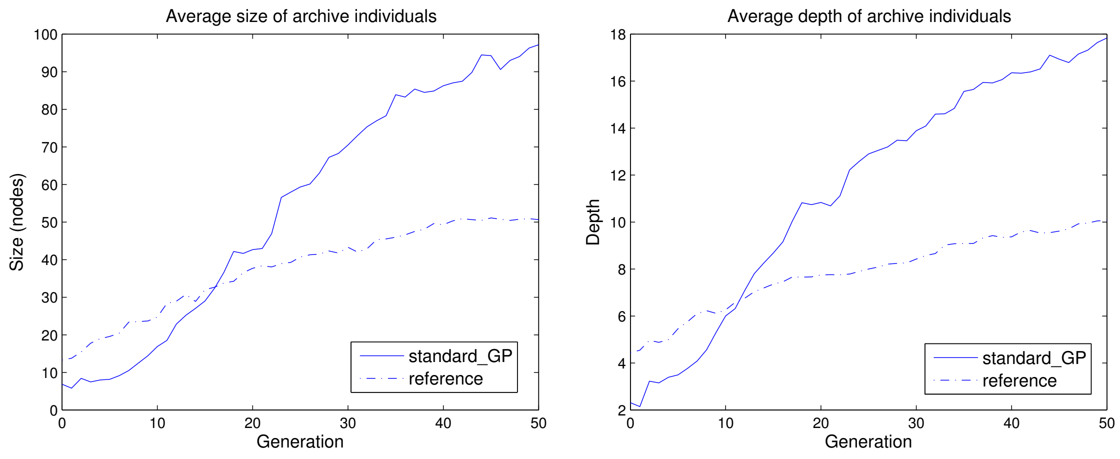
(A) $RMSE$ on building data set



(B) $RMSE$ on validation data set

FIGURE 5.15: Branin-Hoo test case, comparison between standard GP and reference implementation: R^2 boxplots for building and validation data set. Individuals with RMSE>100 as well as undefined ones are not included

In Fig. 5.16 the average size and depth of the individuals in the archive (40) are plotted against the generations. As in the previous test case, parameter insertion algorithm and tuning effectively curbs both size and depth of the generated individuals: the average depth and size of *reference* archive individuals at the end of the evolution are approximately a half of the values of these parameters observed in *standard GP*.



(A) size dynamics (B) depth dynamics

FIGURE 5.16: Branin-Hoo test case, comparison between standard GP and reference implementation: average size and depth of the 40 individuals belonging to the archive

5.3.3.6 Comparison between standard and hybrid GP: conclusions

The comparison between the two genetic programming implementations *reference* and *standard GP* has been carried out taking into account generalisation ability of the returned models and computational cost, which has been estimated recording the time needed to complete a GP evolution.

In Fig. 5.17 training time distributions of the two implementations for all the test cases considered are shown. Boxplots are used and medians are clearly indicated by a red bar.

Training times prove that *standard GP* is far faster than the *reference* implementation of HyGP. However, higher training times appear to be the price that has to be paid for improved generalisation ability and consistency. Evidence backing this conclusion is provided in Table 5.13, where for each test case are reported which implementation scored better for generalisation, best accuracy, archive individuals average size and depth.

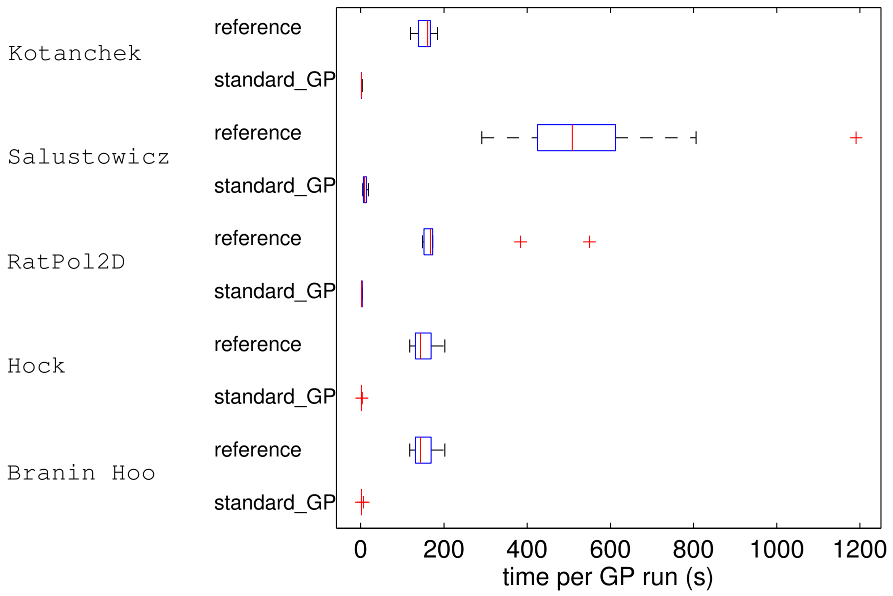


FIGURE 5.17: Runtime comparison between standard GP and reference on the five test problems

TABLE 5.13: Results of the comparison between *reference* HyGP implementation and *standard GP*: the implementation that showed best generalisation performance, returned the best individual and exhibited the smallest average archive individuals' size and/or depth are reported in the columns

Test problem	Best for generalisation	Best for accuracy	Difference in $RMSE$ median statistically significant?	Best for size and/or depth
Kotanchek	*reference*	*standard GP*	NO	*reference*
Salustowicz	identical (bad)	*standard GP*	N/A	*reference*
RatPol2D	*reference*	*reference*	NO	*reference*
Hock	identical	*reference*	YES	*reference*
Branin-Hoo	*reference*	*reference*	YES	*reference*

From Table 5.13 hybrid GP (*reference*) appears to outperform *standard GP* in most of the test problems with respect to the parameters considered (generalisation, accuracy of the best individual, size/depth). Salustowicz test case is a particular test problem, where high proportion of individuals (80-90%) turns out to be undefined on the validation data set, so although *standard GP* performs better than *reference*, this result is not statistically significant.

The comparison shows that the parameter insertion algorithm and consequent parameter tuning is an effective strategy to improve generalisation ability and reduce size/depth of the model generated. However, in order to make the strategy more appealing for practical use in regression problems, the comparison also shows that future research efforts should be directed towards the reassessment of the tuning strategy, to avoid tuning when not contributing effectively to population fitness improvement and so reduce overall training times.

5.3.4 Assessment of the influence of copies

For all the test functions a limit of 50 generations was set. Regarding the population size, for Salustowicz function 300 individuals were used, whereas for all the other cases population size was reduced to 200. The primitives used for each test case are the same as the ones reported in Section 5.3.5.

A first insight into the results is provided by Table 5.14. In the table, the percentage of undefined individuals (column (2) and (6)), the percentage of "bad" individuals (having RMSE $>$ 100 - column (3) and (7)), median and interquartile range (IQR) are shown for each test problem on building and validation data sets. The increase in the number of "bad" and undefined individuals from building to validation data set provides evidence of the importance of reevaluating individuals on a larger data set after model generation. Salustowicz and Branin-Hoo metamodels returned by *reference* implementation are mostly undefined on the validation data set, whereas expressions generated by *with COPIES* are less affected by the problem.

p-values obtained by the comparison of the RMSE samples using Wilcoxon rank sum test and one-way ANOVA test are shown in Table 5.15. The NULL hypothesis to test is whether there is significant evidence of difference between the RMSE medians on each test problem or not. Assuming a 5% significance level, *p*-values returned by Wilcoxon test suggest that there is no significant evidence that removing copies from the *elite* is beneficial for the evolution (ANOVA results are given only as a reference). However, it must be noted that in Salustowicz case only one model out of the ten returned by *reference* was actually used for the comparison, so the corresponding statistical test is not reliable. Hock test problem appears as the only one where *reference* outperforms *with COPIES*: on the validation data set *reference* RMSE median is approximately half of *with COPIES* RMSE median, and the interquartile range of the first is an order of magnitude smaller than the other. Statistics seems not to support the better behaviour of *reference*, even though the

TABLE 5.14: Pathologies on building and validation data sets for *reference* and *with COPIES* implementations. All experiments consisted of 10 runs: %∞ is the percentage of individuals not defined, %bad is the percentage of individuals with a RMSE>100 on the particular data set. Median and IQR are computed on the remaining individuals.

Experiment	Training data				Test data			
			RMSE				RMSE	
	%∞	%bad	median	IQR	%∞	%bad	median	IQR
Column no.	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)
Kotanchek								
with COPIES	0	0	7.908E-02	1.840E-02	0	0	1.390E-01	4.453E-02
reference	0	0	6.676E-02	2.378E-02	10	0	1.109E-01	3.394E-02
Salustowicz								
with COPIES	0	0	1.977E-01	2.691E-02	20	0	2.263E-01	9.980E-02
reference	0	0	1.496E-01	1.023E-01	90	0	4.036E-01	0.000E+00
RatPol2D								
with COPIES	0	0	5.929E-01	1.856E-01	0	0	1.463E+00	9.902E-01
reference	0	0	6.165E-01	8.537E-02	0	10	1.690E+00	9.644E-01
Hock								
with COPIES	0	0	1.943E+00	1.143E+00	0	0	**7.583E+00**	**1.225E+01**
reference	0	0	8.412E-01	1.162E+00	0	0	**4.132E+00**	**4.059E+00**
Branin-Hoo								
with COPIES	0	0	6.647E+00	1.074E+01	10	0	1.959E+01	1.676E+01
reference	0	0	5.220E+00	2.368E+00	50	0	1.959E+01	1.357E+01

TABLE 5.15: *reference* vs. *with COPIES*: *p*-values returned by Wilcoxon rank sum test

NULL hypothesis: *reference* and *with COPIES* have the same distribution	p-values	
	ANOVA	Wilcoxon
Kotanchek	0.9053	0.2775
Salustowicz	0.1806	0.6667
RatPol2D	0.6072	0.5490
Hock	0.1100	0.0640
Branin-Hoo	0.5576	0.6064

p-value is really close to the 0.05 threshold. More definitive conclusions might probably be drawn from a larger number of runs.

5.3.5 Results for each test problem

In the following sections the performances of the remaining implementations are compared on a test problem at a time. Although the results reported in the previous section do not give statistical evidence that removing copies from the elite is beneficial, in all the experiments reported in the following copies were removed from the *elite*. This was done to improve semantical variety, in other words to discover the widest range possible of "best" individuals with acceptable accuracy but different mathematical structure. This can be useful to improve the chances to find expressions similar to the underlying function sought, which otherwise could be approximated with alternative expressions that are not interesting for the user (for example overfitted models, with structure completely different from the desired one - on the importance of retrieving a meaningful structure see discussion in Section 5.3.1.1). The HyGP input settings used to generate the best metamodel can be found in Appendix C.

5.3.5.1 Kotanchek test problem

The common GP parameters used for all the experiments are listed in Table 5.16.

TABLE 5.16: Kotanchek test problem: main parameters used in the experiments

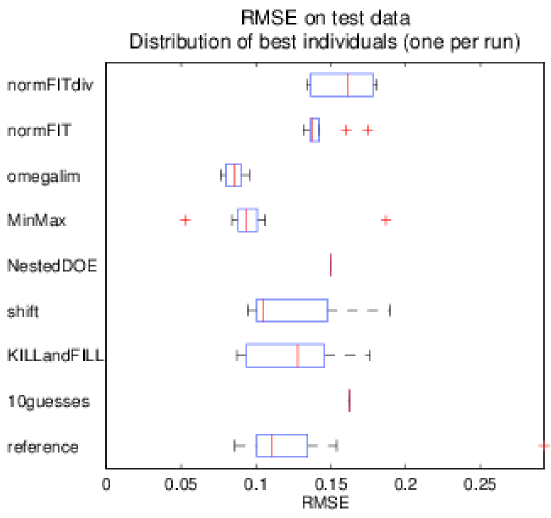
Population size:	200
Generations:	50
Primitives:	+, -, *, / (protected), square, cube, sin, cos, exp
No. of fitness cases:	40
NestedDOE: building points validating points	 28 12

The pathologies affecting the generated individuals on building and validation data sets are reported in Table 5.17. *NestedDOE* clearly stands out as 90% of the individuals present singularities on the validation data set. The high percentage may be due to either actual or not simplified singularities in the mathematical expressions. The difference is clearly explained if, for example, we consider the expression Z_1/Z_1: for $Z_1 = 0$ the fraction is undefined, but after simplification it returns a well defined result, 1. HyGP implementations are not able to discriminate between not simplified and actual singularities, so, adopting a conservative approach, *NestedDOE* results have been excluded from further statistical analysis as its RMSE sample is made by only one observation.

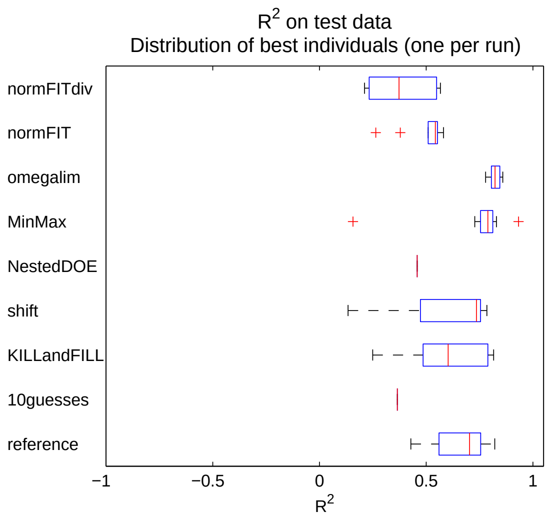
TABLE 5.17: Pathologies on building and validation data sets for Kotanchek test case. All experiments consisted of 10 runs: %∞ is the percentage of individuals not defined, %bad is the percentage of individuals with a RMSE>100 on the particular data set. Median and IQR are computed on the remaining individuals.

Kotanchek	Building data set		RMSE		Validation data set		RMSE	
	%∞	%bad	median	IQR	%∞	%bad	median	IQR
reference	0	0	6.676E-02	2.378E-02	10	0	1.109E-01	3.394E-02
10guesses	0	0	5.860E-02	0.000E+00	0	0	1.622E-01	0.000E+00
KILLandFILL	0	0	7.072E-02	1.697E-02	0	0	1.281E-01	5.244E-02
shift	0	0	6.229E-02	1.902E-02	10	0	1.047E-01	4.678E-02
NestedDOE	0	0	9.329E-02	1.999E-02	90	0	1.499E-01	0.000E+00
MinMax	0	0	6.852E-02	7.716E-03	0	0	9.343E-02	1.254E-02
Omegalim	0	0	6.155E-02	1.529E-02	0	0	8.571E-02	9.749E-03
normFIT	0	0	3.590E-01	4.122E-02	0	0	1.377E-01	6.452E-03
normFITdiv	0	0	4.100E-01	6.152E-02	0	0	1.611E-01	4.159E-02

The case of *10guesses* is also interesting, as the interquartile range is zero on both data sets. This might suggest that increasing the number of SQP random initial guesses improves the HyGP robustness, to a point that evolution always returns the same best individual regardless the stochastic nature of the algorithm: the search becomes independent from the initial population state. Unfortunately, the quality of the unique best individual returned is not particularly high. In Fig. 5.18 boxplots of the RMSE and R^2 samples computed on the validation (test) data set are shown.



(A) RMSE



(B) R^2

FIGURE 5.18: Kotanchek test case: RMSE and R^2 boxplots for validation data set. Individuals with RMSE>100 as well as undefined ones are not included

The global p-value returned by Kruskal Wallis test is 4.705E-07 so there is significant evidence at 5% level that some RMSE samples are definitely better than others in terms of median (one-Way Anova p-value, equal to 1.546E-06, seems to confirm it).

Table 5.18 shows the results of the pairwise comparisons between RMSE samples: p-values smaller than 0.05 indicate that the two implementations' RMSE medians as evaluated on the test data set are significantly different.

Considering that the high percentage of undefined individuals excludes *NestedDOE* from the comparisons (the corresponding row and column have to be neglected), *Omegalim* and *MinMax* implementations outperform all the others. Between the two there is no significant difference in the median according to the p-value (0.09). The best model

TABLE 5.18: Kotanchek test case: p-values resulting from pairwise comparison using Wilcoxon rank sum test. To compare two implementations, start from the row with the name of the first one; then read along the row until the column with the second implementation's name is found. If a white box is reached, keep reading down the column until the row with the second implementation's name is found.

Kotanchek	median	reference	10guesses	KILLandFILL	shift	NestedDoE	MinMax	OmegaLim	normFIT
reference	1.109E-01								
10guesses	1.622E-01	0.00							
KILLandFILL	1.281E-01	0.78	0.00						
shift	1.047E-01	1.00	0.00	1.00					
NestedDoE	1.499E-01	0.60	0.18	0.36	0.60				
MinMax	9.343E-02	0.03	0.00	0.06	0.03	0.36			
OmegaLim	8.571E-02	0.00	0.00	0.00	0.00	0.18	0.09		
normFIT	1.377E-01	0.03	0.00	0.27	0.16	0.55	0.00	0.00	
normFITdiv	1.611E-01	0.01	1.00	0.02	0.03	0.91	0.00	0.00	0.16

generated by *MinMax* is:

$$\begin{aligned}\tilde{f}(z_1, z_2) = {} & (0.0774878972863\, \mathrm{z}_1{}^3 - 0.508295723871\, \mathrm{z}_1{}^2 + 0.732193486674\, \mathrm{z}_1 \\ & - 0.0902514598889\, \mathrm{z}_2{}^2 + 0.436203142055\, \mathrm{z}_2)^2 - 0.000185135700008\, \mathrm{z}_1 \\ & - 0.0287136503092\end{aligned} \tag{5.24}$$

whereas the best model produced by *Omegalim* implementation[27] is:

$$\begin{aligned}\tilde{f}(z_1, z_2) = {} & 0.182099273973\ \sin(0.694646208945\, \mathrm{z}_1) \\ & + 0.174097117467\ \sin(1.82679481612\, \mathrm{z}_1) - 0.017891848784\, \mathrm{z}_1{}^2\, \mathrm{z}_2 \\ & + 0.186506234991\, \mathrm{z}_2{}^2 - 0.0463726896637\, \mathrm{z}_2{}^3 + 0.0000046354018192\, \mathrm{z}_1{}^2\, \mathrm{z}_2{}^7 \\ & - 0.0489317086592\end{aligned} \tag{5.25}$$

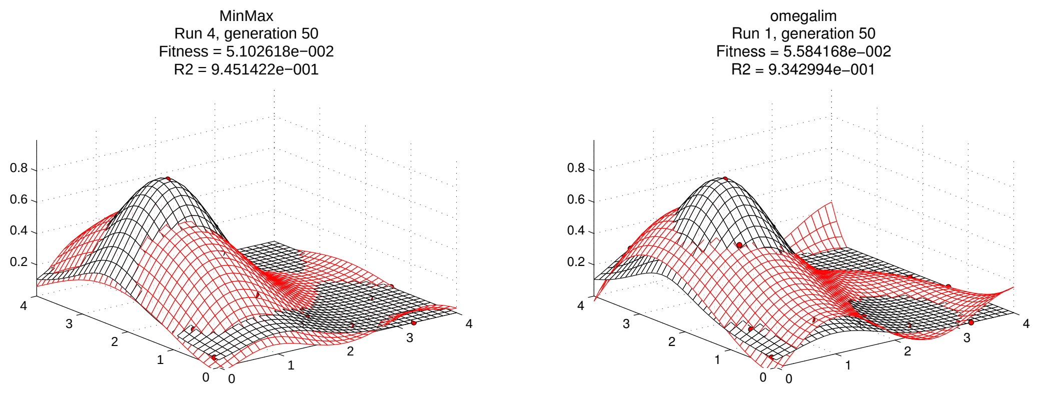
The corresponding plots are shown in Fig. 5.19 superimposed to the original Kotanchek function.

5.3.5.2 Salustowicz test problem

The main GP parameters used for all the experiments are shown in Table 5.19.

The pathologies affecting the individuals generated by the different GP implementations are shown in Table 5.20. From the high percentages of undefined individuals on the

[27]the input settings used are reported in Section C.1, Appendix C.



(A) *MinMax* (B) *Omegalim*

FIGURE 5.19: Kotanchek test problem: best models generated by *MinMax* and *Omegalim* implementations (in red) superimposed to Kotanchek function (in black)

TABLE 5.19: Salustowicz test problem: main parameters used in the experiments

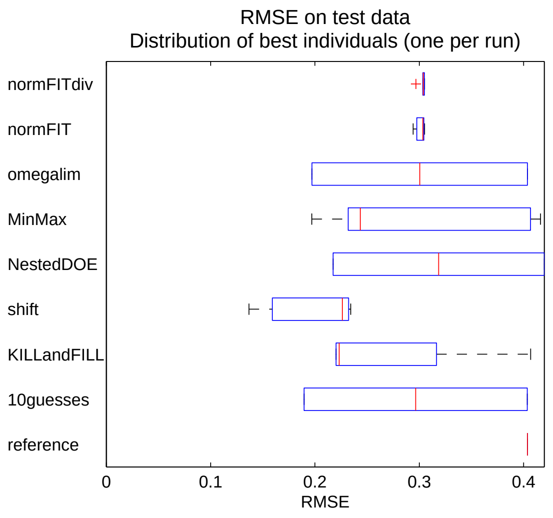
Population size:	300
Generations:	50
Primitives:	+, -, *, / (protected), square, cube, sin, cos, exp, negative exp, opposite, reciprocal
Fitness cases:	100
NestedDOE: building points validating points	 66 34

validation data set (column "%∞" under "Validation data set"), it is clear that all the implementations struggled to produce individuals with good generalisation ability, although the number of primitives, the population size (300), and the number of fitness cases were increased with respect to Kotanchek test case. Despite the poor general performances, *normFIT* and *normFITdiv* seem to be the most robust techniques.

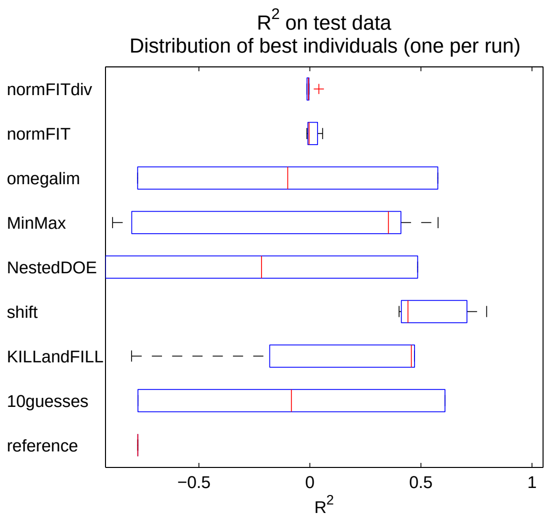
TABLE 5.20: Salustowicz test problem: pathologies on building and validation data sets

Salustowicz	Building data set				Validation data set			
			RMSE				RMSE	
	%∞	%bad	median	IQR	%∞	%bad	median	IQR
reference	0	0	1.496E-01	1.023E-01	90	0	4.036E-01	0.000E+00
10guesses	0	0	5.748E-02	6.042E-02	80	0	2.966E-01	2.139E-01
KILLandFILL	0	0	1.945E-01	3.011E-02	50	10	2.233E-01	9.634E-02
shift	0	0	1.200E-01	1.199E-01	60	10	2.263E-01	7.309E-02
NestedDOE	0	0	1.390E-01	1.084E-01	80	0	3.186E-01	2.025E-01
MinMax	0	0	1.794E-01	3.262E-02	50	0	2.435E-01	1.749E-01
Omegalim	0	0	1.702E-01	7.376E-02	80	0	3.004E-01	2.065E-01
normFIT	0	0	8.536E-01	1.157E-01	40	0	3.035E-01	6.717E-03
normFITdiv	0	0	8.474E-01	9.268E-02	30	0	3.039E-01	1.397E-03

Boxplots of the RMSE and R^2 samples are shown in Fig. 5.20: *shift* produced the best individual in terms of RMSE and R^2. However, such conclusion is not statistically significant as Kruskal-Wallis *p*-value for RMSE samples as evaluated on the validation data set is ten times larger than the 0.05 threshold (0.5675). In any case, the low percentage of individuals defined on the validation data set made the tests unreliable.



(A) RMSE



(B) R^2

FIGURE 5.20: Salustowicz: RMSE and R^2 boxplots. Individuals with RMSE>100 as well as undefined ones are not included

The best metamodel returned by *shift*[28] is plotted in red in Fig. 5.21 overlapped to Salustowicz function in black. The metamodel expression is reported below:

$$\begin{aligned}\tilde{f}(\mathrm{z}_1) = 0.5569171313814 + ((0.6030669446878*\\ (\cos[((((66.38679520928 * (-82.78990399331 + 1.093325012629\,\mathrm{z}_1))))/\\ (((-21.87042454172 * (25.03295588305 + \mathrm{z}_1\,\mathrm{z}_1\,\mathrm{z}_1)))))]))\\ - (-5.774999946019\,\mathrm{z}_1 + 5.905808685662\,\mathrm{z}_1))\end{aligned} \tag{5.26}$$

5.3.5.3 RatPol2D test problem

In Table 5.21 the main GP parameters are listed.

The pathologies of the individuals generated by the HyGP implementations on building and validation data sets are reported in Table 5.22.

[28]the input settings used to generate the metamodel can be found in Section C.2, Appendix C.

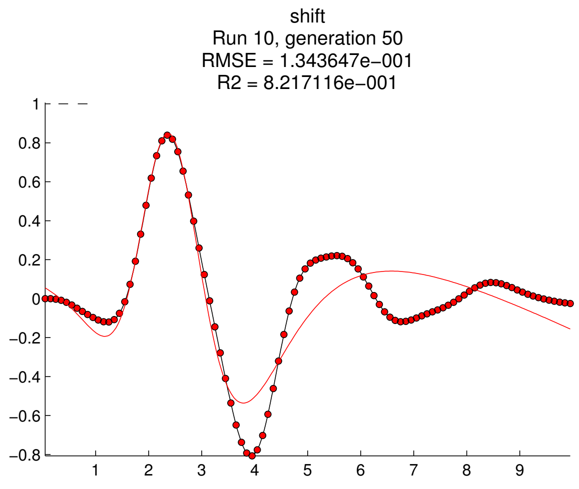


FIGURE 5.21: Salustowicz test problem: best model generated by *shift* implementation (red line) and Salustowicz function (black line)

TABLE 5.21: RatPol2D test problem: main parameters used in the experiments

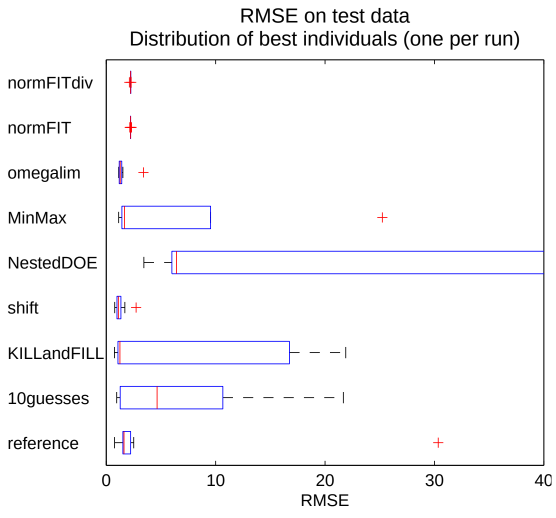
Population size:	200
Generations:	50
Primitives:	+, -, *, / (protected), square, cube, sin, cos, exp, reciprocal
Fitness cases:	40
NestedDOE: building points validating points	 28 12

TABLE 5.22: RatPol2D test problem: pathologies on building and validation data sets

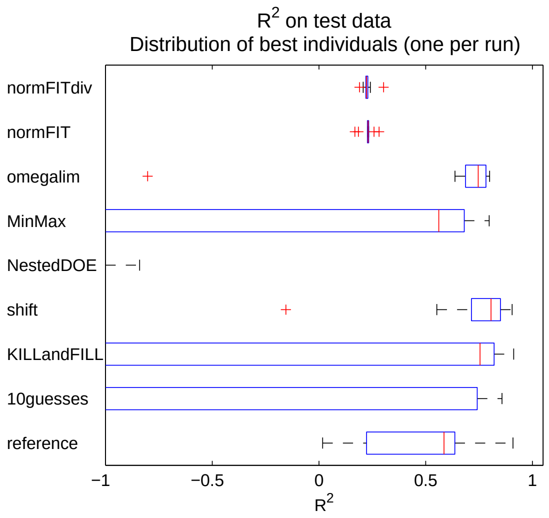
RatPol2D	Building data set				Validation data set			
			RMSE				RMSE	
	%∞	%bad	median	IQR	%∞	%bad	median	IQR
reference	0	0	6.165E-01	8.537E-02	0	10	1.634E+00	7.017E-01
10guesses	0	0	4.923E-01	2.069E-01	0	10	4.653E+00	9.383E+00
KILLandFILL	0	0	5.439E-01	1.336E-01	0	30	1.258E+00	1.567E+01
shift	0	0	4.279E-01	3.367E-01	0	10	1.117E+00	3.586E-01
NestedDOE	0	0	1.316E+00	1.615E-01	0	0	6.433E+00	3.484E+01
MinMax	0	0	4.845E-01	2.440E-01	0	0	1.681E+00	8.114E+00
Omegalim	0	0	4.864E-01	2.994E-01	0	10	1.279E+00	2.357E-01
normFIT	0	0	5.021E-01	3.152E-02	0	0	2.228E+00	8.648E-03
normFITdiv	0	0	5.061E-01	3.860E-02	0	0	2.238E+00	1.321E-02

Differently from Salustowicz test problem, all the generated metamodels are defined on the validation data set, although in a few cases a small percentage of them score a high RMSE (column "%bad" under "Validation data set"). From the boxplots of the RMSE and R^2 samples shown in Fig. 5.22, *reference*, *shift* and *Omegalim* implementations stand out as giving the best combination of low RMSE median and interquartile range (IQR).

Significant evidence of a difference among the RMSE medians is confirmed by the low



(A) RMSE



(B) R^2

FIGURE 5.22: RatPol2D test problem: RMSE and R^2 boxplots. Individuals with RMSE>100 as well as undefined ones are not included

p-value returned by Kruskal-Wallis test[29] (5.969E-05<0.05). Pairwise comparisons made by Wilcoxon rank sum test highlight the better performances of *shift* and *Omegalim* with respect to *NestedDOE*, *normFIT* and *normFITdiv*. There is no significant difference at 5% level between the RMSE medians of *shift* and *Omegalim*. The list of *p*-values produced by Wilcoxon rank sum test is presented in Table 5.23.

TABLE 5.23: RatPol2D test case: *p*-values resulting from pairwise comparison using Wilcoxon rank sum test. To compare two implementations, start from the row with the name of the first one; then read along the row until the column with the second implementation's name is found. If a white box is reached, keep reading down the column until the row with the second implementation's name is found

RatPol2D	median	reference	10guesses	KILLandFILL	shift	NestedDoE	MinMax	OmegaLim	normFIT
reference	1.634E+00								
10guesses	4.653E+00	0.34							
KILLandFILL	1.258E+00	0.30	0.47						
shift	1.117E+00	0.08	0.03	0.47					
NestedDoE	6.433E+00	0.00	0.18	0.06	0.00				
MinMax	1.681E+00	0.84	0.97	0.19	0.01	0.05			
OmegaLim	1.279E+00	0.04	0.11	0.61	0.06	0.00	0.03		
normFIT	2.228E+00	0.04	0.24	0.16	0.00	0.00	0.47	0.00	
normFITdiv	2.238E+00	0.05	0.24	0.16	0.00	0.00	0.47	0.00	0.31

[29]The *p*-value returned by One-Way Anova test, 1.615E-02, backs the conclusion.

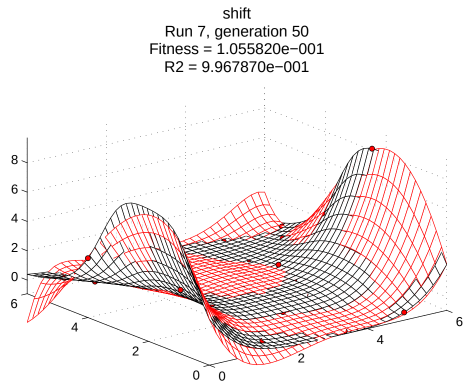
The expression of the best model generated by *shift* is reported below[30]:

$$\begin{aligned}\tilde{f}(z_1, z_2) &= -0.00173828969516\, z_1{}^6\, z_2 + 0.0334579253585\, z_1{}^5\, z_2 \\ &+ 0.0000351699480953\, z_1{}^4\, z_2{}^4 - 0.00987633264153\, z_1{}^4\, z_2{}^2 \\ &- 0.213450848609\, z_1{}^4\, z_2 + 0.0698782011422\, z_1{}^4 + 0.0942687219247\, z_1{}^3\, z_2{}^2 \\ &+ 0.546678587066\, z_1{}^3\, z_2 - 0.886656122808\, z_1{}^3 - 0.426718761705\, z_1{}^2\, z_2{}^2 \\ &+ 3.93178636296\, z_1{}^2 - 0.0674199876581\, z_1\, z_2{}^3 + 1.46482757145\, z_1\, z_2{}^2 \\ &- 3.48056045116\, z_1\, z_2 - 6.75940150248\, z_1 + 0.173809936525\, z_2{}^3 \\ &- 2.28987637434\, z_2{}^2 + 6.6340120764\, z_2 + 2.19324217003\end{aligned} \tag{5.27}$$

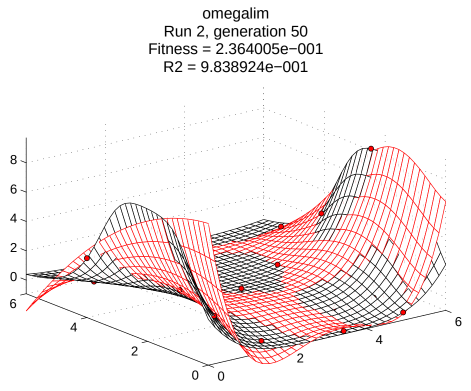
The best model produced by *Omegalim* is given in Eq. 5.28:

$$\begin{aligned}\tilde{f}(z_1, z_2) &= 0.000411031272381\, z_1{}^4\, z_2{}^3 - 0.045187441492\, z_1{}^4\, z_2 + 0.182445578214\, z_1{}^4 \\ &- 0.0501672989779\, z_1{}^3\, z_2{}^2 + 0.690345540243\, z_1{}^3\, z_2 - 2.28965459769\, z_1{}^3 \\ &- 0.0205614550616\, z_1{}^2\, z_2{}^3 + 0.36685951209\, z_1{}^2\, z_2{}^2 - 3.2366096702\, z_1{}^2\, z_2 \\ &+ 10.0265022305\, z_1{}^2 - 0.00360820550814\, z_1\, z_2{}^5 + 0.0544395450417\, z_1\, z_2{}^4 \\ &- 0.231088472394\, z_1\, z_2{}^3 + 4.5212342401\, z_1\, z_2 - 17.5162957202\, z_1 \\ &- 0.2690291428\, z_2{}^2 - 0.194045031546\, z_2 + 8.74325580434\end{aligned} \tag{5.28}$$

The metamodels are plotted in Fig. 5.23 superimposed to RatPol2D function.



(A) *shift*



(B) *Omegalim*

FIGURE 5.23: RatPol2D test case: best individuals from *shift* and *Omegalim* implementations (in red) superimposed to RatPol2D function (in black)

[30]the input settings used to produce the metamodel are listed in Section C.3, Appendix C

5.3.5.4 Hock test problem

The main GP parameters are shown in Table 5.24.

TABLE 5.24: Hock test problem: main parameters used in the experiments

Population size:	200
Generations:	50
Primitives:	+, -, *, / (protected), square, cube, sin, cos, exponential
Fitness cases:	20
NestedDOE: building points validating points	 14 6

The pathologies affecting the generated metamodels are described in Table 5.25.

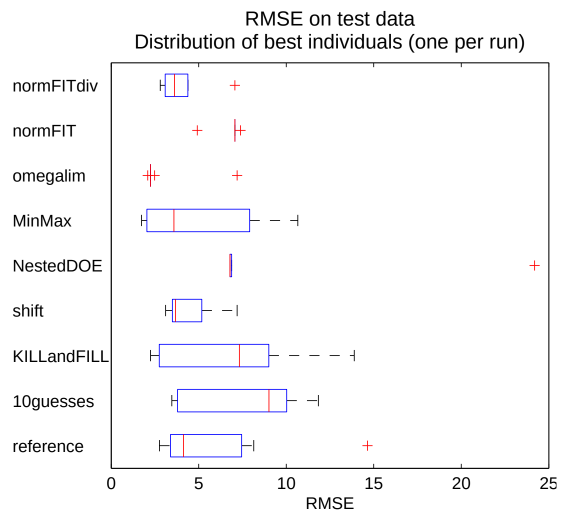
TABLE 5.25: Hock test problem: pathologies on building and validation data sets

Hock	Building data set				Validation data set			
			RMSE				RMSE	
	%∞	%bad	median	IQR	%∞	%bad	median	IQR
reference	0	0	8.412E-01	1.162E+00	0	0	4.132E+00	4.059E+00
10guesses	0	0	7.465E-01	2.979E-01	0	10	9.016E+00	6.235E+00
KILLandFILL	0	0	8.587E-01	1.161E+00	0	0	7.325E+00	6.256E+00
shift	0	0	1.086E+00	1.172E+00	0	0	3.672E+00	1.681E+00
NestedDOE	0	0	2.865E+00	2.190E-04	0	0	6.788E+00	9.337E-02
MinMax	0	0	6.805E-01	2.389E-01	0	30	3.581E+00	5.877E+00
Omegalim	0	0	8.587E-01	0.000E+00	0	0	2.252E+00	9.412E-06
normFIT	0	0	1.445E-02	0.000E+00	0	10	7.062E+00	7.942E-08
normFITdiv	0	0	2.539E-03	1.476E-03	0	10	3.612E+00	1.295E+00

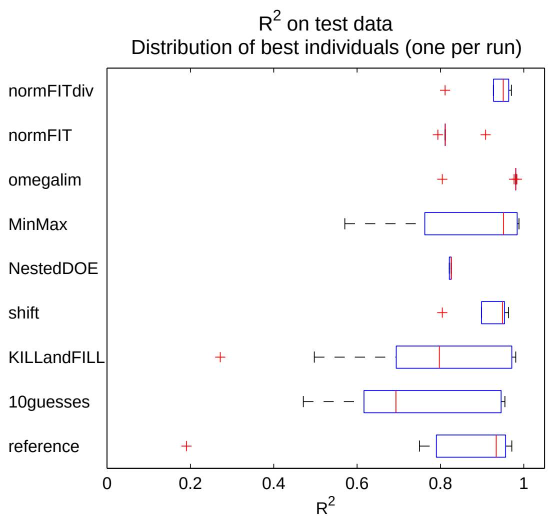
No particularly bad behaviours emerged during the RMSE and R^2 evaluation on validation data set. A small percentage of individuals generated by *MinMax* shows a bad behaviour on validation data set. In the boxplots shown in Fig. 5.24 *Omegalim* stands out for the low RMSE median and the extremely low RMSE and R^2 interquartile range (IQR).

The low p-value returned by Kruskal-Wallis test (4.577E-04) provides significant evidence, at the 5% level, of a difference in the RMSE medians among the samples[31]. Pairwise Wilcoxon rank sum tests confirm the better performance of *Omegalim*: all the p-values resulting from a comparison involving *Omegalim*, except for the one with *MinMax*, are smaller than 0.05 (see Table 5.26).

[31]ANOVA test however does *not* support the conclusion, as the p-value returned is 0.33.



(A) RMSE



(B) R^2

FIGURE 5.24: Hock test problem: RMSE and R^2 boxplots. Individuals with RMSE>100 as well as undefined ones are not included

TABLE 5.26: Hock test case: p-values resulting from pairwise comparison using Wilcoxon rank sum test. To compare two implementations, start from the row with the name of the first one; then read along the row until the column with the second implementation's name is found. If a white box is reached, keep reading down the column until the row with the second implementation's name is found

Hock	median	reference	10guesses	KILLandFILL	shift	NestedDoE	MinMax	OmegaLim	normFIT
reference	4.132E+00								
10guesses	9.016E+00	0.08							
KILLandFILL	7.325E+00	0.62	0.32						
shift	3.672E+00	0.85	0.08	0.47					
NestedDoE	6.788E+00	0.31	0.48	0.79	0.02				
MinMax	3.581E+00	0.60	0.07	0.23	0.67	0.46			
OmegaLim	2.252E+00	0.00	0.00	0.00	0.00	0.00	0.31		
normFIT	7.062E+00	0.45	0.26	0.55	0.03	0.03	0.21	0.00	
normFITdiv	3.612E+00	0.32	0.01	0.18	0.32	0.00	0.92	0.00	0.00

The best metamodelgenerated by *Omegalim*[32] is reported in Eq. (5.29):

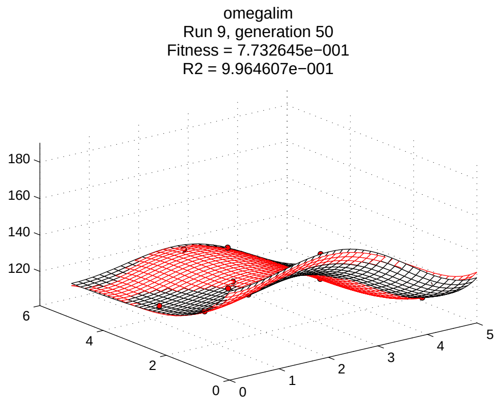
$$\tilde{f}(z_1, z_2) = \left(3.59644475784\,\mathrm{z}_1 - 0.770501844171\,{\mathrm{z}_1}^2\right)^2 - 24.074516819\,\mathrm{z}_2 \\ - 4.30155851714\,\mathrm{z}_1 + 7.02267630102\,{\mathrm{z}_2}^2 - 0.681383038881\,{\mathrm{z}_2}^3 \\ + 148.962136216 \tag{5.29}$$

[32]the input settings used can be found in Section C.4, Appendix C.

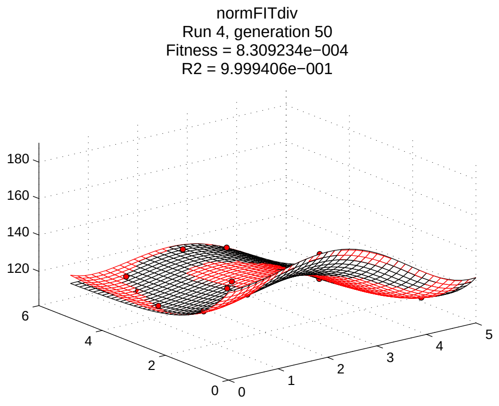
The best metamodel generated by *normFITdiv* is shown in Eq. (5.30):

$$
\begin{aligned}
\tilde{f}(z_1, z_2) = &-0.00140908481001\,\mathrm{z_1}^4\,\mathrm{z_2}^2 + 0.6294892722\,\mathrm{z_1}^4 - 5.85582776761\,\mathrm{z_1}^3 \\
&+ 0.465151867005\,\mathrm{z_1}^2\,\mathrm{z_2} + 13.2924383738\,\mathrm{z_1}^2 - 1.5541368693\,\mathrm{z_1}\,\mathrm{z_2} \\
&- 3.3565923571\,\mathrm{z_1} + 0.131421354898\,\mathrm{z_2}^4 - 1.89803058221\,\mathrm{z_2}^3 \\
&+ 10.6252778175\,\mathrm{z_2}^2 - 26.989797409\,\mathrm{z_2} + 149.19569559
\end{aligned}
\tag{5.30}
$$

The metamodels presented above are plotted in Fig. 5.25 superimposed to the Hock function.



(A) *Omegalim*



(B) *normFITdiv*

FIGURE 5.25: Hock test problem: best individuals from *Omegalim* and *normFITdiv* implementations (in red) superimposed to Hock function (in black)

5.3.5.5 Branin-Hoo test problem

The main GP parameters used for all experiments are listed in Table 5.27.

TABLE 5.27: Branin-Hoo test problem: main parameters used in the experiments

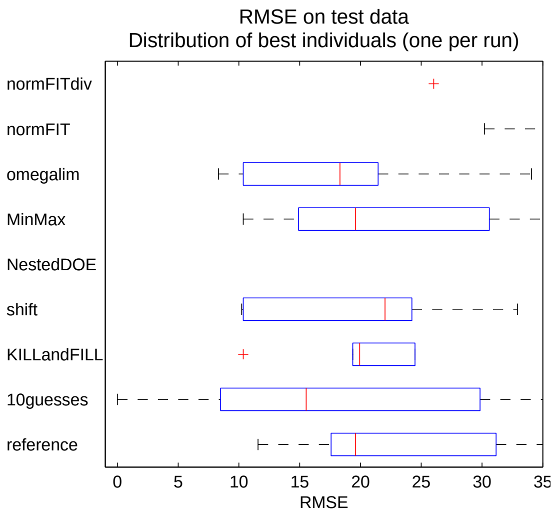
Population size:	200
Generations:	50
Primitives:	+, -, *, / (protected), square, cube, sin, cos, exp
Fitness cases:	30
NestedDOE: building points validating points	 20 10

The pathologies of the generated individuals on building and validation data sets are reported in Table 5.28.

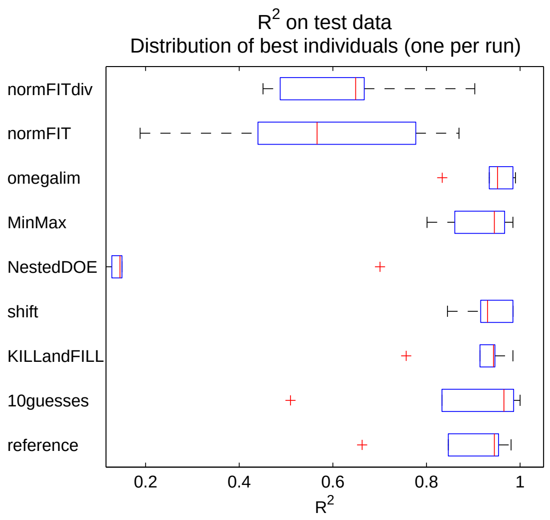
TABLE 5.28: Branin-Hoo test problem: pathologies on building and validation data sets

Branin-Hoo	Building data set				Validation data set			
			RMSE				RMSE	
	%∞	%bad	median	IQR	%∞	%bad	median	IQR
reference	0	0	5.220E+00	2.368E+00	50	0	1.959E+01	1.357E+01
10guesses	0	0	5.144E+00	3.540E+00	50	0	1.553E+01	2.136E+01
KILLandFILL	0	0	5.223E+00	2.132E+00	40	0	1.994E+01	5.115E+00
shift	0	0	5.973E+00	2.730E+00	30	10	2.202E+01	1.388E+01
NestedDOE	0	0	4.130E+01	7.277E-02	20	0	7.724E+01	9.798E-01
MinMax	0	0	6.113E+00	1.508E+00	20	0	1.959E+01	1.570E+01
Omegalim	0	0	5.944E+00	9.271E-01	40	0	1.832E+01	1.110E+01
normFIT	0	0	1.979E-01	5.636E-02	40	10	5.502E+01	2.295E+01
normFITdiv	0	0	1.386E-01	9.913E-02	30	0	4.951E+01	1.160E+01

All the implementations struggled to generate metamodels with good generalisation ability, as shown by the high percentage of undefined individuals on the validation data set. All RMSE and R^2 samples are characterised by high interquartile ranges as shown by the boxplots in Fig. 5.26. The poor performances of *NormFIT*, *normFITdiv* and *NestedDoe* implementations clearly emerge. The RMSE samples produced by the other implementations are not substantially different.



(A) RMSE



(B) R^2

FIGURE 5.26: Branin-Hoo test problem: RMSE and R^2 boxplots. Individuals with RMSE>100 as well as undefined ones are not included

Kruskal-Wallis test results support the hypothesis that not all RMSE medians are equal (*p*- value: 2.310E-05), at least at 5% level. Such conclusion may however not be reliable given the high percentage of undefined individuals[33].

[33]The evidence of a difference among the RMSE medians is backed by Anova *p*-value (2.848E-11), but this value too could be biased by the reduced size of the samples.

A few conclusions can be drawn from pairwise comparison of the RMSE samples by Wilcoxon rank sum test, whose p-values are reported in Table 5.29. *NestedDOE* is outperformed by all the other implementations, as anticipated by boxplots. *NormFIT* and *normFITdiv* are also characterised by poor performances. There is no significant evidence of differences in RMSE medians for the remaining implementations.

TABLE 5.29: Branin-Hoo test case: p-values resulting from pairwise comparison using Wilcoxon rank sum test. To compare two implementations, start from the row with the name of the first one; then read along the row until the column with the second implementation's name is found. If a white box is reached, keep reading down the column until the row with the second implementation's name is found

Branin-Hoo	median	reference	10guesses	KILLandFILL	shift	NestedDoE	MinMax	OmegaLim	normFIT
reference	1.959E+01								
10guesses	1.553E+01	0.55							
KILLandFILL	1.994E+01	0.93	0.54						
shift	2.202E+01	0.79	0.66	0.94					
NestedDoE	7.724E+01	0.00	0.00	0.00	0.00				
MinMax	1.959E+01	0.94	0.62	0.75	0.75	0.00			
OmegaLim	1.832E+01	0.43	0.93	0.48	0.59	0.00	0.41		
normFIT	5.502E+01	0.03	0.10	0.01	0.01	0.01	0.01	0.01	
normFITdiv	4.951E+01	0.01	0.03	0.00	0.00	0.01	0.00	0.00	1.00

It is worth noting that the best individual generated by *10guesses*[34], whose expression is shown in Eq. (5.31), is as a matter of fact the Branin-Hoo function:

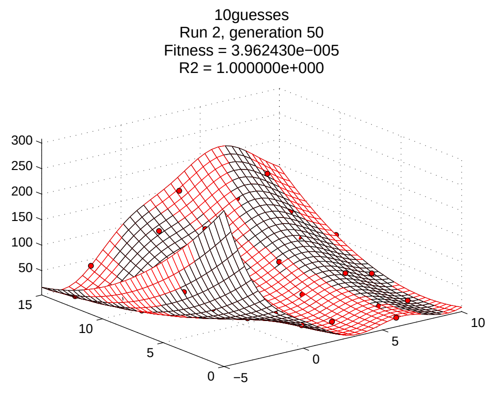
$$\begin{aligned}\tilde{f}(z_1, z_2) &= 9.60209703926\ \cos(1.00000053842\,\mathrm{z}_1) - 12.0000255964\,\mathrm{z}_2\\ &- 19.0985885529\,\mathrm{z}_1 + 3.1830982295\,\mathrm{z}_1\,\mathrm{z}_2 - 0.258368814492\,{\mathrm{z}_1}^2\,\mathrm{z}_2\\ &+ 4.08324072854\,{\mathrm{z}_1}^2 - 0.411206971382\,{\mathrm{z}_1}^3 + 0.0166886394138\,{\mathrm{z}_1}^4\\ &+ 1.00000121669\,{\mathrm{z}_2}^2 + 46.0001253703\end{aligned} \tag{5.31}$$

The best individual generated by *Omegalim* implementation is instead a polynomial:

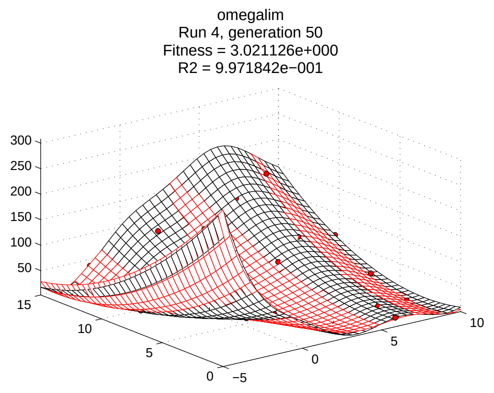
$$\begin{aligned}\tilde{f}(z_1, z_2) &= 0.00110046560295\,{\mathrm{z}_1}^6 - 0.0195603884183\,{\mathrm{z}_1}^5 + 0.0651324905047\,{\mathrm{z}_1}^4\\ &+ 0.163610467325\,{\mathrm{z}_1}^3 - 0.245036641918\,{\mathrm{z}_1}^2\,\mathrm{z}_2 + 2.1005820657\,{\mathrm{z}_1}^2\\ &+ 3.08536116255\,\mathrm{z}_1\,\mathrm{z}_2 - 21.4397538069\,\mathrm{z}_1 + 0.958637398866\,{\mathrm{z}_2}^2\\ &- 11.5720729266\,\mathrm{z}_2 + 51.4956500855\end{aligned} \tag{5.32}$$

[34]the input settings used can be found in Section C.5, Appendix C.

The plots of the metamodels reported in Eqs. (5.31-5.32) are shown in Fig. 5.27, superimposed to Branin-Hoo function.



(A) *10guesses*



(B) *Omegalim*

FIGURE 5.27: Branin-Hoo test problem: best individuals from *10guesses* and *Omegalim* implementations (in red) superimposed to Branin-Hoo function (in black)

5.3.6 Discussion of the results

Two different kinds of conclusions can be drawn from the results described in the previous sections. First of all, all HyGP implementations appear to struggle on rational functions like Kotanchek (Eq. (5.19)) and RatPol2D (Eq. (5.21)) functions. The poor performance in this cases can be explained by the fact that the division, although protected, is an operator likely to produce overfitted or udefined individuals. Division by a term that is zero within the design space is the simplest way to introduce high non-linearities in a model without increasing size, and so to increase fitness according to the definition given in Section 5.2.5. If the poles of the division do not belong to the building data set, then the model has more chances to survive than the others as a result of overfitting. However, during validation the drawbacks of overfitting eventually emerge, resulting in undefined individuals. If on the other hand the poles belong to the building data set, the penalisation of illegal operation (see objective F_3 in fitness function definition - Section 5.2.5) reduces the likelihood that the model survives to selection and be able to spread the division operator. In both cases this means that division will rarely appear in an acceptable model. A proof of this interpretation is the fact that none of the best expressions generated for Kotanchek and RatPol2D test cases contains a division, being in particular a linear combination of terms. Two possible solutions to better exploit operators like division will be shown in the next section (5.4) and in Chapter 7.

Statistical tests however confirm that for each single test case some implementations perform definitely better than others. *NestedDOE* (described in Section 5.3.0.5), *normFIT* and *normFITdiv* (in Section 5.3.0.7) have generally shown poor performances in most of the problems (see Kotanchek, RatPol2D, Branin-Hoo test functions). The negative performance of *NestedDOE* is not expected, as being the implementation of the cross-validation strategy (*hold out method* - see Section 1.2.2, Chapter 1) it should generate models with the best compromise between accuracy and generalisation: results show however that its curbing effect on model size is too conservative (high bias, low variance). *Omegalim* (described in Section 5.3.0.6) appears to be the best implementation on the tested symbolic regression problems, both for quality (low RMSE median) and reliability (low RMSE interquartile range) of the metamodels generated. *Shift* implementation (described in Section 5.3.0.4) has also produced high-quality models for Salustowicz and RatPol2D test cases, proving that the increased redundancy resulting from using the unary operator “shift” can improve the search for a good metamodel.

On the other end, unexpected behaviours have emerged. The *KILLandFILL* (see Section 5.3.0.2) implementation has not produced consistent results, maybe as a result of a lack of protection in tournament selection of newly generated individuals (“young” individuals) against fitter ones (“older” individuals). As a matter of fact, the ALPS algorithm which inspired *KILLandFILL* implementation features a selection strategy that prevents individuals of different “age” from being competitors in the same tournament selection, age being defined as the number of generations an individual survives (Hornby 2006). In the experiments previously described such protection based on age was not introduced as the original purpose was to assess the effect of the *cascade* operator, which does not feature age-based protection (see Section 3.1.5.6, Chapter 3). The mediocre performance of *10guesses* is also surprising: despite the increased tuning effort, the search appears not to be consistenly improved.

In conclusion, *Omegalim* strategy and *shift* unary operator can be considered as the most effective approaches to improve HyGP performances among the ones tested.

5.4 Problem specific knowledge exploitation in HyGP

The ability to build metamodels only from data, regardless the nature of the system that produced the data, is the main strength of data-driven metamodelling techniques (Affenzeller and Wagner 2004, Vladislavleva 2008). Any GP implementation can operate as a "black-box", leaving to the user only the setting of the parameters required to launch an experiment and the final assessment of the quality of the returned metamodel.

Yet, it is plausible to think that the quality of the evolved metamodels can benefit from any information regarding the system under analysis, deriving for example from establihsed analytical or fundamental models (see Section 1.1, Chapter 1) or from prior knowledge of the system. Domain knowledge has been indeed used in GP to bias the evolutionary search in design applications, as noted by Barbosa and Bernardino (2011). In some cases, such knowledge has been used to actively control GP evolution. Dimensionally aware genetic programming (Keijzer and Babovic 1999), introduced in Section 4.3.1, Chapter 4, is an example of how the physical consistency of the operations in a syntax tree can be used as a criterion to direct the evolution.

In this section a strategy to exploit constraints that may be implicitly posed by the physical nature of the system under study is presented. The approach makes use of the lower and upper bounds that in many cases may be assumed on the system responses to consider them feasible, for mathematical or physical reasons. For instance, in mechanics kinetic energy is by definition constrained by a lower bound equal to zero, being always positive. Another example are scaled or normalised quantities: in thermodynamics efficiency is generally defined in the range $[0,\ 1]$. As such bounds are generally inferred from existing analytical models and do not trequire additional simulations or experiments, they represent knowledge that can be directly used to direct GP evolution at a reduced, if not null, computational cost.

If the existence of a feasible range for the response is ascertained, it is then possible to penalise at the evaluation stage the GP individuals that return unfeasible values. An additional data set provided by the user can be used for the feasibility check. Such approach was implemented in HyGP by adding an extra term to the fitness function defined in Section 5.2.5. The reformulated fitness function is shown in Eq. (5.33):

$$F\,(i,t) = a_1 F_1(i,t) + a_2 F_2(i,t) + a_3\, 10^6\, F_3(i,t) + a_4 F_4(i,t) + a_5 F_5(i,t) \tag{5.33}$$

$$a_1 + a_2 + a_3 + a_4 + a_5 = 1 \tag{5.34}$$

The definition of the additional term F_5 had to take into account the response returned by the metamodel and the maximum or minimum value the actual response can assume in each point of the additional data set C used for the feasibility check. To this purpose, for the i-th metamodel $\tilde{f}$ generated at generation t a parameter $d_{i,t}$ was defined on C as:

$$d_{i,t} = \sum_{k=1}^{|C|} d_k \tag{5.35}$$

$$d_k = \begin{cases} \left|\tilde{f}(\mathbf{x}_k) - u_k\right| & \text{if } \tilde{f}(\mathbf{x}_k) \geq u_k \\ \left|\tilde{f}(\mathbf{x}_k) - l_k\right| & \text{if } \tilde{f}(\mathbf{x}_k) \leq l_k \\ 0 & \text{if } \tilde{f}(\mathbf{x}_k) > l_k \ \wedge \ \tilde{f}(\mathbf{x}_k) < u_k \end{cases} \tag{5.36}$$

where $|C|$ is the number of points the data set C is made of, $\mathbf{x}_k$ is the vector defining the position of the k-th point belonging to C, u_k and l_k are respectively the upper and lower bound of the response in point k. It should be noted that u_k and l_k are not necessarily both defined: for example positive responses are defined in the seminterval $[0 \ \infty)$, so u_k is undefined or infinite. Eqs. (5.35, 5.36) formally express the concept that $d_{i,t}$ is the sum of the distances of the metamodel responses from a set of points belonging to the boundary of the response feasible region. If all the estimated responses on C are in the feasible region then $d_{i,t}$ is zero.

Preliminary tests showed that defining $F_5(i,t)$ (Eq. (5.33)) using a linear dependency on $d_{i,t}$ (Eq. (5.35)) resulted in no quality improvement, being the penalisation too weak. An exponential dependency on $d_{i,t}$ was finally adopted:

$$F_5(i,t) = 10^6 \left(e^{d_{i,t}^p} - 1\right) \qquad p \in \mathbb{N} \tag{5.37}$$

from Eq. (5.37) is easy to see that the penalisation due to unfeasible response smoothly reduces to zero with $d_{i,t}$ approaching zero. The coefficient p can be tuned to tighten or loosen the penalisation. In the next section the benefits of the presented approach are described using a benchmark problem.

5.4.1 Test of the penalisation approach

The effect of the penalisation term F_5 defined in Eq. (5.37) was tested on the Kotanchek function, defined in Eq. (5.19) and reported below:

$$f(z_1, z_2) = \frac{e^{-(z_1-1)^2}}{1.2 + (z_2 - 2.5)^2} \tag{5.38}$$

Kotnachek function is strictly positive on $\mathbb{R}^2$: the experiments with the penalisation aimed at testing whether this knowledge can be used effectively through the proposed strategy to improve HyGP symbolic regression.

Experiments made of 10 independent HyGP evolutions were run using different values of the exponent p (defined in Eq. (5.37)) and the coefficient a_5 (defined in Eq. (5.33)), without altering the general settings of the HyGP runs (200 individuals, 50 generations). The implementation used in all experiments was the combined *Omegalim* and *shift* approach, following the conclusions expressed in Section 5.3.6. The primitives used in all evolutions were addition, subtraction, multiplication, protected division, shift, square, cube, sine, cosine and exponential.

The sign of the response produced by HyGP metamodels was checked on an additional data set C made of 20 uniformly spaced points (full factorial DoE) covering the region $[0, 4] \times [0, 4]$. The same building and validation data set defined in Table 5.2 (Section 5.3.1.2) were used to generate and to validate the quality of the best metamodels produced by each HyGP run. The statistical methods described in Section 5.3.1 were used to compare the RMSE and R^2 samples produced by each HyGP experiment.

Eight different HyGP experiments were performed. In four of them penalisation was enabled, setting the exponent p to $p = 2$ and $p = 3$, studying also the effect of different values of the coefficient a_5 (Eq. (5.33)) on the quality of the final best metamodel. In the other four experiments the penalisation was not used, in order to make a comparison with the *reference*, *shift*, *Omegalim* and the combined *Omegalim* and *shift* implementations (described in Section 5.2, Section 5.3.0.4 and Section 5.3.0.6). The experiments included in the analysis will be named as in Table 5.30.

TABLE 5.30: Plan of experiments to assess the effect of the penalisation term F_5

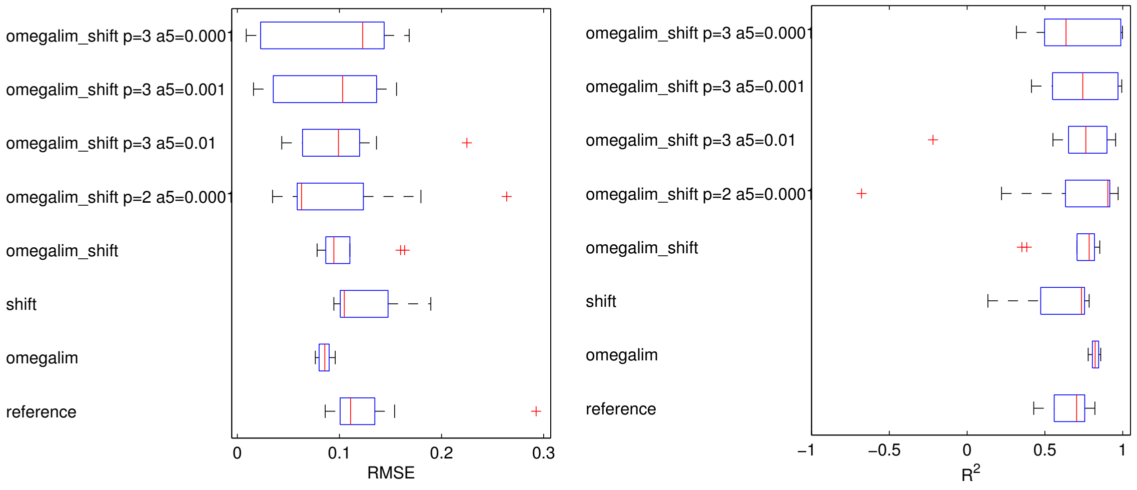
No.	experiment name	description	p	a_5
1	omegalim_shift p=3 a5=0.0001	penalisation enabled	3	0.0001
2	omegalim_shift p=3 a5=0.001	penalisation enabled	3	0.001
3	omegalim_shift p=3 a5=0.01	penalisation enabled	3	0.01
4	omegalim_shift p=2 a5=0.0001	penalisation enabled	2	0.0001
5	omegalim_shift	penalisation DISABLED	$\times$	$\times$
6	shift	penalisation DISABLED	$\times$	$\times$
7	omegalim	penalisation DISABLED	$\times$	$\times$
8	reference	penalisation DISABLED	$\times$	$\times$

Following the experimental methodology already used, the best individuals produced by each run of the HyGP experiments listed in Table 5.30 were validated on the validation data set. The pathologies emerged are reported in Table 5.31.

TABLE 5.31: Pathologies on validation data sets for Kotanchek test case. All experiments consisted of 10 runs: %∞ is the percentage of individuals not defined, %bad is the percentage of individuals with a RMSE>100 on the particular data set. Median and IQR are computed on the remaining individuals.

Kotanchek	Validation data set			
	RMSE			
	%∞	%bad	median	IQR
Omegalim_shift p=3 a5=0.0001	10	0	1.228041e-01	1.209799e-01
Omegalim_shift p=3 a5=0.001	10	0	1.031283e-01	1.012077e-01
Omegalim_shift p=3 a5=0.01	10	0	9.902799e-02	5.594168e-02
Omegalim_shift p=2 a5=0.0001	0	0	6.293210e-02	6.477281e-02
Omegalim_shift	0	0	9.451645e-02	2.356540e-02
shift	10	0	1.047431e-01	4.677519e-02
Omegalim	0	0	8.571021e-02	9.749058e-03
reference	10	0	1.108534e-01	3.393596e-02

The RMSE and R^2 samples generated by the experiments are represented in Fig. 5.28 using boxplots. Each distribution is made of 10 samples, corresponding to the RMSE or R^2 value of the best metamodel generated by each of the 10 independent evolutions the experiment consisted of. RMSE and R^2 were evaluated on the validation data set.



(A) RMSE (B) R^2

FIGURE 5.28: Distribution of RMSE and R^2 values on validation data set returned by the best metamodels generated by GP for different values of p

According to the p-value returned by Kruskal-Wallis[35] test (0.21514) there is not enough evidence to support the hypothesis that the use of the penalisation defined in Eq. (5.37) produced a change in the RMSE and R^2 median value (at 5% significance

[35]ANOVA p-value = 0.48439.

level). However, boxplots show that RMSE and R^2 interquartile range increases when penalisation is enabled, indicating that the feasibility penalisation may increase syntax trees variety during the evolution and, extending the training stage (or youth stage), leads to individuals of better quality. The actual mechanisms that cause an increase in the RMSE and R^2 interquartile ranges when penalisation is enabled are however still unknown and further research is needed to understand how penalisation biases HyGP evolution.

The effect of the feasibility penalisation on Kotanchek metamodels structure can be appreciated from the analysis of the mathematical expressions of the best metamodels generated by "Omegalim_shift" and "Omegalim_shift p=3 a5=0.0001", reported in Eq. (5.39) and Eq. (5.40), respectively.

$$\begin{aligned}\tilde{f}(z_1, z_2) &= -0.120801 + (0.0442430\,\mathrm{z_2}^2)^3 - 0.0436809\,\mathrm{z_1}^2\,\mathrm{z_2} + 0.0872379\,\mathrm{z_1} \\ &+ 0.00899636\,\mathrm{z_1}^2\,\mathrm{z_2}^2 - 0.00499831\,\mathrm{z_2}^4 + 0.279459\,\mathrm{z_2} + 0.168453\,\sin(1.71415\,\mathrm{z_1})\end{aligned} \tag{5.39}$$

$$\begin{aligned}\tilde{f}(z_1, z_2) &= -0.0241637 \\ &+ \frac{[33.5706\,\sin(1.44049\,\mathrm{z_1}) - 0.923777\,\mathrm{z_2} + 62.1925]^2 + 265.5989}{(68.3857\,\mathrm{z_2} - 105.061)^2 + (60.4282\,\mathrm{z_2} - 228.862)^2 + (60.0122\,\mathrm{z_1} - 44.1352)^2 - 269.67}\end{aligned} \tag{5.40}$$

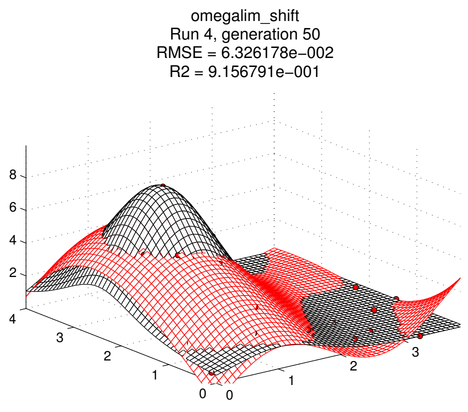
The difference between Eq. (5.39) and Eq. (5.40) is evident: feasibility penalisation allowed HyGP to focus the search on rational metamodels, class which Kotanchek function belongs to. As observed in Section 5.3.6 HyGP struggles to infer rational functions, so the tested penalisation can be useful to improve HyGP performances in such cases.

In Table 5.32 the RMSE and the coefficient of determination (R^2) evaluated on the validation data set for the metamodels defined in Eq. (5.39) and in Eq. (5.40) are compared.

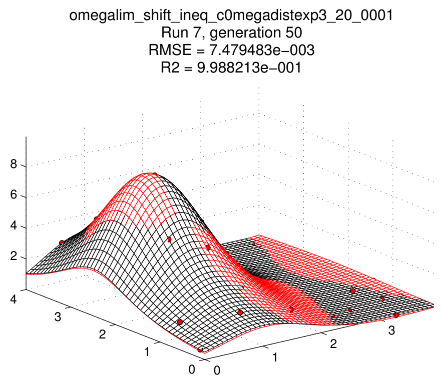
TABLE 5.32: RMSE and R^2 values of the metamodels defined in Eq. (5.39) (penalisation not used: Omegalim_shift) and in Eq. (5.40) (penalisation used - Omegalim_shift p=3 a5=0.0001)

	RMSE	R^2
Omegalim_shift	0.078081	0.852949
Omegalim_shift p=3 a5=0.0001	0.008666	0.998189

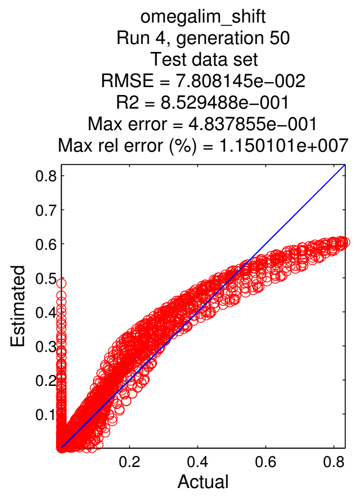
Fig. 5.29 and Fig. 5.30 show the plots of the two metamodels defined in Eq. (5.39) and in Eq. (5.40) superimposed to the Kotanchek function plot.



(A) Kotanchek function (in black) and model generated without penalisation (in red)

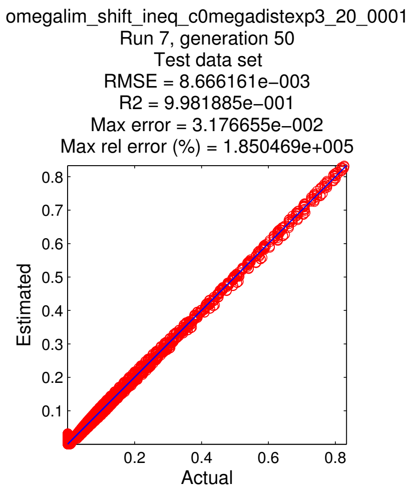


(A) Kotanchek function (in black) and model generated with penalisation (in red)



(B) Model response vs. actual response

FIGURE 5.29: Penalisation not used: *Omegalim_shift*



(B) Model response vs. actual response

FIGURE 5.30: Penalisation used: Omegalim_shift p=3 a5=0.0001

5.5 Genetic programming versus polynomial chaos expansion

So far different strategies to improve HyGP performances have been described. It is however useful to compare HyGP with other metamodelling techniques, to assess its advantages and drawbacks. In this section a series of tests with a parametric technique called Polynomial Chaos Expansion (PCE) is presented. The results here presented are the outcomes of a research activity carried out in collaboration with a major aerospace company with the aim of improving and expanding their metamodelling tool based on PCE.

PCE is a metamodelling technique that uses a linear combination of polynomials of increasing dimensionality (Eldred et al. 2008):

$$\begin{aligned} y =& a_0 B_0 + \sum_{i_1=1}^{\infty} a_{i_1} B_1(\xi_{i_1}) + \sum_{i_1=1}^{\infty}\sum_{i_2=1}^{i_1} a_{i_1 i_2} B_2(\xi_{i_1}, \xi_{i_2}) \\ &+ \sum_{i_1=1}^{\infty}\sum_{i_2=1}^{i_1}\sum_{i_3=1}^{i_2} a_{i_1 i_2 i_3} B_3(\xi_{i_1}, \xi_{i_2}, \xi_{i_2}) + \dots \end{aligned} \tag{5.41}$$

where ξ_{i_k} are input variables modelled by given statistical distributions, B_{i_k} are multivariate polynomials of increasing dimensionality and a_{i_k} are coefficients that have to be tuned on a building DoE provided by the user. The expression in Eq. (5.41) can be reformulated as (Eldred et al. 2008):

$$y = \sum_{j=0}^{\infty} \alpha_j \Psi_j(\xi) \tag{5.42}$$

where the terms $\Psi_j(\xi)$ stand for multivariate polynomials of increasing dimensionality and the coefficients α_j replace the a_{i_k} in Eq. (5.41).

As the mathematical structure of a PCE metamodel is imposed by the sum of the multivariate polynomials B_{i_k} (see Eq. (5.41)) or, equivalently, $\Psi_j(\xi)$ (see Eq. (5.42)), PCE belongs to the class of parametric metamodelling techniques (Friedman 1991) (see Section 1.2.4.1, Chapter 1).

The numerical coefficients α_j of the truncated PCE expansion need to be tuned to obtain the metamodel. The number of coefficients α_j to be found is equal to the number of terms in the truncated PCE, which is equal to (Eldred et al. 2008, p. 3):

$$N = \frac{(p+d)!}{p!\ d!} \tag{5.43}$$

where p is the maximum order of the truncated PCE and d is the number of independent input variables. Although the N coefficients α_j can be computed using exactly N points, many researchers (Eldred et al. 2008, Georgiou and Cooper 2011) have shown the advantages of evaluating them through a least-squares approach using more points than the minimum number N. The expression “oversampling ratio” (r_{over}) will be used to indicate the ratio between the actual number of points N_{used} used to compute the coefficients and the minimum number N required to do it:

$$r_{over} = \frac{N_{used}}{N} \tag{5.44}$$

Eldred et al. (2008, p. 8) for example suggests using an oversampling ratio of 2.

The multivariate polynomials $\Psi_j(\xi)$ are usually defined as products of one-dimensional polynomials belonging to the so-called Askey scheme (Eldred et al. 2008), as in this case the metamodels produced truncating the PCE in Eq. (5.42) can be used to extract analytically the contribution of each input variable to the total variance of the approximated output without requiring any additional sampling. PCE was chosen as the main modelling tool by the aerospace company that supported the analysis detailed in the following as sensitivity analysis can be performed analytically, so immediately, on the PCE model without using Monte Carlo approaches (Sobol 1993, Sudret 2008, Arwade et al. 2010). This feature was considered so important as to neglect the usual lack of flexibility and the extra cost of screening (finding the correct mathematical structure - order of the PCE expansion) intrinsic to parametric techniques. This anyhow proves that in practical cases the concept of "best" machine learning algorithm or modelling technique is quite relative and subject to the actual needs of the final user, who can be interested in other features beyond sheer accuracy (for example PCE can be used just for sensitivity analysis, but not for modelling).

5.5.1 Test functions

The comparison of PCE and HyGP was performed on two symbolic regression problems:

$$f(x_0, x_1) = 100\left(x_0 - x_1^2\right)^2 + (1 - x_1)^2 \tag{5.45}$$

$$f(x_0, x_1) = \frac{e^{-x_0^2}}{1.2 + x_1^2} \tag{5.46}$$

the expression in Eq. (5.45) is the so-called *Rosenbrock* function. This function was not used to test HyGP enhancements (see Section 5.3) because it has proven too simple for HyGP. Eq. (5.46) is instead a reformulation of the Kotanchek function introduced in Section 5.3.1.1.

Due to limitations imposed by the metamodelling software to the selection of the univariate polynomials[36] for the generation of the PCE, the input variables of both functions had to be assumed to follow a normal distribution. Also, instead of a uniform latin hypercube DoE, a normally-weighted Latin Hypercube sampling strategy (Helton and Davis 2003) was used to generate the building data sets. This restriction was however useful to

[36]only Hermite polynomials were available.

assess the ability of HyGP to deal with DoEs different from the uniform latin hypercube used so far.

The parameters used to compare the quality of the metamodels generated by the two techniques are the root mean square error (RMSE) and the coefficient of determination (R^2) computed on a validation data set defined as a full-factorial DoE on the design space and made of a number of points far larger than the building data set.

5.5.2 Results for Rosenbrock function

The input variables x_0 and x_1 were assumed normal with zero mean and a standard deviation equal to 2/3:

$$x_0 \sim N(0.0, 0.66667) \tag{5.47}$$

$$x_1 \sim N(0.0, 0.66667) \tag{5.48}$$

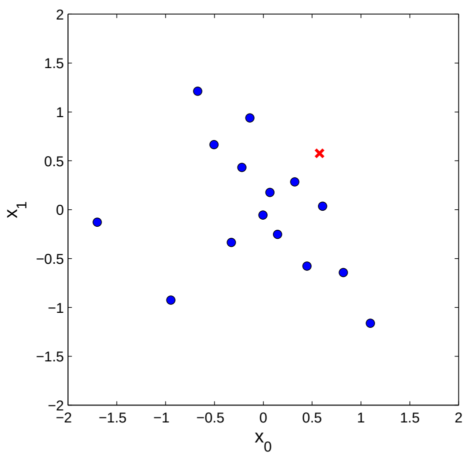
As the dimensionality of the problem is $d = 2$ and the maximum order of the PCE polynomial was set to $p = 4$, the minimum number of DoE points needed for the estimation of the PCE coefficients is $N = 15$ (see Eq. (5.43)). Two experiments were performed using initially a 16-point DoE ($r_{over} = 1$, as the tool used all except one point for metamodel building) and then increasing the DoE size to 31 points ($r_{over} = 2$, one point not used for metamodel building).

The normally-weighted Latin Hypercube DoE used for metamodel building is shown in Fig. 5.31. A full-factorial DoE made of 2500 points in $[0, 2] \times [0, 2]$ was used as a validation data set. The root mean square error (RMSE) and the coefficient of determination (R^2) computed on this data set were finally used to compare the metamodels generated by PCE and HyGP.

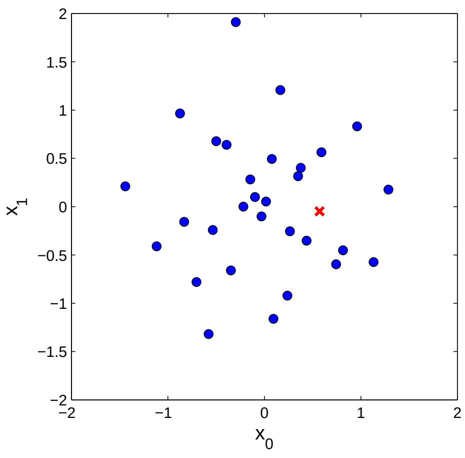
5.5.2.1 Metamodel generated by polynomial chaos expansion

The obtained PCE metamodels are plotted in red in Fig. 5.32A ($r_{over} = 1$) and Fig. 5.33A ($r_{over} = 2$) superimposed to the actual Rosenbrock function (in black). The subplots B and C compare the actual Rosenbrock response with the modelled response for each point on the building and validation data set, respectively. In Table 5.33 are reported the RMSE and the R^2 of the generated PCE metamodels evaluated on the validation data set.

From the figures and the table it emerges that the minimum number of points required for PCE parameter tuning ($r_{over} = 1$) was not enough to generate an accurate metamodel



(A) 16-point DoE ($r_{over} = 1$)



(B) 31-point DoE ($r_{over} = 2$)

FIGURE 5.31: Latin Hypercube DoEs used for modelling Rosenbrock function ($r_{over} = 1, 2$). Building points are represented by •, points represented by $\times$ were used only for internal validation.

TABLE 5.33: RMSE and R^2 for Rosenbrock PCE metamodels

No. of points	r_{over}	RMSE	R^2
15	1	3.991337E+02	6.220599E-01
30	2	3.832273E-04	1.0000000

of Rosenbrock function. Using twice as many points ($r_{over} = 2$) the accuracy dramatically improved: the RMSE decreased by six orders of magnitude (from 3.991337E+02 to 3.832273E-04) and the R^2 increased to reach its upper bound, sign of perfect match between approximation and underlying function.

The text expression of the PCE metamodel generated using $r_{over} = 2$ is:

$$\begin{aligned}
\tilde{f}_{PCE}(x_0, x_1) = {} & 105.149782533 - 1.33333140256 * ((1.0 * ((x_0 - [0])/([0.66667]))^1)) \\
& - 59.2601467423 * ((1.0 * ((x_1 - [0])/([0.66667]))^1)) \\
& + 168.24239822 * ((-0.707106781187 + 0.707106781187 * ((x_0 - [0])/([0.66667]))^2)) \\
& + 3.68067489117\text{e-}06 * ((1.0 * ((x_0 - [0])/([0.66667]))^1) * (1.0 * ((x_1 - [0])/([0.66667]))^1)) \\
& + 62.8545583509 * ((-0.707106781187 + 0.707106781187 * ((x_1 - [0])/([0.66667]))^2)) \\
& + 1.40240452183\text{e-}05 * ((-1.22474487139 * ((x_0 - [0])/([0.66667]))^1 \\
& + 0.408248290464 * ((x_0 - [0])/([0.66667]))^3)) \\
& - 83.8065162129 * ((-0.707106781187 + 0.707106781187 * ((x_0 - [0])/([0.66667]))^2) * \\
& \qquad (1.0 * ((x_1 - [0])/([0.66667]))^1)) \\
& + 2.85742853065\text{e-}06 * ((1.0 * ((x_0 - [0])/([0.66667]))^1) * \\
& \qquad (-0.707106781187 + 0.707106781187 * ((x_1 - [0])/([0.66667]))^2)) \\
& + 4.10831781727\text{e-}07 * ((-1.22474487139 * ((x_1 - [0])/([0.66667]))^1 \\
& + 0.408248290464 * ((x_1 - [0])/([0.66667]))^3)) \\
& + 96.7718974897 * ((0.612372435696 - 1.22474487139 * ((x_0 - [0])/([0.66667]))^2 \\
& + 0.204124145232 * ((x_0 - [0])/([0.66667]))^4))
\end{aligned} \tag{5.49}$$

5.5.2.2 Metamodel generated by genetic programming

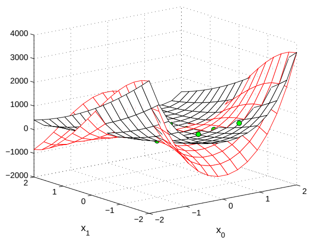
The exact Rosenbrock function was generated by HyGP from the smallest DoE made of 15 points[37]. If slight errors on the coefficients are neglected, the following metamodel, produced by HyGP:

$$\tilde{f}_{GP}(x_0, x_1) = 1.00000 + \left[(-3.16228x_1)^2 - 10.00000x_0\right]^2 - 2.00002x_1 + (-0.99999x_1)^2 \tag{5.50}$$

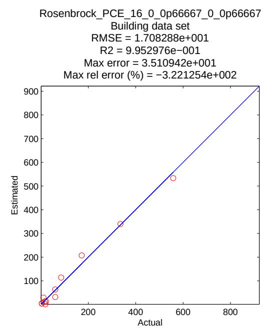
is identical to Eq. (5.45). The metamodel reported in Eq. (5.50) is plotted in Fig. 5.34A superimposed to Rosenbrock function. In Figs. 5.34B and 5.34C the appoximated response is plotted against the actual response on building and validation data sets. Metamodel RMSE and R^2 on the validation data set are reported in Table 5.34.

To assess the computational costs of the symbolic regression performed by HyGP, it should be taken into account that the metamodel reported in Eq. (5.50) was the best out

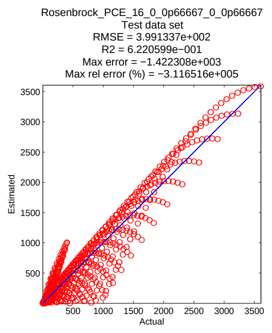
[37]The input settings used to generate the metamodel are reported in Table C.6 in Section C.6, Appendix C.



(A) metamodel (red) and actual function (black)

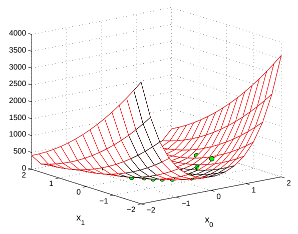


(B) actual vs. estimated response - building data set

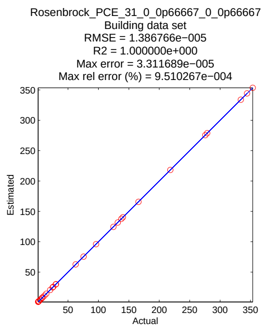


(C) actual vs. estimated response - validation data set

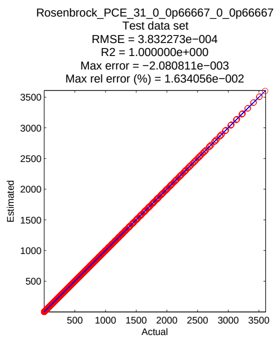
FIGURE 5.32: Rosenbrock metamodel returned by PCE - 15 points ($r_{over} = 1$)



(A) metamodel (red) and actual function (black)

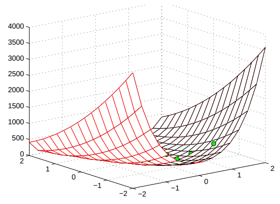


(B) actual vs. estimated response - building data set

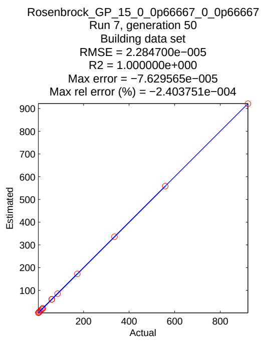


(C) actual vs. estimated response - validation data set

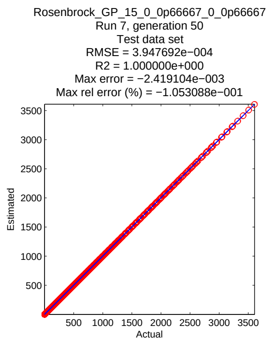
FIGURE 5.33: Rosenbrock metamodel returned by PCE - 30 points ($r_{over} = 2$)



(A) metamodel (red) and actual function (black)



(B) actual vs. estimated response - building data set



(C) actual vs. estimated response - validation data set

FIGURE 5.34: Rosenbrock metamodel returned by HyGP - 15 points

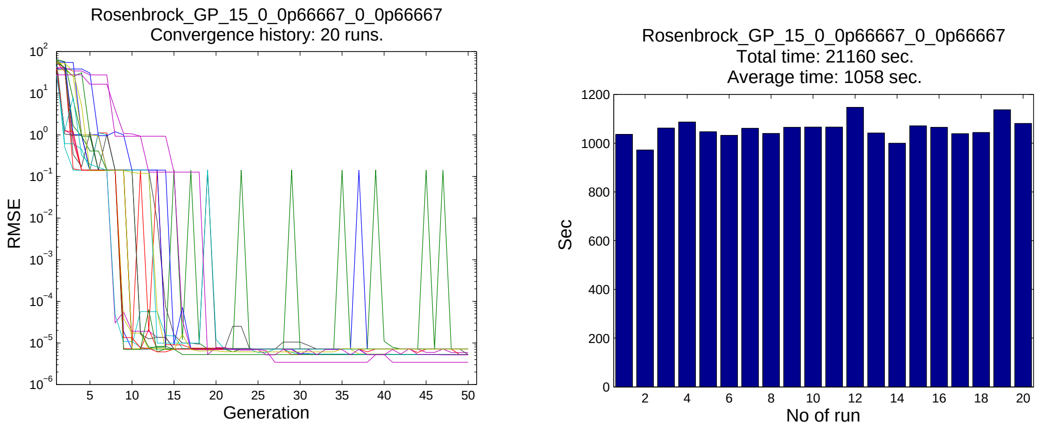
TABLE 5.34: RMSE and R^2 for the Rosenbrock metamodel generated by HyGP

No. of points	RMSE	R^2
15	3.947692E-04	1.000000

of 20 generated metamodels. All runs were performed in parallel mode[38] on a laptop running Ubuntu Linux 3.0.0 equipped with two AMD Turion 64 X2 TL-50 processors and

[38]the parallel implementation used will be described in Section A.3.2, Chapter A.

2.0 GB of RAM. The time required by each HyGP run is shown in Fig. 5.35 together with the runs' convergence history.



(A) GP convergence history (on building data set)

(B) Time required

FIGURE 5.35: Rosenbrock function: convergence history and time required by GP runs

5.5.3 Results for Kotanchek function

Input variables were assumed to be normally distributed, with zero mean and standard deviation $\sigma = 2$:

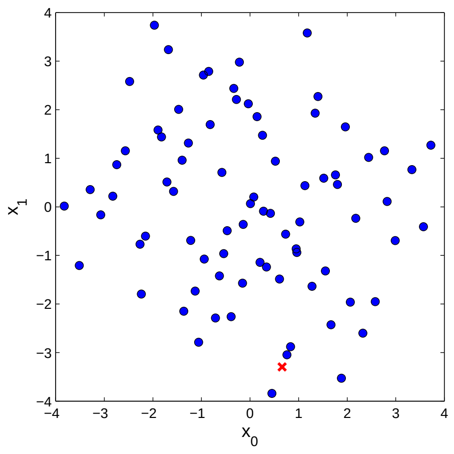
$$x_0 \sim N(0.0, 2.0) \tag{5.51}$$

$$x_1 \sim N(0.0, 2.0) \tag{5.52}$$

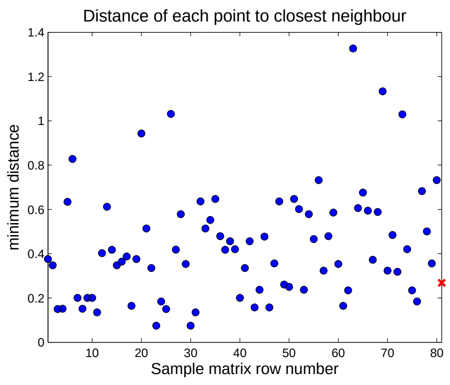
As the objective function is not a polynomial and the order of the PCE that best approximates it is not known (common scenario in industrial applications), the oversampling issue was not taken into account. The maximum order of the PCE was set to $p = 6$. A normally-weighted Latin Hypercube sampling was used to generate a building data set made of 80 points in the region $[-4, 4] \times [-4, 4]$. The building data set is shown in Fig. 5.36.

5.5.3.1 Comparison of PCE and GP metamodels

The best metamodels generated by PCE and HyGP are plotted in red in Fig. 5.37A and in Fig. 5.38A superimposed to the Kotanchek function (in black). The subplots B and C compare the Kotanchek function response with the modelled response for each point on the building and validation data set, respectively.

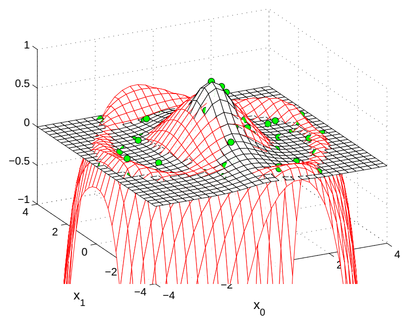


(A) DoE

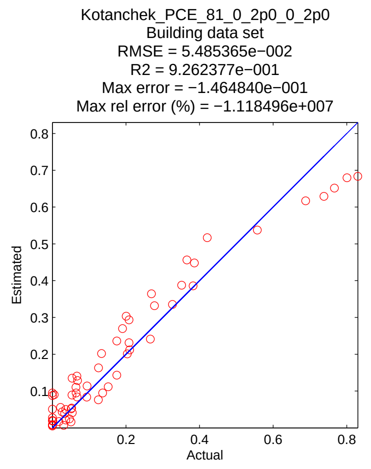


(B) minimum distance between points

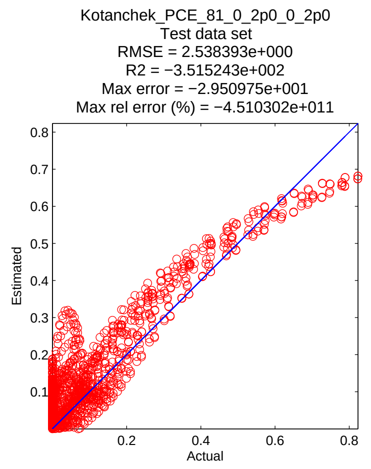
FIGURE 5.36: DoE used as building data set for Kotanchek test function and minimum distance between DoE points (building samples are represented by •, points represented by × were used only for internal validation)



(A) metamodel (red) and actual function (black)

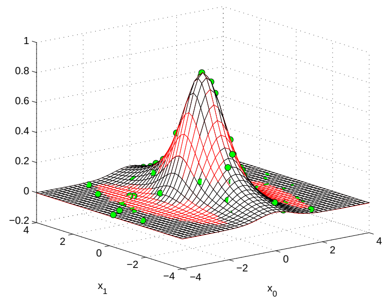


(B) actual vs. estimated response - building data set

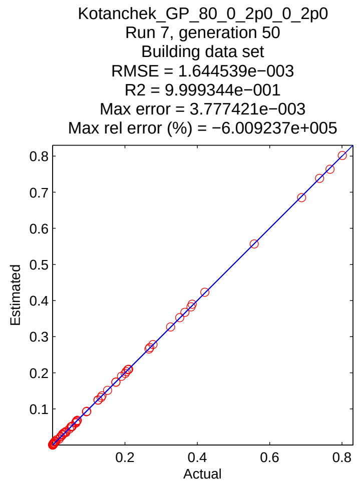


(C) actual vs. estimated response - validation data set

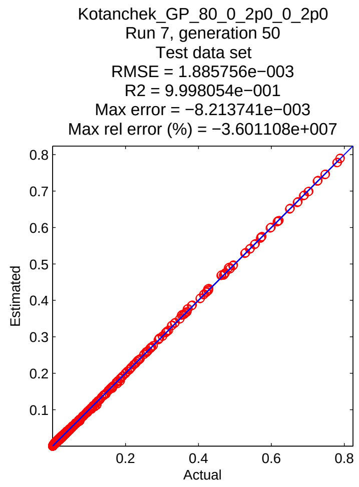
FIGURE 5.37: Kotanchek function metamodel generated by PCE (80 points)



(A) metamodel (red) and actual function (black)



(B) actual vs. estimated response - building data set



(C) actual vs. estimated response - validation data set

FIGURE 5.38: Kotanchek function metamodel generated by HyGP (80 points)

The superior accuracy and generalisation ability of the best metamodel generated by HyGP is proved by the substantial difference in RMSE and R^2 errors on the validation data set with respect to the PCE metamodel, as shown in Table 5.35.

TABLE 5.35: RMSE and R^2 for Kotanchek function metamodels generated by PCE and HyGP

Metamodelling technique	RMSE	R^2
PCE	2.538393E+00	-3.515243E+02
HyGP	1.885756E-03	9.998054E-01

Interestingly, HyGP was able to recognise that the function to be modelled was rational, and as a result the generated metamodel was far more compact than the PCE polynomial, with obvious advantages in terms of interpretability, ease of handling, use of RAM and evaluation time. The expression of the HyGP metamodel is shown in Eq. (5.53):

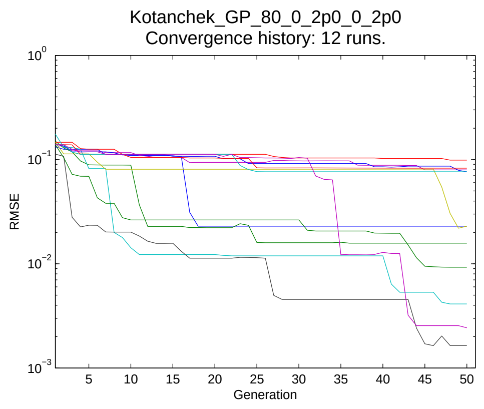
$$\begin{aligned}\tilde{f}_{GP}(x_0, x_1) &= -0.00250846 \\ &+ \frac{345.968 + 0.128425x_1 - 0.204097x_0}{413.08610 + (17.2227x_1x_0^2)^2 + (-17.1527x_1x_0)^2 + (15.3829x_0^3)^2 + (21.4576x_0)^2 + (18.5243x_1)^2}\end{aligned} \tag{5.53}$$

The input settings used to generate the metamodel reported above can be found in Section C.7, Appendix C. As noted for the Rosenbrock function case, indications on the computational cost of the regression can be obtained considering that the metamodel reported in Eq. (5.53) was the best out of the 12 generated metamodels. All runs were performed in parallel mode[39] on a laptop running Ubuntu Linux 3.0.0 equipped with two AMD Turion 64 X2 TL-50 processors and 2.0 GB RAM. The time required for each GP run is shown in Fig. 5.39 together with the runs' convergence history.

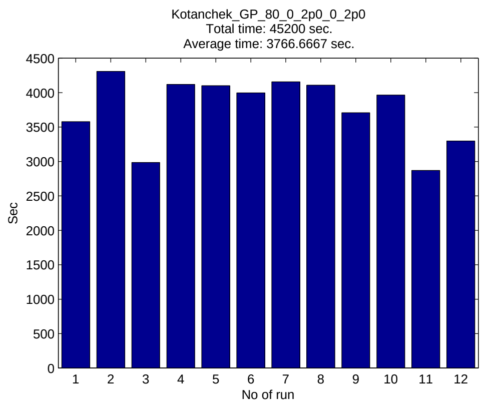
5.5.4 Conclusion

The better accuracy and generalisation of HyGP with respect to PCE show that HyGP has the potential to be used by the company who funded the research as a valid substitute to PCE. Of course HyGP does not allow for sensitivity analysis through analytical means, but the expression returned by HyGP can be used with Monte Carlo methods to perform fast sensitivity analysis, as the output of HyGP expression can be evaluated almost instantly.

[39]the parallel implementation used will be described in Section A.3.2, Chapter A.



(A) GP convergence history (on building data set)



(B) Time required

FIGURE 5.39: Kotanchek function: convergence history and time required by GP runs

5.6 Genetic programming versus moving least squares method

The weight minimisation of a planar 10-bar truss is a well known test case for metamodelling and optimisation algorithms (Haftka and Gürdal 1993, Elishakoff et al. 1994, Herencia and Haftka 2010, Hofwing et al. 2011, Lamberti and Pappalettere 2011). Its formulation is relatively simple and analytical models describing the axial force acting in the bars are available for a wide range of concentrated loads applied to the structure (Elishakoff et al. 1994, Herencia and Haftka 2010). Despite the problem apparent simplicity, bars' axial stresses are highly non linear and so their modelling represents a challenge for most metamodelling techniques, given also the high dimensionality of the problem.

The problem was therefore considered a good test case to compare HyGP with a parametric technique, the moving least squares method (MLSM) presented in Section 1.2.7 Chapter 1, on a more complex problem than the ones seen so far (see Section 5.3.1.1). Both techniques were used to generate metamodels of the axial stresses in the truss bars from the output generated by the analytical expressions of such stresses, whereas the optimum design was found through a GA search on the generated metamodels. It may be observed that modern simulation tools like finite element modelling (FEM) could also be used to compute the axial stresses in the bars once the cross sectional areas are known. Although correct, this approach was not chosen, mainly because the aim is to assess if HyGp is able to "understand" from data the structure of the underlying function that explains the outputs produced (see discussion in Section 5.3.1.1 sparked by observations reported in Korns (2011)), which in this rare case are available. FEM analysis would return required outputs for any DoE point, but of course no explicit model linking these outputs. Moreover, if exact analytical solutions exist, it appeared reasonable to use them, as more accurate than FEM and immediately available.

As for HyGP modelling, due to the nature of the problem, the penalisation approach described in Section 5.4 could not be applied. The penalisation can be applied only in case the output of the function to approximate is intrinsically bounded, by definition or for the physics of the problem. In this case the stresses in the bars are not bounded by any physical law or definition. The optimisation process has to find a truss which, given the loads, presents stresses that are included in a user defined range, but this does not imply that stresses should not go beyond the required bounds for other configurations.

5.6.1 Optimisation problem

The layout of the planar 10-bar truss to be optimised is shown in Fig. 5.40. The structure is assumed to be subjected to the loads P_1 and P_2 described in the figure.

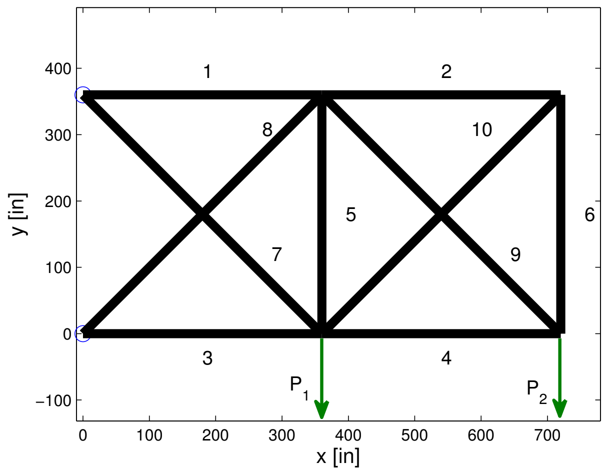


FIGURE 5.40: 10-bar truss with loads P_1 and P_2 (the bars are identified with numbers)

In mathematical terms the optimisation problem can be defined as follows:

$$find : A_i \quad i = 1, \ldots, 10 \tag{5.54}$$

$$minimising : m(\mathbf{A}) = \sum_{i=1}^{10} \rho_i l_i A_i \tag{5.55}$$

$$subject\,to : 0.1\,in^2 \leq A_i \leq 10.0\,in^2 \quad i = 1, \ldots, 10 \tag{5.56}$$

$$|\sigma_i| \leq 25000 \text{ psi} \;\; i = 1, \ldots, 8, 10 \tag{5.57}$$

$$|\sigma_9| \leq 75000 \text{ psi} \tag{5.58}$$

where A_i are the cross-sectional areas of the truss bars, whose length is $l_i = 360$ inches for the bars on the sides ($i = 1, \dots, 6$) and $l_i = 360 \cdot \sqrt{2} = 50.9117$ inches ($i = 7, \dots, 10$) for the diagonals. The bars are supposed to be all made of aluminium with density $\rho_i = 0.1 \; lb/in^3$ ($i = 1, \dots, 10$) and $m(\mathbf{A})$ is the total mass of the truss, objective of the minimisation. The magnitude of the loads P_1 and P_2 applied to the lower bars of the truss (see Fig. 5.40) is 10^5 lbf. Constraints were imposed on the absolute normal stress (tensile or compressive) $|\sigma_i|$ in the bars[40].

The optimisation problem defined in (5.54-5.55-5.56-5.57-5.58) is particularly challenging for HyGP as axial forces, and so axial stresses, display a highly non-linear behaviour, being rational functions of bars' cross-sectional areas (Herencia and Haftka 2010). HyGP difficulty in modelling rational functions was noted in Section 5.3.6. The expressions of the axial forces in the bars as a function of the cross sectional areas are shown below (adapted from Herencia and Haftka (2010), setting $P_3 = 0$):

$$N_1 = P_2 - \frac{\sqrt{2}}{2} N_8 \tag{5.59}$$

$$N_2 = N_6 = -\frac{\sqrt{2}}{2} N_{10} \tag{5.60}$$

$$N_3 = -P_1 - 2\,P_2 - \frac{\sqrt{2}}{2} N_8 \tag{5.61}$$

$$N_4 = -P_2 - \frac{\sqrt{2}}{2} N_{10} \tag{5.62}$$

$$N_5 = -P_2 - \frac{\sqrt{2}}{2} N_8 - \frac{\sqrt{2}}{2} N_{10} \tag{5.63}$$

$$N_7 = \sqrt{2}\,(P_1 + P2) + N_8 \tag{5.64}$$

$$N_8 = \frac{b_1 - a_{12}\,N_{10}}{a_{11}} \tag{5.65}$$

$$N_9 = \sqrt{2}\,P_2 + N_{10} \tag{5.66}$$

$$N_{10} = \frac{b_2 a_{11} - b_1 a_{21}}{a_{11} a_{22} - a_{12} a_{21}} \tag{5.67}$$

[40]It may be objected that buckling has not been explicitly addressed in the definition of the maximum allowable axial stress. The objection is correct, but the problem was considered only as a test case for HyGP metamodelling ability and not aimed at optimising a real structure. Morever, the stress constraints were taken from Haftka and Gürdal (1993).

where:

$$a_{11} = \left(\frac{1}{A_1} + \frac{1}{A_3} + \frac{1}{A_5} + \frac{2\sqrt{2}}{A_7} + \frac{2\sqrt{2}}{A_8}\right)\frac{L}{2E} \tag{5.68}$$

$$a_{12} = a_{21} = \frac{L}{2A_5E} \tag{5.69}$$

$$a_{22} = \left(\frac{1}{A_2} + \frac{1}{A_4} + \frac{1}{A_5} + \frac{1}{A_6} + \frac{2\sqrt{2}}{A_9} + \frac{2\sqrt{2}}{A_{10}}\right)\frac{L}{2E} \tag{5.70}$$

$$b_1 = \left(\frac{P_2}{A_1} - \frac{P_1 + 2P_2}{A_3} - \frac{P_2}{A_5} - \frac{2\sqrt{2}(P_1 + P_2)}{A_7}\right)\frac{\sqrt{2}L}{2E} \tag{5.71}$$

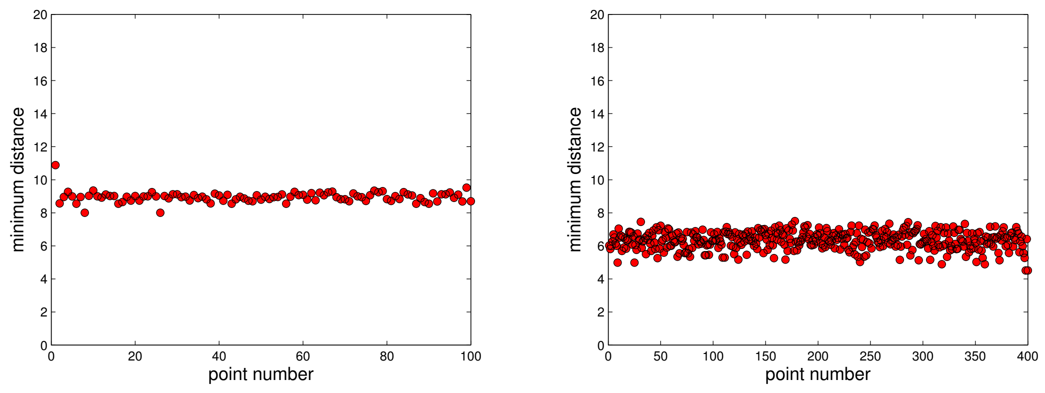
$$b_2 = \left(\frac{\sqrt{2}(-P_2)}{A_4} - \frac{\sqrt{2}P_2}{A_5} - \frac{4P_2}{A_9}\right)\frac{L}{2E} \tag{5.72}$$

where P_1 and P_2 are the applied forces, A_j ($j = 1, \ldots, 10$) are the bars' cross-sectional areas, L is the length of each horizontal bar and E is the Young's modulus of the bars' material.

A permutation GA algorithm (Bates et al. 2004) was used to generate two Latin hypercube DoEs in the 10-dimensional design space defined by (5.56). Preliminary tests showed that 100 points were enough for HyGP to infer the truss total mass $m(\mathbf{A})$. A 400-point DoE was instead generated to model the axial stresses in the bars: the formulae (5.59, ..., 5.72) were used to compute the bars' axial force for each combination of cross sectional areas defined by DoE points, then divided by the area to obtain the axial stress. The space filling properties and uniformity of the DoEs was checked evaluating the minimum Euclidean (2-norm) distance between DoE points: such distance is shown in Fig. 5.41 for each point in the two generated DoEs. The average minimum distance for the 100-point DoE used for total mass inference is 8.95, with a standard deviation of 0.32. For the 400-point DoE used for axial force inference the average minimum distance is 6.33, standard deviation 0.55.

5.6.2 Optimisation using HyGP metamodels

HyGP experiments made of 10 independent runs each were performed to generate the metamodels of the bars' axial forces and of the total mass. The combined *omegalim* and *shift* implementations were used in all cases (the input settings used are reported in Section C.8, Appendix C).



(A) mass (B) force

FIGURE 5.41: Minimum distance between points in the DoEs used for symbolic regression of total truss mass and bars' axial forces

HyGP was able to infer the correct expression for the total mass in all 10 independent runs using the 100-point DoE. The metamodels of the bars' axial stresses were obtained dividing the expressions generated by HyGP by the area of the corresponding bar. The selection of the most accurate axial force expression for each bar was hindered by the fact that no validation data set was used to assess the generalisation ability of each metamodel. Although the final metamodel selection was based on the highest coefficient of determination R^2 on the building data set, more than one metamodel for the same bar were considered in some cases. As a result, four different set of metamodels were used in the optimisation. The standard genetic algorithm embedded in HyperStudy (Toropov et al. 2005, Harewood et al. 2007, Alt 2009) was used to find the vector of cross sectional areas corresponding to the lightest truss satisfying the constraints on the bars' axial stresses.

The truss minimum mass and the bars' axial stresses returned by the four sets of metamodels are compared to the corresponding actual mass and stresses computed using the analytical solutions (5.59, . . . , 5.72) in Tables 5.36, 5.37, 5.38, 5.39. In the tables the second column contains the cross sectional areas of each bar, the third the actual stress σ, the fourth the stress estimated by HyGP metamodel σ_{est} and the last one the relative error on the stress computed as $100 * (\sigma_{est} - \sigma)/|\sigma|$. Bold is used whenever the actual stresses are beyond the allowable stress.

In Fig. 5.42 a schematic compares the suboptimal trusses resulting from the four different sets of HyGP metamodels (Figs. 5.42B - 5.42C - 5.42D - 5.42E) with the optimal truss (Fig. 5.42A). The optimal truss is defined by the vector of cross sectional areas A_{opt}

TABLE 5.36: First set. Validation of axial stresses and mass estimates at the suboptimum found by GA

bar (no.)	Area (in^2)	σ (lbf/in^2)	σ_{est} (lbf/in^2)	ϵ_{rel} (%)
1	8.74	21776.0	20533.6	-5.70
2	1.11	12301.5	14616.9	18.82
3	8.60	-24369.2	-24628.1	-1.06
4	5.15	-16758.9	-15796.7	5.74
5	1.50	2725.1	1972.4	-27.62
6	0.89	15372.9	15860.8	3.17
7	6.70	23140.5	23926.2	3.40
8	5.54	-23074.5	-20968.7	9.13
9	4.75	25695.0	25181.8	-1.99
10	1.02	-18874.7	-24642.1	-30.56

Objective:

	m (lb)	m_{est} (lb)	ϵ_{rel} (%)
total mass	1853.06	1853.06	0.0

TABLE 5.37: Second set. Validation of axial stresses and mass estimates at the suboptimum found by GA

bar (no.)	Area (in^2)	σ (lbf/in^2)	σ_{est} (lbf/in^2)	ϵ_{rel} (%)
1	8.96	19853.4	19102.5	-3.78
2	1.47	21989.9	24982.2	13.60
3	8.79	**-25358.1**	-24979.4	1.49
4	3.69	-18326.5	-16694.3	8.90
5	1.67	5729.8	11031.5	92.53
6	1.58	20483.3	24889.8	21.51
7	7.03	24694.9	24985.6	1.18
8	4.44	-24582.4	-24197.4	1.57
9	2.36	40458.4	37498.1	-7.32
10	1.80	**-25520.1**	-24823.6	2.73

Objective:

	m (lb)	m_{est} (lb)	ϵ_{rel} (%)
total mass	1736.46	1736.46	0.0

TABLE 5.38: Third set. Validation of axial stresses and mass estimates at the suboptimum found by GA

bar (no.)	Area (in^2)	σ (lbf/in^2)	σ_{est} (lbf/in^2)	ϵ_{rel} (%)
1	7.46	**25419.3**	24902.0	-2.03
2	1.18	14043.0	24891.9	77.25
3	8.34	**-25207.4**	-24978.4	0.91
4	4.11	-20.2621	-18383.8	9.27
5	0.46	13569.0	6463.0	-52.37
6	1.12	14990.3	24772.6	65.26
7	5.99	**26059.2**	24998.1	-4.07
8	6.62	-19168.7	-17584.8	8.26
9	3.69	31927.6	28691.4	-10.14
10	1.13	-20757.5	-24954.9	-20.22

Objective:

	m (lb)	m_{est} (lb)	ϵ_{rel} (%)
total mass	1703.63	1703.63	0.0

TABLE 5.39: Fourth set. Validation of axial stresses and mass estimates at the suboptimum found by GA

bar (no.)	Area (in^2)	σ (lbf/in^2)	σ_{est} (lbf/in^2)	ϵ_{rel} (%)
1	8.19	22825.2	21847.8	-4.28
2	1.05	20434.3	24954.6	22.12
3	8.98	-23734.1	-24999.9	-5.33
4	3.08	**-25503.9**	-21889.4	14.17
5	2.26	3679.6	5960.4	61.99
6	1.13	18937.0	24999.5	32.01
7	6.58	24312.9	24999.9	2.83
8	5.36	-22927.5	-22139.7	3.43
9	3.44	32332.4	29214.4	-9.64
10	1.28	-23558.9	-24967.2	-5.98

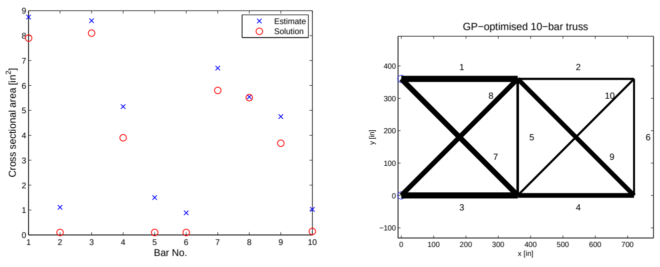
Objective:

	m (lb)	m_{est} (lb)	ϵ_{rel} (%)
total mass	1737.08	1737.08	0.0

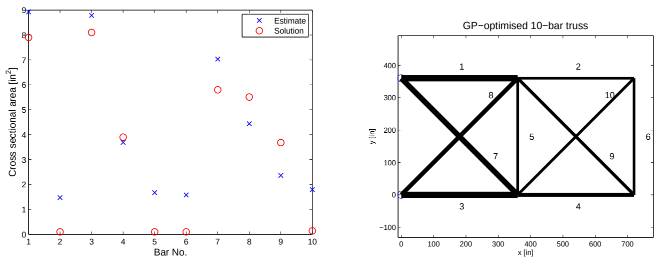
(measured in in^2):

$$A_{opt} = \{7.90\ 0.10\ 8.10\ 3.90\ 0.10\ 0.10\ 5.80\ 5.51\ 3.68\ 0.14\} \tag{5.73}$$

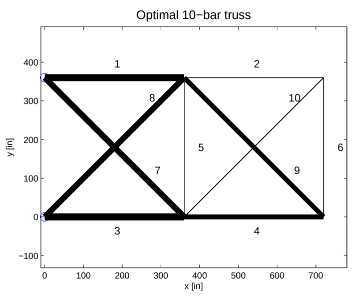
as reported by Haftka and Gürdal (1993), to which corresponds a mass of $1497.5\ lbs$. Simplified representations of the truss are used to highlight the differences in the final designs: the thicknesses of the bars are proportional to the corresponding bars' radii. Scatter plots in Fig. 5.42 are provided to show the discrepancy between the optimal cross sectional area and the estimated one for each bar ($\circ$ optimal value of the cross sectional area, $\times$ cross sectional area estimated by HyGP metamodels).



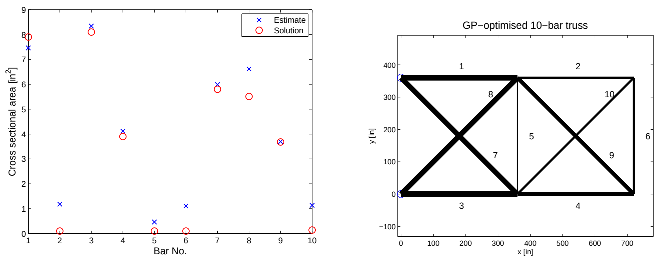
(B) Suboptimal truss found for the first set of metamodels (1853.06 lbs)



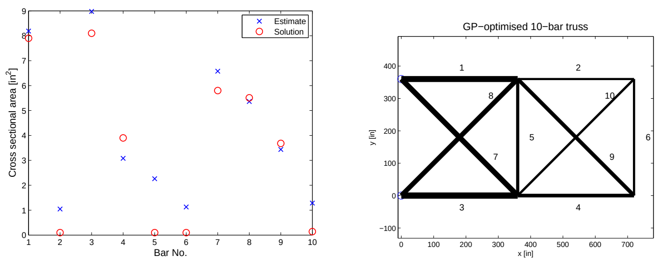
(C) Suboptimal truss found for the second set of metamodels (1736.46 lbs)



(A) Optimal truss (1497.5 lbs)



(D) Suboptimal truss found for the third set of metamodels (1703.63 lbs)



(E) Suboptimal truss found for the fourth set of metamodels (1737.08 lbs)

FIGURE 5.42: Schematic comparing the optimal 10-bar truss with the suboptimal trusses found using HyGP metamodels ($\circ$ optimal value of the cross sectional area, $\times$ cross sectional area estimated by HyGP metamodels)

5.6.3 Optimisation using MLSM metamodels

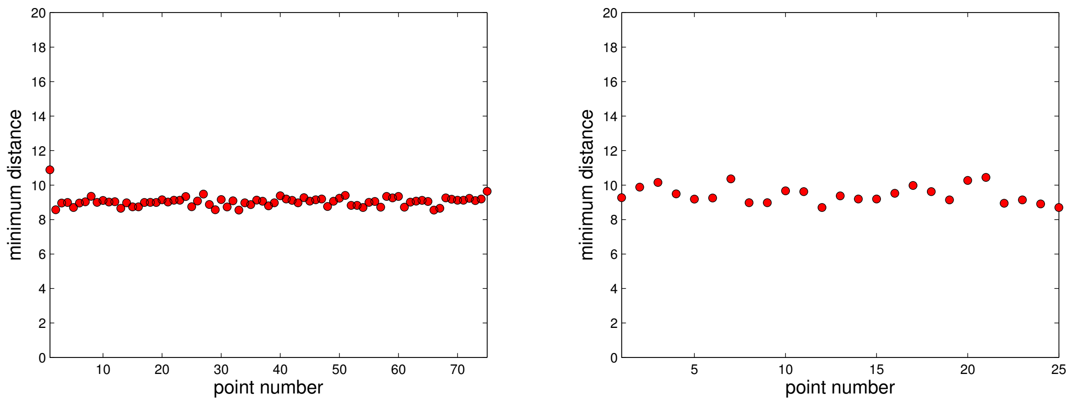
The optimisation was repeated using the moving least square method (MLSM), a parametric metamodelling technique whose theoretical background was introduced in Section 1.2.7, Chapter 1. The technique is available as a tool in the commercial optimisation software HyperStudy v11 (Toropov et al. 2005, Alt 2009) and is widely used in industry (Harewood et al. 2007, Zeguer and Bates 2011) for accuracy and robustness, so it was considered a valid and mature reference to compare the capability of HyGP with.

Polynomials of first, second and third order were used as basis for the MLSM method for the generation of global metamodels of the axial stresses. As for the total mass, a first order polynomial basis was of course enough to approximate the mass as a function of cross sectional areas. The same building data set processed by HyGP was used, although only two thirds of the samples were used to build the metamodels. The remaining third was used instead as a validation data set to tune the closeness of fit (automatic closeness of fit set in HyperStudy software), as suggested by Loweth et al. (2011). A Gaussian decaying function was used in all cases.

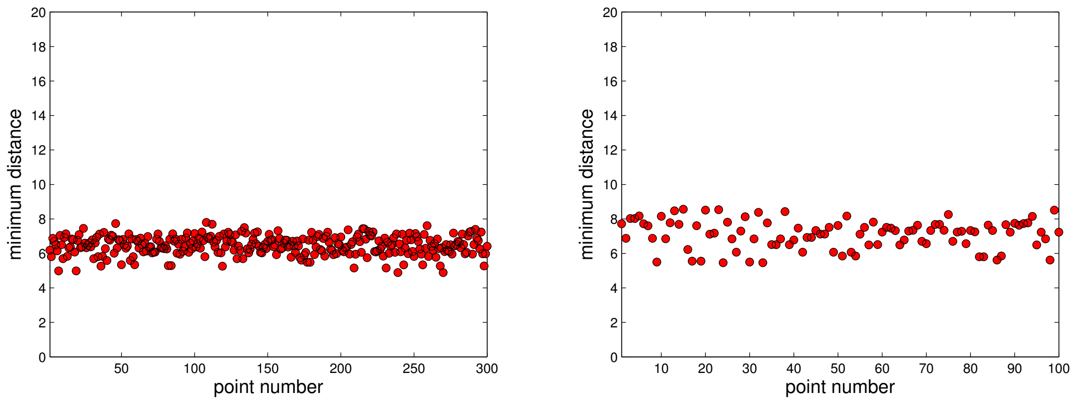
Building and validation data sets were generated splitting randomly the building DoE used for HyGP experiments. For mass model inference, 75 points were used for building, 25 for validation. For stress model inference, 300 points were used for building, the remaining 100 for validation. Although the point selection was done randomly, the building and validation data sets featured good space filling properties, as shown by the minimum distance between DoE points plotted for either data set in Fig. 5.43.

The coefficient of determination R^2 of the MLSM metamodels generated using polynomials of 1st, 2nd and 3rd order as basis are reported in Table 5.40: R^2 was evaluated on building and validation data sets as well as on the merged data set.

GA was used to find the set of cross sectional areas corresponding to the lightest truss satisfying the stress constraints, as done with HyGP metamodels. The optima found using the three different polynomial bases are shown in Tables 5.41, 5.42, 5.43.



(A) mass DoE points minimum distance in building (left) and validation (right) data sets



(B) stresses DoE points minimum distance in building (left) and validation (right) data sets

FIGURE 5.43: Minimum distance between DoE points in building and validation data sets used by MLSM to generate mass and stresses metamodels

TABLE 5.40: Value of the coefficient of determination R^2 of the stress and mass MLSM metamodels on building, validation and merged data sets

Axial stress σ_{est}

	Order 1			Order 2			Order 3		
bar	Build	Test	Merged	Build	Test	Merged	Build	Test	Merged
1	0.74639	0.57409	0.71714	0.84657	0.59424	0.82806	0.99910	-3.53790	0.98924
2	0.94606	0.63998	0.92953	0.83383	0.64683	0.81675	0.99454	-2.69958	0.96583
3	0.99225	0.62621	0.98796	0.98356	0.72556	0.97434	0.99796	-1.21364	0.97395
4	0.95203	0.61262	0.93301	0.81026	0.73389	0.80991	0.99693	-1.81781	0.96589
5	0.96988	0.57294	0.95274	0.86128	0.60607	0.82698	0.99635	-0.64826	0.96387
6	0.91967	0.45168	0.89176	0.85656	0.54941	0.82295	0.99367	-4.80223	0.91781
7	0.93331	0.65347	0.90891	0.97413	0.79497	0.96248	0.99983	-0.88337	0.99649
8	0.97115	0.66848	0.95666	0.95240	0.74254	0.93457	0.99879	-1.18166	0.98248
9	0.99679	0.72650	0.99517	0.98696	0.81313	0.98059	0.99842	-0.55860	0.98847
10	0.94943	0.63685	0.93014	0.94808	0.70466	0.92974	0.99891	-2.91754	0.98782

Mass m

	Order 1			Order 2			Order 3		
	Build	Test	Merged	Build	Test	Merged	Build	Test	Merged
	1.0	1.0	1.0	1.0	1.0	1.0	N/A	N/A	N/A

TABLE 5.41: Exact and estimated axial stresses and truss mass at the found suboptimum, using MLSM with 1st order polynomial basis

bar	Area	σ	σ_{est}	ϵ_{rel}
(no.)	(in^2)	(lbf/in^2)	(lbf/in^2)	(%)
1	9.86	20509.7	21847.8	6.52
2	0.10	21286.2	24954.6	17.23
3	5.40	**-36596.6**	-24999.9	31.69
4	2.37	**-41334.1**	-21889.4	47.04
5	0.10	**43705.1**	5960.4	-86.36
6	0.10	21286.2	24999.5	17.44
7	4.12	**33528.3**	24999.9	-25.44
8	7.33	-19719.2	-22139.7	-12.27
9	2.63	52575.0	29214.4	-44.43
10	0.10	**-30103.3**	-24967.2	17.06

Objective:

	m	m_{est}	ϵ_{rel}
	(lb)	(lb)	(%)
total mass	1367.93	1367.93	0.0

TABLE 5.42: Exact and estimated axial stresses and truss mass at the found suboptimum, using MLSM with 2nd order polynomial basis

bar	Area	σ	σ_{est}	ϵ_{rel}
(no.)	(in^2)	(lbf/in^2)	(lbf/in^2)	(%)
1	5.32	**36600.4**	550.2	-98.50
2	1.40	11886.9	17017.9	43.17
3	3.97	**-51738.1**	-24999.7	51.68
4	4.23	-19721.3	-23852.7	-20.95
5	1.54	7329.1	12162.7	65.95
6	1.83	9063.3	17427.7	92.29
7	4.84	**30769.5**	24998.0	-18.76
8	3.86	**-34673.8**	-24700.0	28.765
9	4.34	27198.7	38993.6	43.37
10	1.02	-22919.8	-24635.0	-7.48

Objective:

	m	m_{est}	ϵ_{rel}
	(lb)	(lb)	(%)
total mass	1374.12	1374.12	0.0

TABLE 5.43: Exact and estimated axial stresses and truss mass at the found suboptimum, using MLSM with 3rd order polynomial basis

bar	Area	σ	σ_{est}	ϵ_{rel}
(no.)	(in^2)	(lbf/in^2)	(lbf/in^2)	(%)
1	3.87	**62474.2**	24997.2	-59.99
2	0.79	7127.8	24977.1	250.42
3	2.40	**-66043.4**	-24926.1	62.26
4	3.20	**-29468.9**	12701.7	143.10
5	5.91	7981.8	-14580.5	-282.67
6	0.10	**56396.0**	22317.9	-60.43
7	3.15	**26215.2**	-13630.1	-151.99
8	8.34	-24008.9	8535.6	135.55
9	6.06	22029.9	12450.4	-43.48
10	7.88	-1011.5	10637.6	1151.67

Objective:

	m	m_{est}	ϵ_{rel}
	(lb)	(lb)	(%)
total mass	1880.72	1880.72	0.0

5.6.4 Comparison of results

The comparison of suboptima found using GP metamodels (Tables 5.36, 5.37, 5.38, 5.39) with the suboptima found by MLSM metamodels reveals the better performance of HyGP.

The optimal value of 1497.5 lbs for the 10-bar truss mass, reported by Haftka and Gürdal (1993), was not obtained by any attempts. The only truss design satisfying the stress constraints is represented by HyGP "first set" described in Table 5.36, resulting in a truss mass of 1853.06 lbs, 23.74% heavier than the optimal design. The other suboptimal designs found using HyGP metamodels are lighter but slightly violate the constraints, up to a maximum of 4.24% on the stress upper bound (bar 7 in Table 5.38). On the other hand, the violation of stress limits for the MLSM designs are much more critical. For the case of 3rd order polynomial basis (Table 5.43), the maximum stress level allowed is exceeded by 164%, with no particular gain in terms of mass (1880.72 lbs, when HyGP "first set" design results in 1853.06 lbs with no stress violations). The use of 1st or 2nd order polynomial basis does not improve the design, as shown in Tables 5.41 and 5.42: mass is generally lower than the solutions found by HyGP, but stress limits are violated far beyond the 4.24% recorded for HyGP designs.

The poor performance of MLSM can be partly ascribed to the fact that the optima is on the design space boundary: Harewood et al. (2007) also report on MLSM low accuracy in these circumstances. It can be concluded that HyGP and genetic programming in general have some extrapolation ability that make them particularly useful when the optima are located on the design space boundary.

5.6.5 Computational cost

HyGP experiments were performed using the parallel implementation described in Section A.3.3, Chapter A on a Linux cluster made of eight nodes, each equipped with several 3 MHz Intel Xeon processors.

The time required for each HyGP run to generate a metamodel for the axial force is plotted in Fig. 5.44 for all the ten bars. The average time is 79909 seconds, equal to 22 hours and 12 minutes. 12 GB of RAM were allotted to each run. HyGP settings are reported in Table C.8 in Section C.8, Appendix C. The generation of metamodels for the total mass was far faster, with an average time per evolution of 1 hour and 48 minutes. The high time required by HyGP to produce a metamodel is of course an issue that should be solved in the future both by optimisation of the code (memory handling, parallelisation, etc) and optimisation of HyGP parameters (population size, number of generations, number of tuning processes, etc).

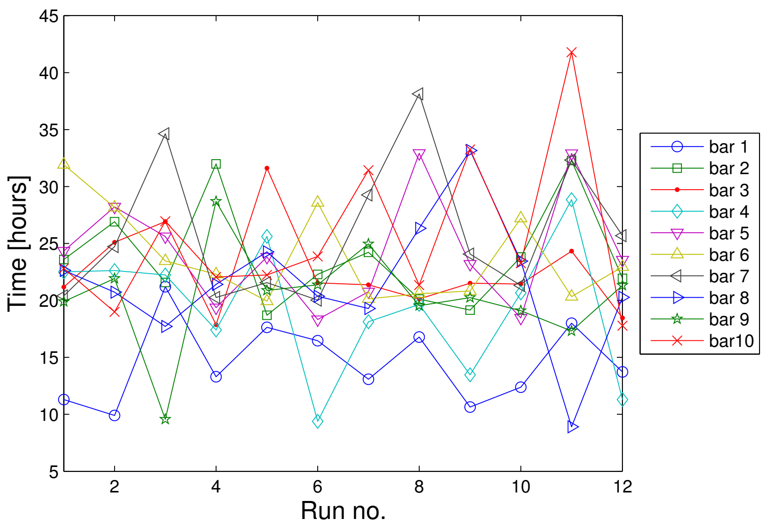


FIGURE 5.44: Time in hours for the generation of the axial forces HyGP metamodels

MLSM was far faster than HyGP, as to build a 3rd order model it took less than 5 minutes on a single processor laptop computer. However, it should be noted that metamodelling had to be repeated three times, one for each order of the polynomial, whereas HyGP is able to search for the better mathematical structure of the metamodel (*screening*) automatically.

5.7 Conclusion

In this chapter a new hybrid genetic programming algorithm, called HyGP, has been presented. A range of strategies to improve its performances, inspired by the analysis of the main GP pitfalls described in the previous chapters, have been proposed (see Section 5.3). The experiments set up to test the validity of these strategies (Section 5.3.5) have confirmed that the use of *Omegalim* strategy and the inclusion of the *shift* unary operator among the primitives effectively improve HyGP performances on a set of symbolic regression test problems.

A further strategy to improve the generalisation ability of the metamodels evolved by HyGP exploiting prior knowledge about the system to be modelled has been introduced (see Section 5.4). Tests on a benchmark have provided promising results, so the strategy is worth further attention in the future.

The final part of the chapter has been dedicated to the comparison of HyGP to other metamodelling techniques. Tests with polynomial chaos expansion (PCE) have shown the advantage of HyGP (and GP in general) on parametric techniques. On the two symbolic regression problems selected, HyGP was able to produce more accurate metamodels using fewer DoE points than PCE. Furthermore, HyGP metamodels were more compact and interpretable.

The 10-bar truss optimisation problem has been selected to test HyGP robustness on a highly non linear and high dimensional metamodelling problem. The reduced accuracy of HyGP metamodels was not able to lead the GA optimiser used to the global optimal design, but anyway it confirmed the good metamodelling capability of the developed code. Metamodels generated by HyGP led to a suboptimal solution but better than the one found by optimisation based on MLSM metamodels. The particular location of the optimum, lying on the design space boundary, was put forward as a possible explanation of the lower performance of MLSM. The experiment in any case confirmed that HyGP has some useful extrapolation capability.

The tests described in this chapter have proven that HyGP is robust enough to be used on real-life metamodelling problems. A comprehensive set of tests performed on industrial problems will be described in the next chapter.

Chapter 6

HyGP application to industrial problems

The main focus of Chapters 2, 3 and 4 has been genetic programming theoretical background and the exploration of the broad range of implementations formulated by GP researchers to overcome GP paradigm pitfalls. In Chapter 5 a new genetic programming software called HyGP has been introduced and experimental activity reported.

Although the validation activity has allowed to improve HyGP performances in terms of generalisation, accuracy and computational efficiency, HyGP use has so far been constrained to benchmark cases, which do not really reflect the complexity and variability of real industrial or academic problems. In this chapter HyGP is finally put to the test on more realistic problems, in order to provide an answer to the following questions:

- are models generated by HyGP accurate and robust enough to be used for real optimisation problems, where the computational cost of the simulations constrains the maximum size of the design of experiments?
- what is the level of human intervention required for setting up an experiment and for the final selection of the best metamodel? Is the process easily transferable to an industrial environment?
- what are HyGP limits and weaknesses in terms of computational cost and time? How does it cope with the "curse of dimensionality"?
- does HyGP bear comparison with other more traditional modelling techniques in terms of metamodels' accuracy?

As the following sections will focus on metamodels quality and use, this chapter can be considered as a natural extension of Chapter 1.

6.1 General optimisation framework

As broadly described in Chapter 1, the use of metamodels in modelling and optimisation allows to reduce the number of direct experimental test or numerical simulations required to identify the behaviour of a complex system and to optimise its response. In the last decades this advantage has been recognised by industry and academia (Jin et al. 2001), so that the use of metamodels in lieu of actual simulations for modelling and optimisation purposes is nowadays widely spread (Toropov et al. 2005, Bonte et al. 2005, Shahpar et al. 2008, Loweth et al. 2011).

Metamodel building, validation and exploitation stages have already been introduced in Section 1.2, Chapter 1: typically the experimental data are fed into a modelling technique and a metamodel (or more than one) is returned. The exploitation generally implies extrapolation or forecast of the system behaviour for unsampled combinations of the input parameters. For optimisation tasks, an optimiser is generally used to find the set of input parameters minimising or maximising a predefined cost or objective defined as a function of the system output(s).

The role of the optimiser is to efficiently explore the design space. As a result, the selection of the search algorithm represents a delicate compromise. Population-based algorithms are renowned for their ability to scan efficiently the entire design space, even in case of noisy or non smooth objective functions. On the other hand, gradient-based approaches are far more efficient but their performance is negatively affected by noisy and non smooth functions and the results depend critically on the initial guess provided by the user. Consequently, the optimiser selection has to take into account the features of the metamodel to be explored. For instance, Ong et al. (2003) used EA to find an optimal wing design optimisation through metamodels generated by radial basis functions. Harewood et al. (2007) used MLSM coupled with GA to optimise a coronary stent. Shahpar et al. (2008) used an SQP optimiser and MAM metamodels to optimise the aerodynamic design of NASA rotor 37 compressor rotor blade. A Pareto-based EA was used by Syberfeldt et al. (2009) to explore metamodels generated by artificial neural networks (ANN). In some cases, combining stochastic and deterministic optimisers can be beneficial for the accuracy and the robustness of the search: for example Bonte et al. (2005) performed a metal forming process optimisation using SQP to explore Kriging metamodels. To reduce the probability of finding sub-optimal configurations, SQP initial guesses were generated randomly. A similar approach was used by Alvarez (2000) to tune GP individuals, although initial guesses were generated by a GA algorithm.

The final step of the optimisation process is generally the validation of the optimum (optima) found by the optimiser. A final set of experiments or simulations is generally performed to assess the discrepancy between the optimal values returned by the metamodel based optimisation process and the actual responses at the defined point (an example can be found in Rogers and LaMarsh (1995)).

The whole process above described is sketched in Fig. 6.1.

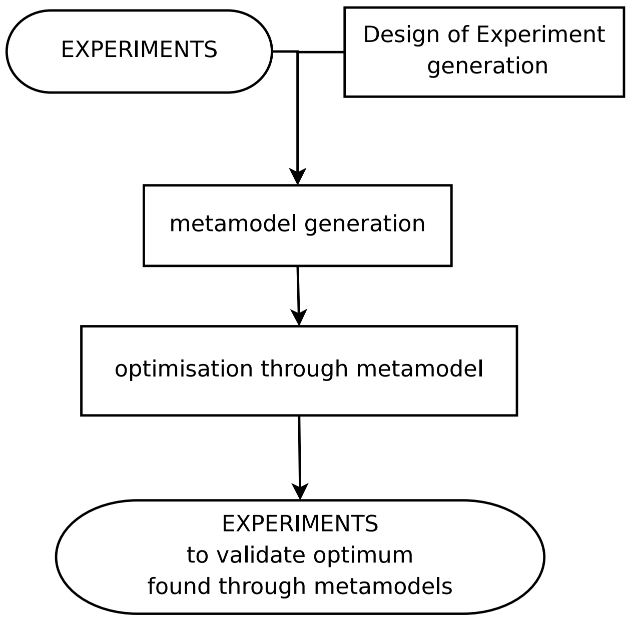

EXPERIMENTS
Design of Experiment generation
metamodel generation
optimisation through metamodel
EXPERIMENTS to validate optimum found through metamodels

FIGURE 6.1: Framework generally adopted in metamodel-based optimisation

6.2 GP in real applications

HyGP performance has been assessed on real modelling and optimisation problems: the case studies that have been selected for the test are reported in Table 6.1. The problems span very different fields, from fluid dynamics to chemistry, and feature a broad dimensionality range, from what can be considered low (1 to 4 variables) to high dimensionality (more than 10 variables). An example of mid dimensionality problem was the

10-bar truss optimisation, featuring 10 variables, described in the previous chapter, Section 5.6. Most data have originated from numerical simulations, while only in one case models were generated from data gathered from a real physical process. The data generated by numerical simulations were assumed reliable and hence not affected by noise (problems Hw, Jp, Bb, Af, Rb). Basically the same hypothesis was assumed for the experimental problem, as no particular measures were put in place by the experimenters to reduce measurement noise (problem Cd).

TABLE 6.1: List of selected modelling and optimisation problems

Problem		Dimension	Data	Sec.
ID	description		(experim./real)	
Hw	Hospital ward ventilation optimisation	1	numerical	6.3
Cd	Modelling of chromate diffusion process in aeronautical paint	2	experimental	6.4
Jp	Modelling of supersonic jet pump entrained flow rate	3	numerical	6.5
Bb	Bread baking oven design optimisation	3	numerical	6.6
Af	Structural optimisation of a lattice aircraft fuselage barrel	7	numerical	6.7
Rb	Aerodynamic optimisation of NASA rotor 37 compressor rotor blade	25	numerical	6.8

In all cases, the research activity consisted in applying HyGP to the data produced or collected by other researchers. In the next sections the metamodelling and optimisation activity is described in detail, following the framework described in the previous section (Fig. 6.1). For the problem having the highest dimensionality (Rb) the computational cost associated with the metamodel generation is reported. The settings used for each problem are reported in Appendix C.

6.3 Hospital ward ventilation optimisation

Ventilation in healthcare environments is critical to ensuring patient comfort and at the same time reducing the transmission of airborne infectious particles to other patients or healthcare workers. CFD simulations have been used to simulate air temperature and velocity fields as well as pathogen particles trajectories inside hospital wards to help design comfortable and safe environments (Khan et al. 2011ab).

Khan et al. (2011a) formulated ventilation design as a multi-objective optimisation problem, where the objectives to be minimised were the normalised average thermal comfort $|\overline{T_{res}}|$ and pathogen concentration $|\overline{C}|$ in a monitoring region A defined in a simplified model of a hospital ward. In Khan's model, a hospital room was represented as a two-dimensional cavity, whose walls are maintained at a temperature T_w. Fresh air having temperature T_{in} and velocity u_{in} is introduced in the cavity by an air inlet whose position is fixed. Stale air is extracted by a single air outlet, whose position on the walls of the cavity can be changed acting on the normalised parameter s. The positions of the pathogen source S_ϕ and the monitoring region A are fixed. In Figure 6.2 a schematic diagram of the hospital ward model is shown.

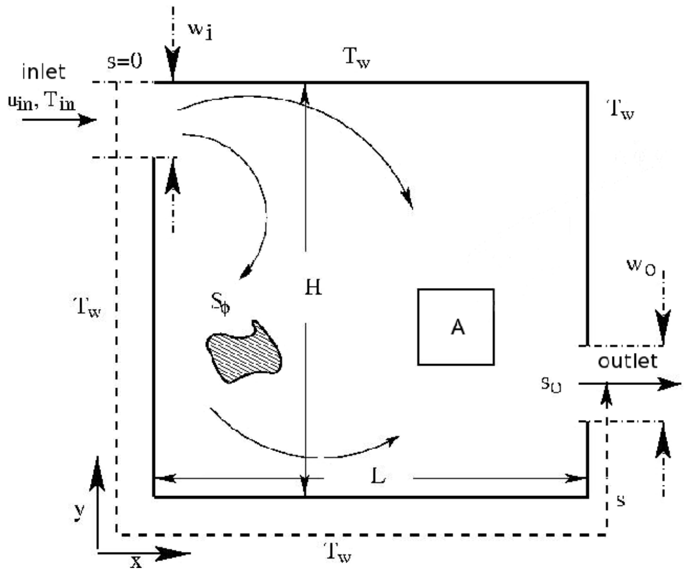


FIGURE 6.2: Two-dimensional model of an hospital ward with pathogen source S_ϕ and monitoring region A

The averaged thermal comfort $|\overline{T_{res}}|$ and pathogen concentration $|\overline{C}|$ in the monitoring region A are assumed to be functions of the air outlet location s. The optimal ventilation design was identified as the one minimising the weighted cost function shown in Eq. (6.1):

$$f(s) = W_C|\overline{C(s)}| + W_{T_{res}}|\overline{T_{res}(s)}| \tag{6.1}$$

$$0.3 \leq s \leq 0.9125 \tag{6.2}$$

where W_C and $W_{T_{res}}$ are weights used to change the relative importance of either objective.

Khan's optimisation problem was approached using HyGP coupled with a GA search algorithm to find the minimum of the cost function defined in Eq. (6.1), with W_C and $W_{T_{res}}$ set both to 0.5. To generate thermal comfort $|\overline{T_{res}}|$ and pathogen concentration $|\overline{C}|$

metamodels, exactly the same data generated by Khan et al. (Khan et al. 2011a) was used, consisting of 45 points uniformly sampled in the range $[0.3, 0.9125]$.

For $|\overline{T_{res}}|$ metamodel generation, HyGP population was set to 400 individuals, imposing a maximum of 200 generations. The primitives used were addition, subtraction, multiplication, division, shift[1], square, cube, sine, cosine, exponential, reciprocal and hyperbolic functions (sinh, cosh, tanh).

As the response to be modelled is a normalised quantity, its range does not exceed the interval $[0, 1]$. Such constraints on the output were exploited using the strategy described in Section 5.4, Chapter 5: a penalisation was introduced for metamodels returning values bigger than 1 on an additional data set C made of 32 points regularly spaced in $[0.3, 0.92]$. A total of 16 independent evolutions were run. The best metamodel found for the normalised, averaged thermal comfort $|\overline{T_{res}}|$ is shown in Equation (6.3):

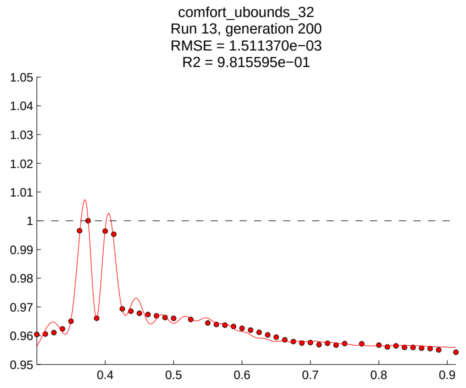
$$\begin{aligned} |\overline{T_{res}(s)}| = 0.955786 - \frac{1}{\left(327.076\,\mathrm{s} - \frac{49.5365}{\mathrm{s}}\right)^2 + 1170605.54229\,\mathrm{s}^9} \times \\ [29.0308\ \cos(34.6155\,\mathrm{s}) - 8.12347\ \cos\left(187.947\,\mathrm{s}^2\right) + \\ 59.2608\ \cos\left(30.0985\,\mathrm{s}^2\right) + \frac{111.213}{\mathrm{s}} - 305.371] \end{aligned} \tag{6.3}$$

Different HyGP settings were used to generate the metamodel of the normalised average pathogen concentration $|\overline{C(s)}|$. A population of 300 individuals was used, the maximum number of generations set to 100. The functional primitives set was identical to the one used for thermal comfort, but division and reciprocal were not used. The penalisation described in Section 5.4, Chapter 5 was introduced for metamodels returning responses outside the feasible region $[0, 1]$: a set of 7 points uniformly distributed in $[0.3, 0.42]$ was used to check the existence of negative responses, whereas another set of 25 points uniformly sampled in $[0.44, 0.92]$ was used to check the existence of responses bigger than 1. In total, 12 independent evolutions were run. The best metamodel found for the normalised, averaged pathogen concentration $|\overline{C(s)}|$ is shown in Equation (6.4):

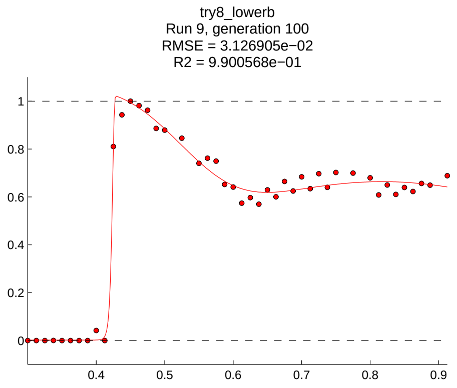
$$\begin{aligned} |\overline{C(s)}| = 0.00194827 - \tanh\left[\left(3.47996769908 \cdot 10^{53}\right)\ \mathrm{s}^{144}\right] \times \\ \left[\tanh\left(18.8286\,\mathrm{s}^5\right) - 4.06459\,\mathrm{s} + 2.48559\,\mathrm{s}^2\right] \end{aligned} \tag{6.4}$$

In Fig. 6.3 the metamodels defined in Eq. (6.3-6.4) are plotted as a function of the normalised position s. The dots represent the building data set.

[1]see Section 5.3.6, Chapter 5 for the advantages related to the use of shift primitive.



(A) Thermal comfort $|\overline{T_{res}(s)}|$ (Eq. (6.3))



(B) Pathogen concentration $|\overline{C(s)}|$ (Eq. (6.4))

FIGURE 6.3: Plots of thermal comfort and pathogen concentration metamodels against normalised outlet position s

The availability of the explicit metamodels for thermal comfort and pathogen concentration eased the analysis of the effect of weights variation on the cost function: Eq. (6.1) is plotted in Fig. 6.4 for different values of W_C and $W_{T_{res}}$.

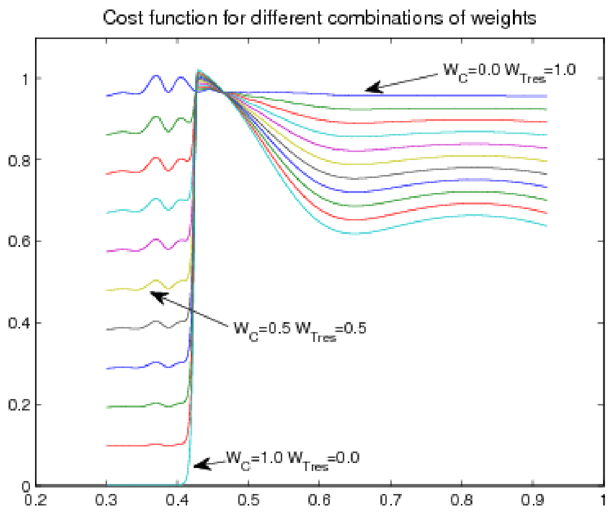


FIGURE 6.4: Cost function for different values of the weights

The minimum of the cost function assembled using the metamodels defined in Eqs. (6.3-6.4) for $W_C = 0.5$ and $W_{Tres} = 0.5$ was found in $s = 0.3$. In Table 6.2 the corresponding value of the cost function is shown and compared to the response returned by CFD validation. In the second row the minimum found by Khan et al. (2011a) using metamodels generated by moving least squares method (MLSM) is reported.

TABLE 6.2: Minima of the cost function $f(s)$ found by different metamodelling techniques

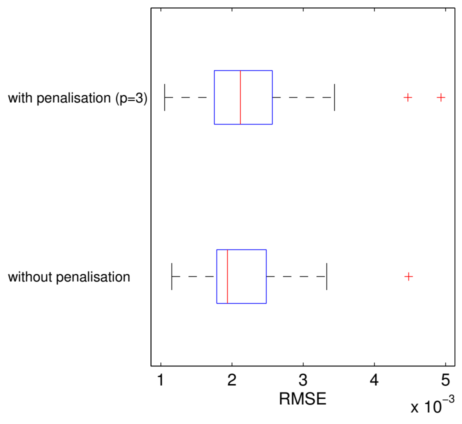
technique	s_{min}	$f(s_{min})$	CFD validation	rel. error
GP + GA	0.3	0.4791136	0.4802253	0.2%
MLSM + GA	0.3	0.4801972	0.4802253	0.006%

The comparison shows that both techniques converged to the same point s_{min} associated to the minimum of the cost function. The metamodels generated by MLSM are characterised by higher accuracy than the ones produced by HyGP. For both cases the validation error is far lower than 1%.

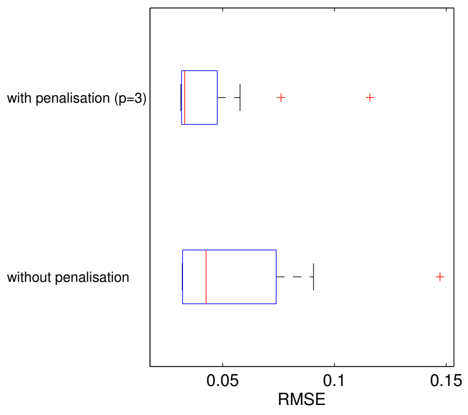
6.3.1 Effect of penalisation on metamodels quality

In Fig. 6.5 the RMSE distributions corresponding to the best metamodels generated for the optimisation problem are compared to the RMSE set resulting from a HyGP experiment where the penalisation of the output was not used. The RMSE refer to the data used for building the metamodels.

The *p*-values returned by Kruskal-Wallis test performed on the RMSE distributions evaluated on the building data set do not provide any evidence of a significant difference in the median due to the effect of the penalisation. The *p*-value returned for thermal comfort experiments is 0.49752. For pathogen concentration, the use of the penalisation slighlty improved metamodels accuracy, although such improvement is not statistically meaningful (Kruskal-Wallis *p*-value = 0.11903).



(A) thermal comfort $|\overline{T_{res}(s)}|$ models



(B) pathogen concentration $|\overline{C(s)}|$ models

FIGURE 6.5: Boxplots representing RMSE distribution for the best models generated with and without the penalisation

A further test was implemented to check whether penalisation contributed to improve the smoothness of the metamodels on unsampled regions of the design space, effect that could not be recognised observing the metamodels' RMSE on the building data set.

As no validation data set was available to evaluate other RMSE distributions[2], the metamodels' generalisation ability was assessed through the sum of the metamodel's response distance d_i from the output feasible region $[0, 1]$, as described in Eqs. (5.35-5.36) in Section 5.4, Chapter 5. The distance d_i was computed on an additional data set D made of 6201 points uniformly distributed in the interval $[0.3, 0.92]$. This sum is proportional to the area between the model and the output feasible region $[0, 1]$ in the selected domain interval.

In table 6.3 the values of the overall distance d_i from the feasible region $[0, 1]$ for each metamodel returned by HyGP for thermal comfort symbolic regression are given. The boxplots in Fig. 6.6 show the different distributions of d_i for thermal comfort metamodels produced using and not using the output penalisation.

TABLE 6.3: Thermal comfort models' distances d_i from feasible region

	without penalisation	with penalisation
run 1	1.2324e+01	6.3746e-01
run 2	5.8301e+01	4.0053e-02
run 3	2.6372e+00	9.4451e-01
run 4	2.3652e+00	5.2473e+00
run 5	1.8600e+00	6.5424e-02
run 6	4.9864e-01	2.3314e+00
run 7	2.7227e+00	1.5467e+00
run 8	3.1799e+00	0.0
run 9	5.4815e-01	1.0189e+00
run 10	4.5035e+00	0.0
run 11	4.0025e-01	2.8616e+01
run 12	2.0535e+02	1.0229e+01
run 13	8.8728e-01	6.4701e-01
run 14	4.7886e+00	4.8797e-01
run 15	0.0	6.3183e-01
run 16	1.3516e+01	2.4373e+00
average	1.9618e+01	3.4300e+00
median	2.6799e+00	7.9576e-01
IQR	7.8386e+00	2.1077e+00
models inside bounds (%)	6.2	12.5
p value	9.3392e-02	

Table 6.4 lists the values of d_i for each bacteria concentration model generated using and not using the penalisation. The boxplots in Fig. 6.6 show the different distributions of d_i for bacteria concnetration metamodels produced using and not using the output penalisation.

[2]for the importance of the validation data set, see Section 3.2.2, Chapter 3 and Section 5.3.1, Chapter 5.

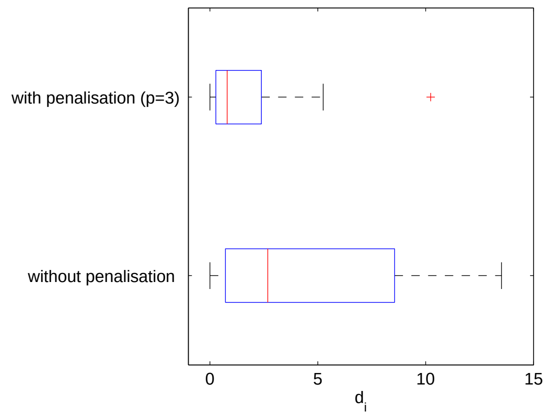


FIGURE 6.6: Boxplots of thermal comfort metamodels' distances d_i from output feasible region

TABLE 6.4: Bacteria concentration metamodels' distances d_i from feasible region

	without penalisation	with penalisation
run 1	9.8925e-01	0.0
run 2	7.3112e-01	6.4714e-01
run 3	1.1504e+01	6.6014e+00
run 4	2.2419e+01	2.0765e+01
run 5	7.3582e-01	8.9439e-01
run 6	7.4186e+00	5.9104e+00
run 7	8.6624e-01	8.2953e+00
run 8	8.9188e-01	8.9392e-01
run 9	4.7552e+00	2.1729e+00
run 10	0.0	2.9932e+00
run 11	4.6822e+01	4.6933e+00
run 12	1.2657e+01	4.2889e+01
average	9.1491e+00	8.0630e+00
median	2.8722e+00	3.8433e+00
IQR	1.1279e+01	6.5542e+00
models inside bounds (%)	8.3	8.3
p value	1.0	

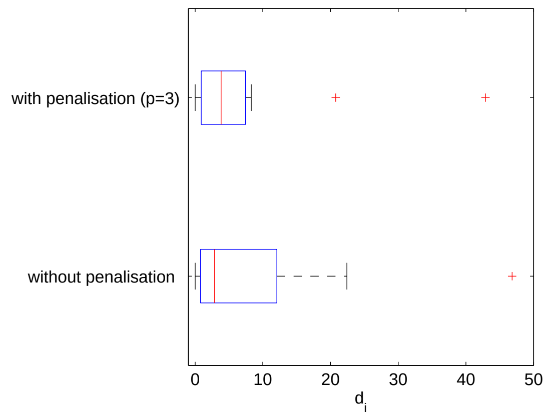


FIGURE 6.7: Boxplots of bacteria concentration metamodels' distances d_i from output feasible region

Combining the results from tables 6.3-6.4 with the observations made at the beginning of the section on the accuracy of the metamodels on the building data set, it emerges that the use of penalisation was beneficial for the symbolic regression, although such conclusion is not confirmed by statistical evidence.

Thermal comfort metamodels generalisation ability was improved, although the penalisation did not influence the accuracy on the building data set. The opposite happened for the pathogen concentration metamodels: the penalisation improved the accuracy on the building data set, whereas generalisation ability was no particularly increased.

The lack of conclusive evidence regarding the effect of the penalisation may be ascribed to the small size (32 points) of the additional data set used for assessing the feasibility of metamodels' output.

6.4 Chromate diffusion model

Chromates (chromium (VI) salts, salts of chromic acid, salts containing the divalent ion, CrO_4^{2-}) are widely used in the aerospace industry for corrosion protection, primarily of aluminium alloys. Although efforts are being made to replace chromates with less harmful alternatives, they are still in widespread use. It is therefore important to understand the role of chromates in corrosion prevention and the mechanisms by which it is achieved.

Corrosion protection schemes for metallic structures, particularly those containing aluminium alloys, imply generally an initial surface treatment (usually involving either chemical conversion or anodising), followed by the application on the resulting oxide layer of a corrosion inhibiting primer containing a soluble chromate salt. If required, a topcoat can be applied to provide additional protection from harsh environments. Typically, chromate salts of heavy metals such as barium chromate ($BaCrO_4$) and strontium chromate ($SrCrO_4$) are used in corrosion inhibiting aerospace primers.

The corrosion protection system is designed to function in the following manner: when a scratch occurs that penetrates the chromate-loaded primer through to the underlying aluminium alloy substrate, the metal is exposed. Contact with an aqueous medium can then trigger the corrosion process leading to the deterioration of the alloy. Soluble chromates present in the primer are able to hinder such process: they dissolve into the aqueous medium and are then transported to the exposed site, where chromium ions react with the aluminium alloy producing a passive layer, arresting further corrosion. Thus, the leaching (or dissolution) of chromate is an important step in corrosion protection process.

Modelling of the leaching process under different conditions may improve understanding of corrosion protection and may lead to a reduction in the amount of testing required for the qualification of new corrosion protection systems.

Research has shown clear time dependency of chromate leaching (Prosek and Thierry 2004, Xia 2000, Scholes et al. 2006): generally, leach rates decrease with time. It has also been reported that the pH of the aqueous medium that comes into contact with the primer has an influence on leaching. High leach rates are found at low pH (Furman et al. 2006). Solubility of chromates has been regarded as not being a controlling factor as the concentration of leached chromate dissolved in the immersion medium is low compared to that of a saturated solution (Xia 2000).

The model proposed by Furman et al. (Furman et al. 2006) defines the mass of released material per surface unit (M_t, measured in mg/dm^2) as proportional to a power of time (the model was found exploiting Fick's second law of diffusion):

$$M_t = kD_{eff}t^n \tag{6.5}$$

in which D_{eff} is the effective diffusion coefficient, k is a constant and t^n is a power of time. Furman et al. proposed to use a constant value of 0.25 for the exponent n of the time variable for all primers under various conditions, letting the effective diffusion coefficient D_{eff} be the only varying parameter in the model described in Eq. (6.5). However, experiments performed on four primers (not described here), whose results are shown in Fig. 6.8, show that the validity of the Furman's simplified model is rather limited. Tests proved that the effective diffusion coefficient D_{eff} has a different value for each primer, and the power of time n cannot be assumed to be equal for all primers either. For example, a basic power fit of the form ct^n to the data plotted in Fig. 6.8 results in different values of the exponent n for primers Aerodur HS 37092 ($n = 0.33$) and Seevenax 313-01 ($n = 0.12$). This proves that a model using a constant power of time is not appropriate for modelling the leaching process and new models are needed.

6.4.1 Methodology

The aim of the research activity, conducted in collaboration with an aerospace company, was to provide mathematical models of the quantity of chromate salts dissolving into an

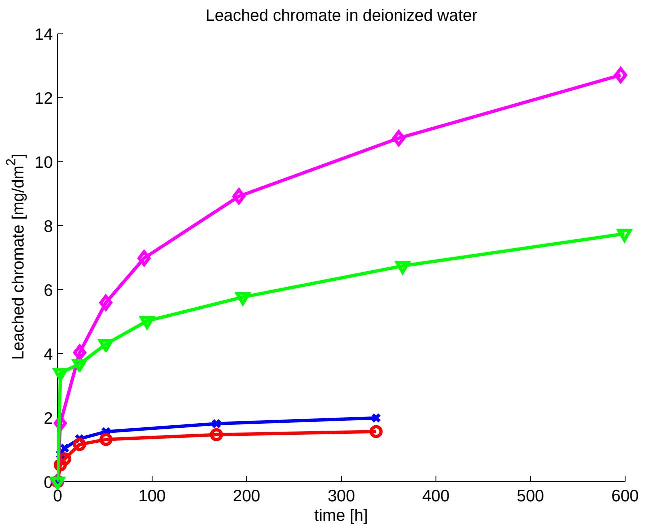


FIGURE 6.8: Leaching curves of different primers in deionised water. The mass of chromate leached per surface unit of the aluminium sample is plotted against time for different primers: $\diamond$ = Aerodur HS 37092, $\triangledown$ = F580-2080, $\times$ = Seevenax 313-01, $\circ$ = Mapaero P60-A

aqueous solution from an aluminium alloy sample treated with a chromate-loaded primer. The study focused on the behaviour of three different primers, which will be referred to as A, B and C[3] in the following.

The generation of the models was only the final stage (step 3) of a process that consisted of the following steps:

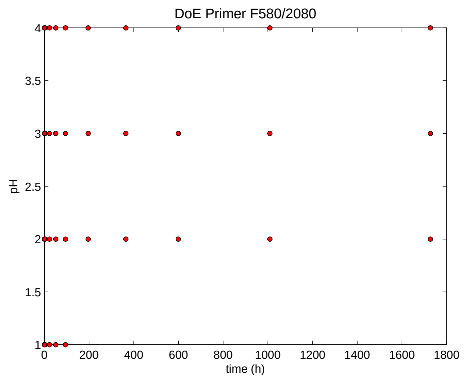
1. preparatory testing
2. chromate leaching measurement
3. model generation through genetic programming

The first two steps, initial testing and chromate leaching measurement, were carried out independently by Mr. D. J. Boon, Dr. L. J. Clarke and Dr. M. B. Stowe in the labs of the aerospace company and the gathered data were kindly provided for the metamodelling activity.

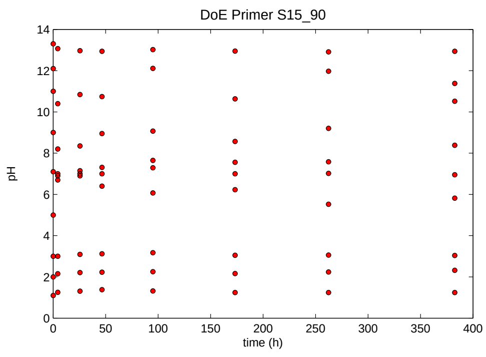
During the preparatory stage a sensitivity analysis of the leaching process was carried out to identify the main variables involved. Tests showed that the temperature of the aqueous solution where the treated alloy sample was immersed did not have a significant effect on the leaching over a range of 9°C to 50°C and for a time period of up to two months. As a result, time and pH were identified as the independent variables to be

[3]The correspondent commercial names are F580-2080 for primer A, Aerodur®S15/90 for primer B, Epoxy Primer 37032A for primer C.

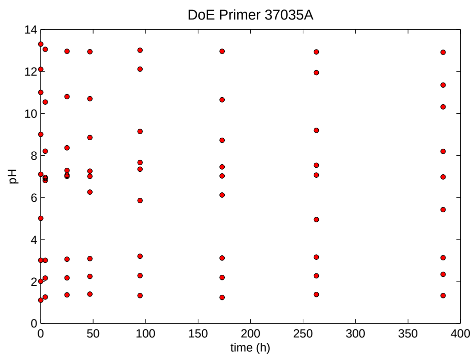
used in the design of experiments. In total, 35 sample points were defined for primer A, whereas 72 sample points were used for primers B and C. The plans of experiments used for each primer are shown in Fig. 6.9.



(A) DoE for primer A



(B) DoE for primer B



(C) DoE for primer C

FIGURE 6.9: DoEs for primers A, B and C

In the second stage, the coated aluminium samples were immersed in an aqueous solution of 300 ml and the total quantity of chromate dissolved into the solution measured at different times and for different initial pH values of the solution, according to the design of experiments for each primer. Over a period of 2 months, 20-ml samples were removed from the medium and chromate content and pH of the samples measured. The chromate concentration in the sample was measured through Atomic Absorption Spectrometry, using an AAnalyst 400 Spectrometer from PerkinElmer, and the total mass of chromate in the solution obtained multiplying the concentration by the volume of the solution. The total volume of the solution was then made back up to 300 ml by adding 20 ml of fresh medium. If the pH of the solution had drifted from the required value, the 20-ml sample that was added to the solution was also used to adjust the pH of the solution back to the desired pH. Because the rate of leaching reduces with time, measurements were taken more frequently at the beginning of the experiment than at the end, as it is shown in the design of experiments for the three primers in Fig. 6.9.

In the third and final stage, data obtained by the measurements on the samples were processed by HyGP to generate empirical models for the quantity of dissolved chromate as a function of time and pH.

Ten independent runs for each input data set were launched to increase the chances of finding acceptable models. A constant population of 200 individuals and 100 generations were used in all the experiments. The functions (primitives) used were the standard algebraic operations (addition, subtraction, multiplication and division) as well as power, sine, cosine, logarithm, reciprocal and the hyperbolic functions. Shift unary operator was also used[4].

6.4.2 Results

In the following paragraphs the best models generated for each primer are described. pH is represented by pH; time is measured in hours, and represented by the letter t.

6.4.2.1 Primer A

The best model found according to the sheer root mean square error on the building data set is reported in Eq. (6.6):

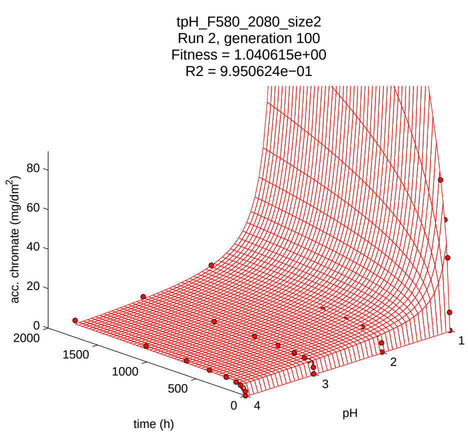
$$f(t,pH) = 2.45694 \cdot 10^{-3}\,\mathrm{t} + \frac{462.344\,\mathrm{t}}{99.2640\,\mathrm{t} - \frac{95.1374\,\mathrm{t}}{\mathrm{pH}} + 203.436} - 0.190638 \tag{6.6}$$

Its coefficient of determination is R^2 = 0.9958987, the maximum error -2.466175 mg/dm^2. However, the most significant model generated is described in Eq. (6.7):

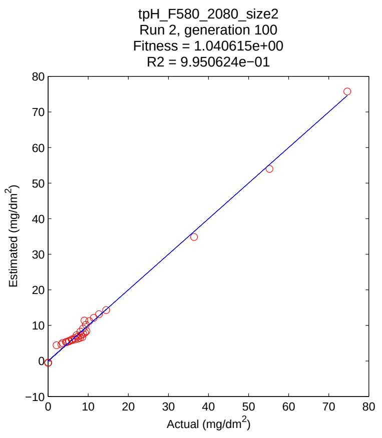
$$f(t,pH) = 4.41652\,(1.91811\,\mathrm{t})^{\frac{1}{4.87395\,\mathrm{pH}-3.04801}} - 0.498776 \tag{6.7}$$

the model in Eq. (6.7) has a coefficient of determination (R^2 = 0.995062) slightly lower than the previous one's. The maximum error, 2.341443 mg/dm^2, is however smaller. The corresponding plot is shown in Fig. 6.10, together with a graphical comparison of the estimated response against the measured (or actual) response. The particular interest in Eq. (6.7) comes from the striking resemblance to the model proposed by Furman et al., described in the introduction (Eq. (6.5)).

[4]see Section 5.3.0.4, Chapter 5



(A) Leached chromate as a function of time and pH



(B) Actual vs. estimated response

FIGURE 6.10: Generated model for primer A

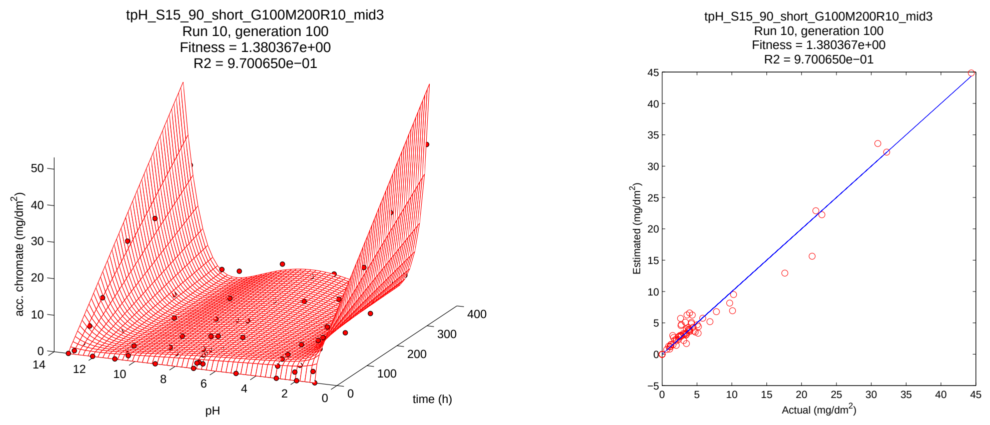
6.4.2.2 Primer B

For primer B HyGP was not able to generate models as simple as the ones found for primer A (Eqs. (6.6-6.7)). The extended pH range or the noise in measurements may have forced the algorithm to increase the average size of the mathematical expressions to increase the accuracy. In other words, the models are generally affected by “bloat”, which is a well-known behaviour for genetic programming techniques, as seen in Section 4.4.1, Chapter 3.

The best model found according to the root mean square error on the building data set was:

$$\begin{aligned} f(t,pH) =& 0.024784\,\mathrm{t\,pH} \\ & -\tanh(0.125306\,\mathrm{t})\;(0.0898783\,\mathrm{t} + 0.36776\,\mathrm{pH} - 5.1529) \\ & -3.18441\cdot 10^{-3}\,\mathrm{pH} + \frac{0.25614\,\mathrm{t}}{\mathrm{pH}^2} - 1.61642\cdot 10^{-3}\,\mathrm{t\,pH}^2 \\ & + \left(3.39314\cdot 10^{-17}\right)\,\mathrm{t\,pH}^{14} + 0.00567243 \end{aligned} \tag{6.8}$$

Although complex, the model shows a high coefficient of determination (R^2 = 0.970065). The maximum error is -5.889410 mg/dm^2. The corresponding plot is shown in Fig. 6.11.



(A) Leached chromate as a function of time and pH

(B) Actual vs. estimated response

FIGURE 6.11: Generated model for primer B

6.4.2.3 Primer C

The models found for primer C were neither compact not easily interpretable. On average, models were larger than in the first case (primer A), being mainly linear combination of non-linear terms.

The model having the best root mean square error was:

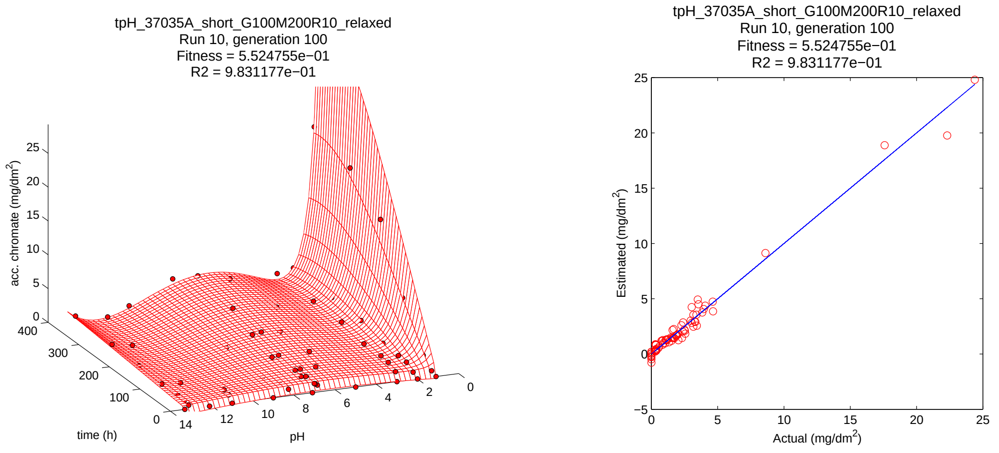
$$\begin{aligned} f(t,pH) =\, & 0.243435\,\mathrm{pH} - 6.47727\cdot 10^{-3}\,\mathrm{t} + \tanh(107.85\,\mathrm{t}) \\ & - \frac{5.14483\cdot 10^{-4}\left(210.141\,\mathrm{t} - \frac{569.132\,\mathrm{t}}{\mathrm{pH}} + \frac{1.56793\cdot 10^{-6}\,\mathrm{t}^4}{\mathrm{pH}}\right)}{\mathrm{pH}} \\ & + 7.08876\cdot 10^{-3}\,\mathrm{t\,pH} - 8.25987\cdot 10^{-5}\,\mathrm{t\,pH}^3 + 2.82282\cdot 10^{-7}\,\mathrm{t\,pH}^5 \\ & - 1.97306\cdot 10^{-2}\,\mathrm{pH}^2 - 0.527521 \end{aligned} \tag{6.9}$$

The corresponding coefficient of determination is relatively high $R^2 = 0.983118$. The maximum error is -2.542556 mg/dm^2. The plot of the model is shown in Fig. 6.12.

6.4.3 Discussion of results

HyGP has shown the ability to produce high quality models for the three data sets provided by the experiments.

The explicit mathematical expressions shown in Eqs. (6.7-6.8-6.9) ease the interpretation of the data produced by the experiments on the treated aluminium samples. With regards to the influence of pH value on the leaching, high chromate releases are highlighted at low pH ($pH < 2$) for all three primers, whereas for primer B high releases are



(A) Leached chromate as a function of time and pH

(B) Actual vs. estimated response

FIGURE 6.12: Generated model for primer C

detected even for high pH ($pH > 12$). A local maximum is detected near neutral pH for primers B and C. During the experiments on the aluminium samples it was interestingly noted that the pH of solutions having a not neutral pH ($pH \neq 7$) had a tendency to move towards neutral. This behaviour, noticed also by Furman et al. (Scholes et al. 2006), indicates that there are reactions going on between elements from the primer and the acidic or alkaline solution. According to Kondratenko et al. (Kondratenko and Sherstyuk 1986) the drift in pH is the result of four equilibrium reactions going on between the acidic or alkaline solution and chromates.

The general trend of the leaching rate with respect to time has already been discussed in the opening of the section (see Fig. 6.8) and the models generated by HyGP conform with it. As shown in Fig. 6.10, 6.11 and 6.12 the leaching is initially high and slows down with time. Such reduction in the leaching rate can be explained considering the different times needed by the chromates to diffuse out of the primer, according to their position in the primer film: the chromates near the surface of the primer dissolve rapidly, whereas the chromates sited below the surface need more time to diffuse outwards. Diffusion of chromates out of the primer plays then an important role in corrosion protection, as the efficacy of the corrosion protection process depends on the quantity of chromate leached.

Focusing instead on the method used to generate the models, the results of the previous paragraph show HyGP ability to extend or generalise existing chromate leaching models. The model in Eq. (6.7) is in fact a generalisation of the model found by S.A. Furman et al. (Furman et al. 2006) presented in Eq. (6.5), as not only time but also pH

is included in the model. Where simple and interpretable expressions were not found, as for the cases of primers B and C, HyGP still provided high-quality models.

6.5 Modelling of supersonic jet pump entrained flow rate

Supersonic jet pumps are devices that are able to pump a flow without the need for moving parts. The flow pressure variation is achieved mixing the flow (also called *entrained flow*) with a high velocity jet (or *primary flow*) and making them go through a duct of variable cross-sectional area (Eves et al. 2011). As a result of the lack of moving parts, supersonic jet pumps boast a far longer life compared to other pump typologies and for this reason they can be considered environmentally friendly devices. Their application is common in refrigeration to desalination industry (Eves et al. 2011).

Eves et al. (2011) applied a formal optimisation framework to the optimisation of a supersonic jet pump design. The entrained flow rate $\dot{V}_{ent}$ was maximised for a broad range of primary flow rates $\dot{V}_{pri}$ (from 200 L/min to 1200 L/min). For the application of the formal optimisation framework (described in Section 6.1), the problem was reformulated as follows:

$$maximise : \dot{V}_{ent} \tag{6.10}$$

$$subject\,to : \dot{V}_{pri} \leq c_0 \tag{6.11}$$

$$c_i^L \leq DV_i \leq c_i^U \quad i = 1, 2, 3 \tag{6.12}$$

the entrained flow rate $\dot{V}_{ent}$ and the primary flow rates $\dot{V}_{pri}$ were assumed function of the independent variables $DV1$, $DV2$, $DV3$ defined as:

$$DV1 = R_N \tag{6.13}$$

$$DV2 = \frac{R_{DI}}{R_N} \tag{6.14}$$

$$DV3 = \frac{R_{DO}}{R_{DI}} \tag{6.15}$$

where R_N is the jet pump nozzle radius, R_{DI} the diffuser inlet radius and R_{DO} the diffuser outlet radius. The input variables $DV2$ and $DV3$ were defined as ratios of physical jet pump parameters to avoid unfeasible designs. CFD simulations provided the values of the objective $\dot{V}_{ent}$ and the constraint $\dot{V}_{pri}$ for each point of a Latin Hypercube DoE made of 100 points. MLSM was used to generate metamodels of the entrained and primary flow rate and GA and SQP algorithms were used to search for the solution of the optimisation problem defined in Eqs. (6.10-6.11-6.12).

6.5.1 Entrained flow modelling using HyGP

The idea to use HyGP to model the jet pump entrained flow rate sparked from a practical need: the final user of the jet pump model required a simple tool to evaluate the jet pump performance for a given set of design parameters. Genetic programming models are returned as explicit text expressions that can be evaluated by spreadsheet software, so HyGP was in this case considered the optimal modelling tool. The request of the jet pump user then gave the opportunity to assess HyGP applicability to industrial problems, as well as confirming that accuracy is not the only criterium that guides the selection of a metamodelling technique.

The modelling stage was repeated using HyGP instead of MLSM on the already existing data kindly provided by Dr. Eves. Model generation and validation were performed on the existing building and validation data sets, respectively made of 136 and 57 points sampled in the following region:

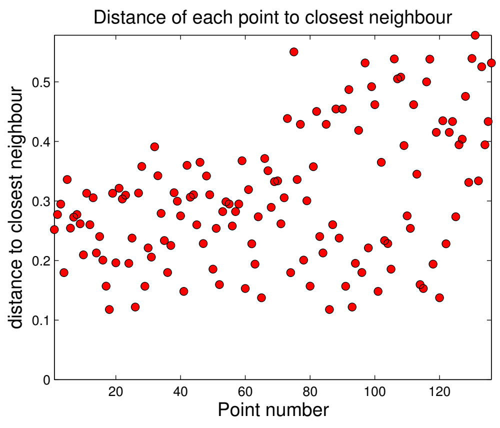
$$1.0\,mm \leq DV1 \leq 3.38\,mm \tag{6.16}$$

$$2.0 \leq DV2 \leq 3.5 \tag{6.17}$$

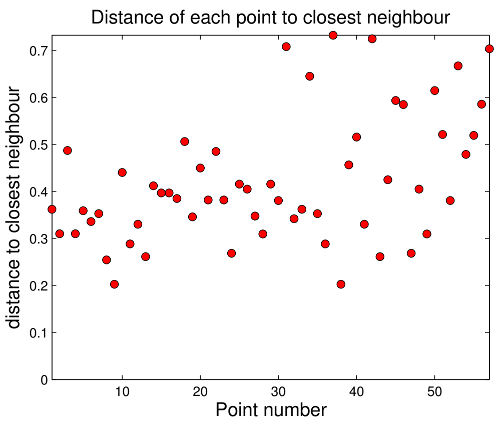
$$1.5 \leq DV3 \leq 4.5 \tag{6.18}$$

The minimum distance between DoE points in the building and validation data sets are shown in Fig. 6.13. For each point the corresponding entrained flow rate had been previously computed by CFD simulations.

10 HyGP independent runs were performed, setting the population size to 200 individuals and the maximum number of generations to 50. The other parameters are reported



(A) Building data set



(B) Validation data set

FIGURE 6.13: Minimum distance bewteen DoE points for building and validation data sets

in Table C.15 in Appendix C. The best model found is reported in Eq. (6.19):

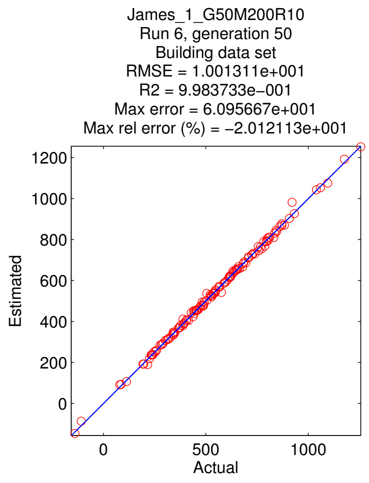
$$
\begin{aligned}
\dot{V}_{ent}(DV1, DV2, DV3) = {} & 395.18\,\mathrm{DV1} - 602.739\,\mathrm{DV2} - 764.272\,\mathrm{DV3} \\
& + 969.842\,\mathrm{DV1}\,\mathrm{DV2} + 169.974\,\mathrm{DV2}\,\mathrm{DV3} - \frac{2892.42\,\mathrm{DV1}}{\mathrm{DV2}} + \frac{160.407\,\mathrm{DV2}}{\mathrm{DV1}} \\
& - 300.893\,\mathrm{DV1}^2\,\mathrm{DV2} - 9.69909\,\mathrm{DV1}^3\,\mathrm{DV2} - \frac{62.4153\,\mathrm{DV1}}{\mathrm{DV3}} + \frac{924.589\,\mathrm{DV3}}{\mathrm{DV2}} \\
& + 639.652\,\mathrm{DV1}^2 - 5.50577\,\mathrm{DV1}^2\,\mathrm{DV2}^2 + \frac{980.552}{\mathrm{DV1}\,\mathrm{DV3}} + \frac{439.244\,\mathrm{DV1}^2}{\mathrm{DV3}} \\
& + \frac{34.079\,\mathrm{DV1}^3}{\mathrm{DV3}} + 0.528353\,\mathrm{DV1}\,\mathrm{DV2}^3\,\mathrm{DV3}^2 - \frac{737.5\,\mathrm{DV1}^2}{\mathrm{DV2}\,\mathrm{DV3}} - \frac{171.104\,\mathrm{DV2}^2}{\mathrm{DV1}^2\,\mathrm{DV3}} \\
& + \frac{81.8517\,\mathrm{DV2}^2}{\mathrm{DV1}^3\,\mathrm{DV3}} s - \frac{304.829\,\mathrm{DV1}\,\mathrm{DV2}}{\mathrm{DV3}} - 7.1222\,\mathrm{DV1}\,\mathrm{DV2}^3\,\mathrm{DV3} + 36.2463
\end{aligned}
\tag{6.19}
$$

The RMSE, coefficient of determination R^2 and the maximum relative error are provided in Table 6.5. The metamodel features good generalisation properties as the RMSE and R^2 on building and validation data sets are comparable. In Fig. 6.14 the estimated entrained flow rate is plotted against the actual flow rate computed by CFD for each sample in the building and validation data sets: the points closeness to the line reflects the high accuracy of the metamodel on both data sets.

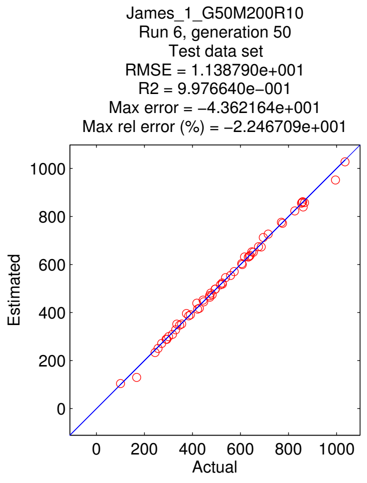
TABLE 6.5: Entrained flow rate metamodels quality on building and validation data sets

	Building	Validation
RMSE	1.00131e+01	1.13879e+01
R^2	0.998373	0.997664
Max relative error (%)	2.01211e+01	2.24671e+01

The accuracy of the model was considered acceptable for industrial use, and the availability of the explicit expression in Eq. (6.19) made metamodel exploitation easier. For



(A) Best model response on building data set



(B) Best model response on validation data set

FIGURE 6.14: Model response versus actual response. Each point represent an entrained flow rate in L/min

example, the plot in Fig. 6.15 showing the dependency of the entrained flow rate on the three input variables was generated using the expression returned by HyGP. In the figure it can be clearly seen that the entrained flow rate maximum lies approximately in the region centered in $\{3.3\ 2.4\ 1.5\}$, observation that is consistent with the results reported by Eves et al. (2011).

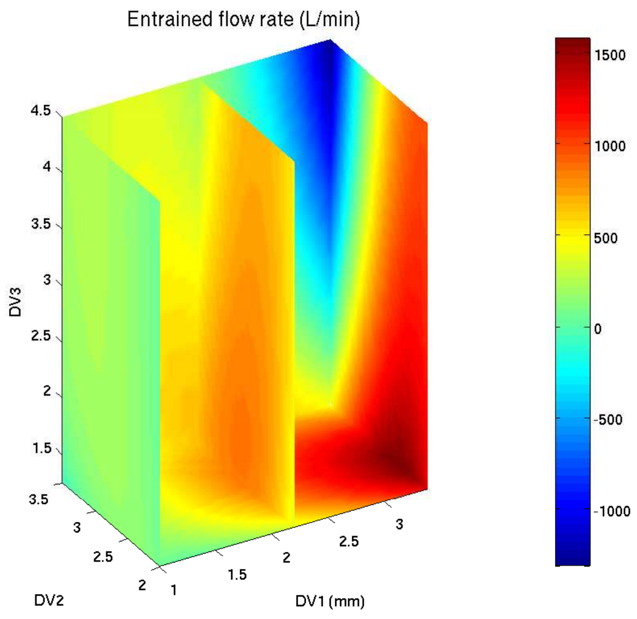


FIGURE 6.15: Entrained flow rate (L/min) as a function of DV1, DV2, DV3

6.6 Bread baking oven design optimisation

Commercial bread-baking industry has been showing increasing interest in computational fluid dynamics (CFD) to model the complex mass and heat transfer processes involved in bread baking, with the aim of reducing cooking times without compromising bread quality.

In forced convection ovens (also called "direct fired" ovens), a common oven typology, hot air is injected in the baking chamber by nozzles located on the ceiling and the floor of the chamber. Bread loaves are placed on a tray located at equal distance H from the ceiling and the floor, as shown schematically in Fig. 6.16.

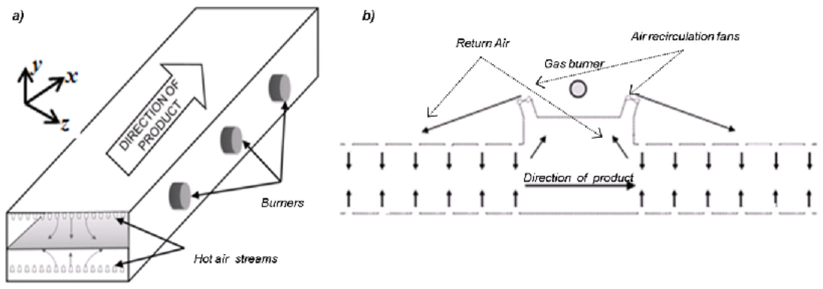


FIGURE 6.16: Three-zone direct fired oven: *a)* Overview of the oven; *b)* Simplified schematic showing the oven longitudinal section and the mechanism for distributing air through the nozzles on the ceiling and on the floor

Air temperature uniformity in the baking chamber is a key factor to ensure uniform heat transfer and then product quality. Following Khatir et al. (2011ab), temperature uniformity in an oven baking chamber can be defined by the parameter σ_T defined in Eq. (6.20):

$$\sigma_T = \sqrt{\frac{\int_V (T_i - T_{zone})^2 dV}{\int_V dV}} \tag{6.20}$$

where T_{Zone} is the temperature of the air flowing through the nozzles fitted on the baking chamber walls, V is the baking domain and T_i is the air temperature at point i of the baking chamber.

By the definition in Eq. (6.20), the optimisation of the oven design consists in finding the set of design parameters that minimises the root mean square temperature variation σ_T.

6.6.1 Methodology

In mathematical terms, the oven design optimisation implies finding a global minimum of the function σ_T. HyGP was used to generate a global metamodel of σ_T: due to oven simmetry, σ_T was modelled only on a portion of the baking chamber. Three parameters defining oven and nozzles characteristics were considered as independent variables, as done in Khatir et al. (2011b): the nozzle jet diameter (D), the dimensionless nozzle-to-surface distance ($\frac{H}{D}$) and the nozzle jet velocity (u_{noz}). The CFD model of the oven baking chamber and the input variables are described in Fig. 6.17.

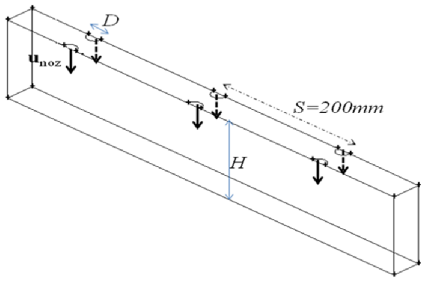


FIGURE 6.17: CFD baking chamber model with design variables: nozzle jet diameter D, jet velocity u_{noz} and distance H between tray surface and chamber ceiling (nozzle to nozzle distance $S = 200$ mm)

CFD simulations provided σ_T values on a Latin hypercube DoE made of 30 points, used as building data set. As by definition σ_T is positive in the whole domain, such knowledge was exploited to improve HyGP symbolic regression through the use of the penalisation introduced in Section 5.4, Chapter 5. The sign of σ_T was checked on an additional data set C made of 120 points generated using a full factorial DoE in the region $[5, 20] \times [2, 10] \times [8, 40]$. For the HyGP experiment, 12 independent evolutions were performed, each using a population of 300 individuals for a maximum number of 200 generations. As functional primitives addition, subtraction, multiplication, power, shift, square, cube, sine, cosine, hyperbolic sine, hyperbolic cosine, hyperbolic tangent, exponential were selected.

The best metamodel found by HyGP is shown in Eq. (6.21):

$$
\begin{aligned}
\sigma_T(D, \tfrac{H}{D}, u_{noz}) &= 6.54966 \times 10^{-6}\,\mathrm{D}^3\,\mathrm{u_{noz}} + 4.26492 \times 10^{-6}\,\mathrm{D}^2\,\mathrm{u_{noz}}^2 \\
&- 1.84683 \times 10^{-4}\,\mathrm{D}\left(\tfrac{\mathrm{H}}{\mathrm{D}}\right)^2 \mathrm{u_{noz}} + 2.69398 \times 10^{-5}\,\mathrm{D}\left(\tfrac{\mathrm{H}}{\mathrm{D}}\right) \mathrm{u_{noz}}^2 \\
&+ 4.97275 \times 10^{-2}\,\mathrm{D}\left(\tfrac{\mathrm{H}}{\mathrm{D}}\right) - 6.163 \times 10^{-1}\,\mathrm{D} + 5.77163 \times 10^{-6}\left(\tfrac{\mathrm{H}}{\mathrm{D}}\right)^5 \mathrm{u_{noz}}^2 \\
&+ 2.13294 \times 10^{-8}\left(\tfrac{\mathrm{H}}{\mathrm{D}}\right)^4 \mathrm{u_{noz}}^4 - 1.68813 \times 10^{-4}\left(\tfrac{\mathrm{H}}{\mathrm{D}}\right)^4 \mathrm{u_{noz}}^2 \\
&- 1.07612 \times 10^{-2}\left(\tfrac{\mathrm{H}}{\mathrm{D}}\right)^4 - 1.76832 \times 10^{-7}\left(\tfrac{\mathrm{H}}{\mathrm{D}}\right)^3 \mathrm{u_{noz}}^4 \\
&+ 9.4134 \times 10^{-4}\left(\tfrac{\mathrm{H}}{\mathrm{D}}\right)^3 \mathrm{u_{noz}}^2 + 2.83469 \times 10^{-2}\left(\tfrac{\mathrm{H}}{\mathrm{D}}\right)^3 \mathrm{u_{noz}} \\
&+ 5.52956 \times 10^{-4}\left(\tfrac{\mathrm{H}}{\mathrm{D}}\right)^2 \mathrm{u_{noz}}^2 - 2.96942 \times 10^{-1}\left(\tfrac{\mathrm{H}}{\mathrm{D}}\right)^2 \mathrm{u_{noz}} \\
&+ 5.81804 \times 10^{-1}\left(\tfrac{\mathrm{H}}{\mathrm{D}}\right)^2 + 5.57406 \times 10^{-1}\left(\tfrac{\mathrm{H}}{\mathrm{D}}\right) \mathrm{u_{noz}} + 1.234\left(\tfrac{\mathrm{H}}{\mathrm{D}}\right) \\
&+ 7.574 \times 10^{-7}\,\mathrm{u_{noz}}^4 + 5.78694 \times 10^{-4}\,\mathrm{u_{noz}}^3 - 4.88598 \times 10^{-2}\,\mathrm{u_{noz}}^2 \\
&+ 5.79625 \times 10^{-1}\,\mathrm{u_{noz}} + 2.86242
\end{aligned}
\tag{6.21}
$$

Although the generated metamodel shown above is quite accurate on the building data set (RMSE = 4.62702e-02, R^2 = 0.999296), its response was found to be positive only inside the hypercube $[5, 20] \times [2.75, 8] \times [8, 39.7]$. This may be explained considering that the fitness function as defined in Eq. (5.33)[5] gathers conflicting objectives: RMSE minimisation on building data set may hinder the minimisation of the distance d_i of the metamodel responses from the feasible region. An attempt to visualise the behaviour of the metamodel at the boundaries of the hypercube $[5, 20] \times [2.75, 8] \times [8, 39.7]$ is shown in Fig. 6.18a.

6.6.2 Results of the optimisation

In the second stage of the analysis, a GA was used to find the minimum of the σ_T metamodel as defined in Eq. (6.21) in the hypercube $[5, 20] \times [2.75, 8] \times [8, 39.7]$. In Table 6.6 the minimum found is compared with the minimum of another σ_T metamodel generated by MLSM using the same input data. The values returned by the metamodels were validated with the responses returned by CFD simulations in the minima locations (Khatir et al. 2011b), reported in fourth column.

The estimates provided by HyGP and MLSM metamodels are affected by relative errors of the same order of magnitude (see fifth column in Table 6.6). The small size of the building data set may have been the main cause of the poor accuracy of the metamodels.

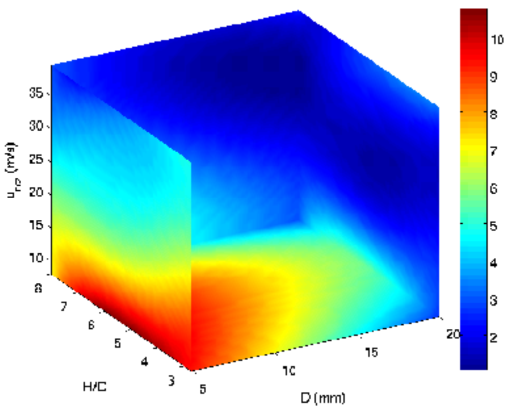
[5]see Section 5.4, Chapter 5.

TABLE 6.6: Minima found using metamodels generated by GP and MLSM

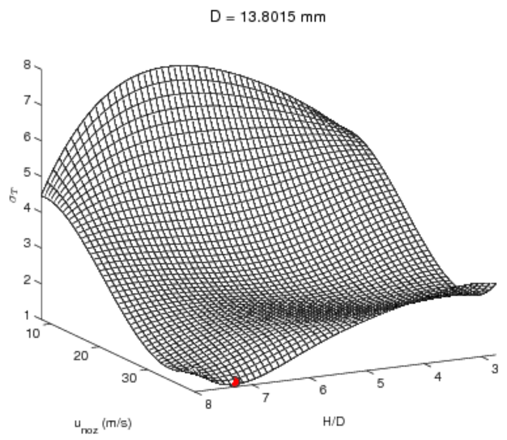
technique	$\{D, H/D, u_{noz}\}$ { [mm], [ ], [m/s] }	σ_T [K]	σ_T from CFD [K]	rel. error %
MLS + GA	{20.00, 6.82, 38.12}	1.16	1.22	4.9%
HyGP + GA	{13.80, 7.31, 39.70}	1.07	1.14	6.1%

Despite their lower accuracy, HyGP metamodel helped discover an optimal design characterised by a value of σ_T (1.14) lower than the one corresponding to the optimum found on MLSM metamodels (1.22). Moreover, the metamodel explicit expression can be easily used to perform Monte Carlo based sensitivity analysis.

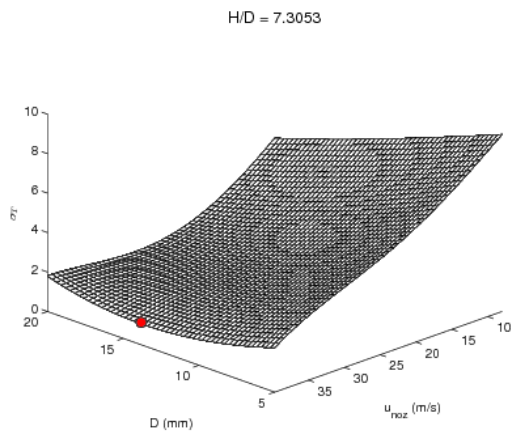
The σ_T metamodel defined in Eq. (6.21) is shown in Fig. 6.18b, 6.18c, 6.18d setting each variable to a constant value: the dot in the figures represents the minimum found by GA.



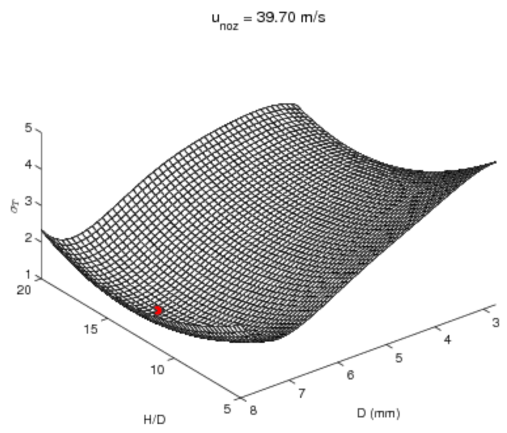
(A) σ_T on boundaries



(B) $\sigma_T(H/D, u_{noz})$ in D = 13.80 mm



(C) $\sigma_T(D, u_{noz})$ in H/D = 7.31



(D) $\sigma_T(D, H/D)$ in u_{noz} = 39.70 m/s

FIGURE 6.18: Behaviour of σ_T metamodel (Eq. (6.21)) on the boundaries of the hypercube $[5, 20] \times [2.75, 8] \times [8, 39.7]$ and around the minimum

6.6.3 Effect of the penalisation

As done for the hospital ward ventilation optimisation (see Section 6.3.1), the effect of the penalisation on HyGP models was assessed comparing the experiment described in the previous section with an identical experiment where instead the penalisation was not used.

In Fig. 6.19 are shown the distributions of RMSE and R^2 of the best models generated by each run of these two experiments, computed on the building data set.

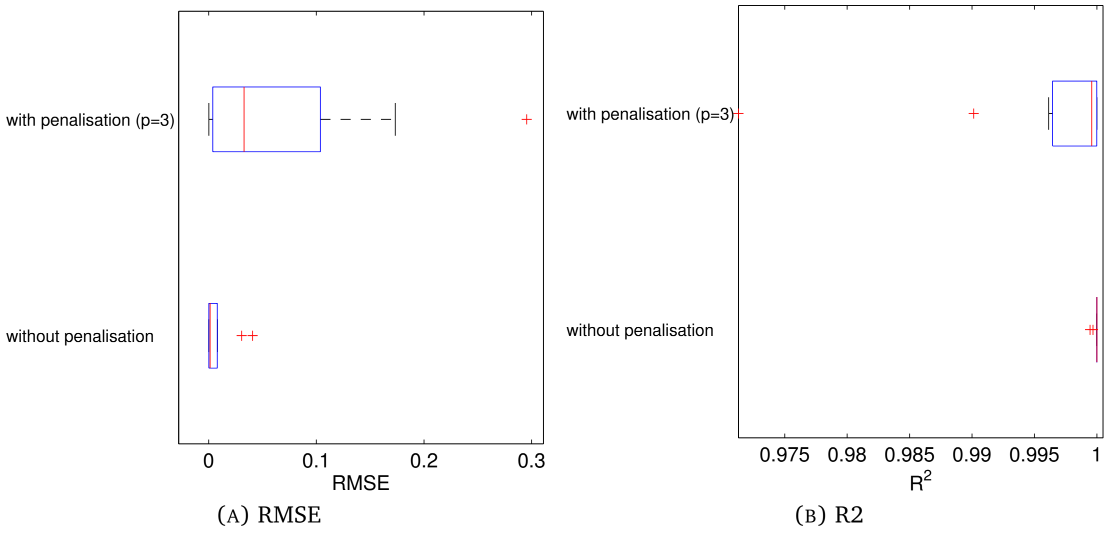


FIGURE 6.19: Boxplots representing RMSE and R^2 distribution on building data set of the best models generated with and without the penalisation

According to Kruskal-Wallis test (p-value 0.0055836), the penalisation seems to have consistently worsened the accuracy of the models, rather than improving it. Such conclusion can be however misleading, as it refers only to the models' behaviour on the building data set, not taking into account possible overfitting issues. In fact, the introduction of the penalisation may increase generated models' average error on the building data set as an additional objective, output feasibility, has to be satisfied on unsampled regions of the design space.

So, a thorough assessment of the generalisation ability was carried out following the methodology described in Section 6.3.1. The sum of the distances of the models responses from the feasible region, d_i, was computed on a full factorial DoE made of 8000 points sampled in the original design space $[5, 20] \times [2, 10] \times [8, 40]$. The overall distances d_i for each model generated with and without the penalisation are reported in Table 6.7. Boxplots in Fig. 6.20 show the distributions of d_i.

TABLE 6.7: σ_T models' distances d_i from feasible region and corresponding statistics

	without penalisation	with penalisation
run 1	2.7051e+03	6.8186e+02
run 2	2.2827e+01	5.3148e+02
run 3	1.1712e+03	1.0375e+03
run 4	6.7022e+02	4.9405e+02
run 5	9.6636e+02	2.0205e+02
run 6	2.9234e+02	3.2230e+02
run 7	7.8744e+02	4.1522e+02
run 8	5.9548e+02	1.6276e+02
run 9	1.1573e+03	6.4371e+01
run 10	8.7899e+02	3.8935e+02
run 11	7.8389e+02	9.9915e+04
run 12	6.5040e+02	8.0844e+02
average	8.9013e+02	8.7520e+03
median	7.8566e+02	4.5463e+02
IQR	4.3887e+02	4.8297e+02
p value	0.14096	
models inside bounds (%)	0	0

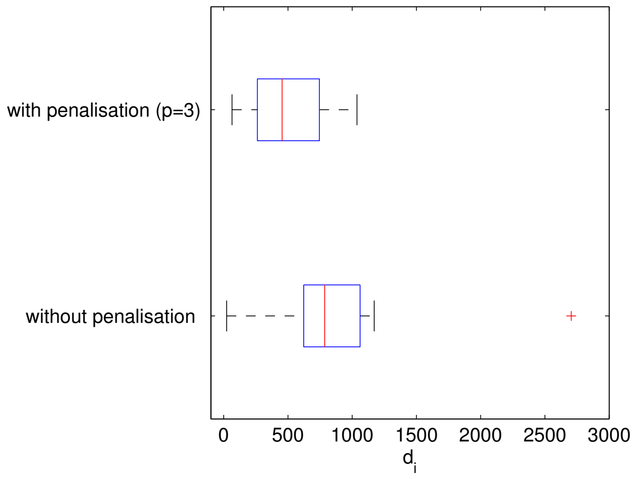


FIGURE 6.20: Boxplots of σ_T metamodels' distances d_i from output feasible region

From Table 6.7 and Fig. 6.20 it emerges that the use of penalisation affected the generalisation property of the models, producing a decrease in the median of the sum of the distances. Such conclusion is however not supported by Wilcoxon rank sum test (p-value = 0.1406).

6.7 Parametric optimisation of a lattice aircraft fuselage barrel

In the EU FP7 collaborative research programme ALaSCA (Advanced Lattice Structures for Composite Airframes) (Ala 2010), the potential of anisogrid structures for a composite fuselage section has been investigated and a developed fuselage design has been improved by topology optimization with respect to weight and structural performance (Niemann et al. 2012). The design is derived from the composite lattice structure, whose geometry (Vasiliev 2001) is shown in Fig. 6.21, which has been successfully applied for years in Russian rocket technology for their excellent strength and stiffness to weight ratio (Vasiliev 2012).

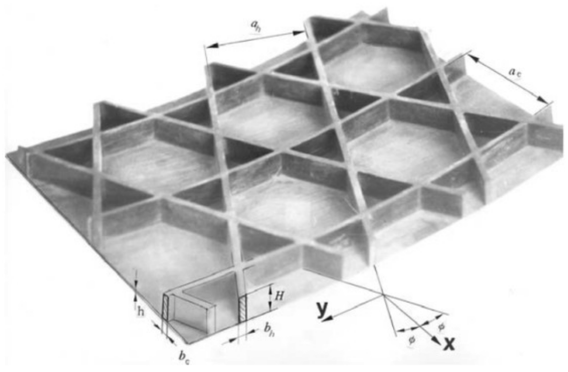


FIGURE 6.21: Geometry of lattice structure in a composite fuselage barrel

The ALaSCA project investigates aircraft fuselage designs made of carbon fibre reinforced plastic (CFRP) material. These materials have a high specific strength and high specific stiffness. However, as opposed to metals, which are homogenous materials, composites are orthotropic materials made of fibres held together by a matrix. Because fibre composites behave differently from metals, a new weight efficient composite anisogrid fuselage design is sought. The most weight efficient application of such a material occurs when the fibres are oriented parallel to the loads. In the studied anisogrid structure, the grid forming stiffeners are predominantly made of a unidirectional composite in which the fibres are parallel, while the skin is made of a laminate consisting of plies oriented in various angles.

The design process of the composite lattice structure is a multi-parameter optimisation problem, for which a metamodel-based optimization technique is used to obtain the optimal solution describing the lattice element geometry in this paper. In the optimization of a lattice composite fuselage structure, one of the design variables, the number of helical ribs, is integer. It is assumed that it is allowed to perform a response function evaluation only for points that have discrete values of the design variables (Balabanov and Venter 2004). This makes it impossible to initially ignore the discrete nature of the design variables, solve a continuous problem and adjust the result to the given set of the discrete values, as sometimes suggested (Stolpe 2011).

In the following, a discrete form of genetic algorithm (GA) (Michalewicz 1996, Bates et al. 2004) is used to search for the optimal solution in terms of weight savings subject to stability, strength and strain requirements. The optimal solution has been examined by the FE simulation of the lattice fuselage barrel to determine the true structural responses which are then compared to those provided by the metamodels. A composite lattice barrel, considered in the current ALaSCA EU FP7 research project, is used to demonstrate the efficiency and accuracy of the metamodel-based optimization technique as well as provide the designers with a wealth of information on the structural behaviour of the novel anisogrid design.

6.7.1 Finite Element modelling and simulations

An automated multiparametric global fuselage barrel finite element (FE) tool is developed for optimization purposes. This tool simulates the behaviour of a user defined composite anisogrid fuselage barrel under specified loads. The software used are MSC Patran and MSC Nastran; while the programming language is PCL. This model generation code is a powerful pre and post processing tool.

The automated multiparametric global barrel FE tool consists of three pieces of codes: a request file, a pre-processing function and a post processing function. The request session file contains the list of parameter sets used as inputs for the model generation. This piece of code effectively contains the list of fuselage models to be simulated. The pre-processing file generates the fuselage models, requests the analysis, and after the analysis is complete, calls the post processing function. The post processing file harvests and formats the results. A user defined number of models can thus be generated and analysed in a batch mode with this program. The detailed flowchart for the automated multiparametric global barrel FE tool is shown in Fig. 6.22.

The code models a simplified section of a composite anisogrid fuselage barrel. Geometric factors such as fuselage section length and diameter, skin thickness, stiffener locations and stiffener cross sections can be defined by the user. Also the material properties, loads, and mesh density are user defined. Using these inputs, the smallest and most important building block of the code, the unit cell geometry, is generated first. In the case of the fuselage barrel here discussed, the unit cell is triangular as seen in Fig. 6.23. Then the nodes associated with loads and multipoint constraints are created. The early creation of these nodes allows for a continuous low node id nomenclature being applied

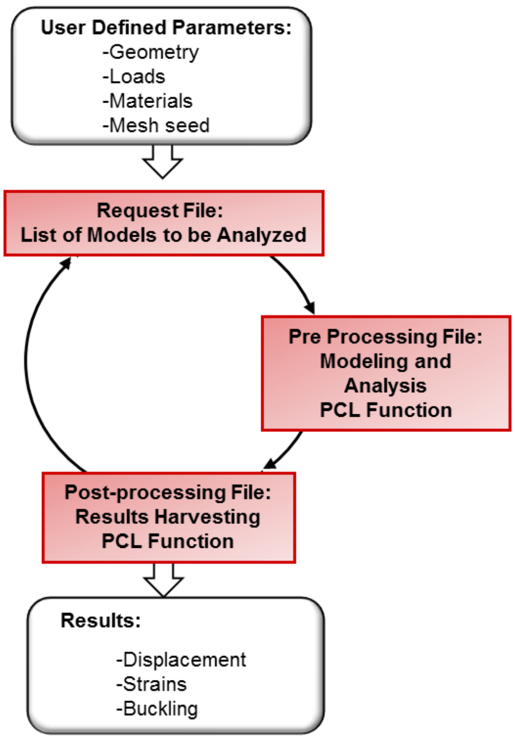


FIGURE 6.22: Automated multiparametric global barrel FE tool flowchart

to the loaded nodes. This is instrumental for the application of the loads and boundary conditions. The barrel mesh is then created starting with the meshing of the unit cell including the skin and the stiffeners. The sample mesh shown in Fig. 6.23 can be made finer or coarser by the user. The unit cell mesh is then multiplied via translation and rotation to generate the full fuselage barrel mesh. The material properties are applied to the skin, frames and helical ribs. Miscellaneous settings such as loads, load case creation, section property request, and group formation are implemented. Finally, the static and the linear buckling analyses are set up in MSC/Nastran Solution 400 and started. When the analysis is done, the post processing function is called.

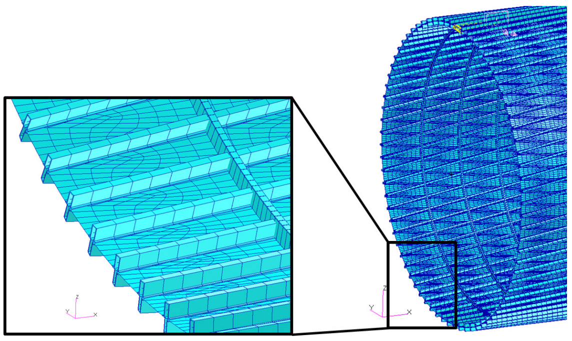


FIGURE 6.23: Barrel with visualised frames and helical ribs

The thus built model consists of a constant radius simplified fuselage barrel with triangular skin fields, which are bounded by hat shaped helical ribs and Z shaped circumferential frames. The skin is modelled with shell elements. The helical ribs, which are located on either side of the skin, are modelled with offset beam elements. The circumferential frames, located on the inner side of the skin, are also modelled with offset beam elements. A sample fuselage model is shown in Fig. 6.23 with a three dimensional visualization of the stiffeners modelled with beam elements. The geometry of the analysed fuselage barrel is described in detail in Section 6.7.2.

After analysis, the post processing function imports the results, arranges the model on the screen, generates strain and buckling ranges, and takes screen shots of the results. The strains are harvested from a result section which excludes the load application zones and boundary condition application zones so as to avoid numerically induced strain peaks. Displacements are also extracted from that results section. The buckling result, on the other hand is taken from the full fuselage model. Fig. 6.24 shows sample results. These results are then transferred into a *csv* file. At the end of the batch mode analysis, the results for all analysed models are summarized in the same *csv* file and screen shots are available for most results.

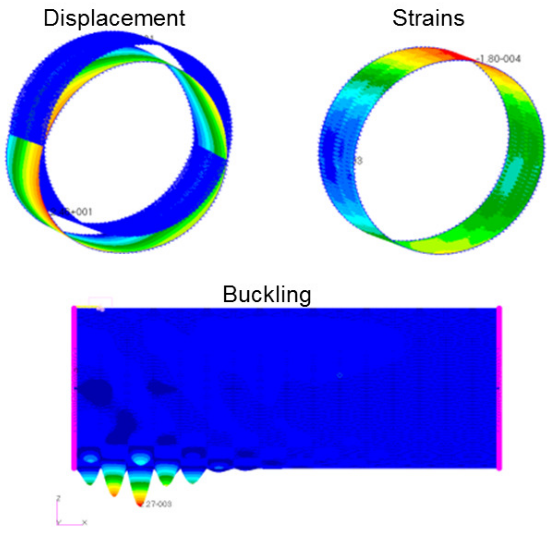


FIGURE 6.24: Sample results

The advantages of the automated tool are an efficient model generation and analysis when a large number of similar models are to be analysed, reliable model generation and data collection, as well as the flexibility to alter the unit cell geometry with acceptable programming effort in order to analyse different skin bay geometries. As more than 100 fuselage barrel geometries had to be analysed for the sample optimization described in the next section, the automated multiparametric global barrel FE tool was used.

6.7.2 Fuselage optimisation example

One of the aims of the ALaSCA EU FP7 research project was the development of an anisogrid composite fuselage structure. As part of the early design phase, an optimisation of the fuselage structure was performed, using as building data set a uniform Latin Hypercube DoE made of 101 points, each one of them corresponding to the input parameter set defining a specific fuselage geometry paired with the corresponding response generated via finite element models (MSC/Nastran software used). The bar chart of the minimum distances between the DoE building points is shown in Fig. 6.25, where the good uniformity of the DOE can be appreciated. Metamodels generated by HyGP were built to explore through a GA optimiser the structural properties of different geometries in order to find the optimum structure. The resulting optimum structure was then validated through FEM simulation to verify the structural behaviour predicted via optimization.

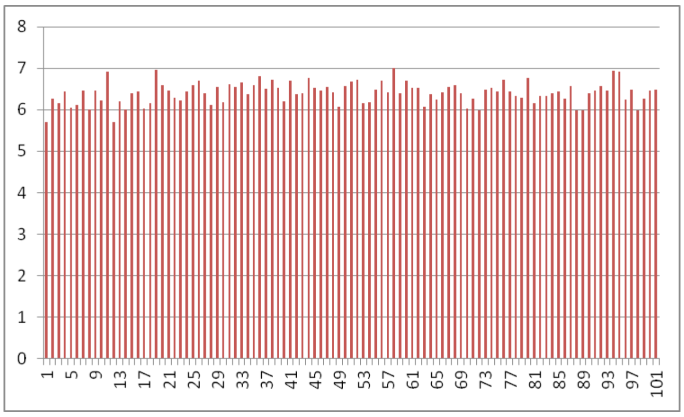


FIGURE 6.25: Minimum distances between points in the used 101-point optimal Latin hypercube design

The fuselage structure considered consists of a lattice-derived structure with a load bearing skin and stiffeners located on either side of the skin as shown in Fig. 6.26. The outer stiffeners are surrounded by protective foam, which in turn is covered by a thin aerodynamic skin (Niemann et al. 2012). The absence of rib crossings found in typical grid structures and shown in Fig. 6.21 is a substantial production advantage of this structure. The optimized grid type fuselage section is a simple structure without windows or floors consisting only of the repeated structural triangular unit cell.

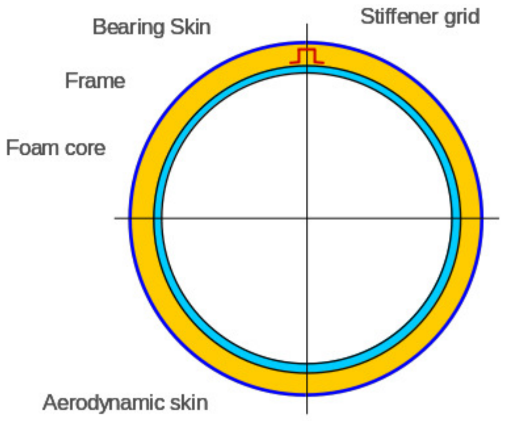


FIGURE 6.26: ALaSCA airframe concept with primary structure elements (Niemann et al. 2012)

Fig. 6.27 shows the finite element fuselage barrel model with the inner helical ribs in green, their counter parts on the outside of the skin in blue, the circumferential frames in yellow and the skin in red. The stiffening ribs are arranged at an angle so as to describe a helical path along the fuselage barrel skin. Hence, these ribs are called helical ribs. The helical ribs have a hat cross section, whereas the circumferential frames have a z shaped cross section. The upper barrel with the opaque skin illustrates the presence of only one set of parallel helical ribs on the outside of the skin. Below, the same barrel with transparent skin shows the presence of a second set of helix ribs winding around the barrel on the inside and in the opposite direction. These ribs in conjunction with the circumferential frames create uniform triangular skin bays. The helical ribs form an angle of 2φ between them as illustrated in Fig. 6.28. This angle remains constant throughout the barrel model.

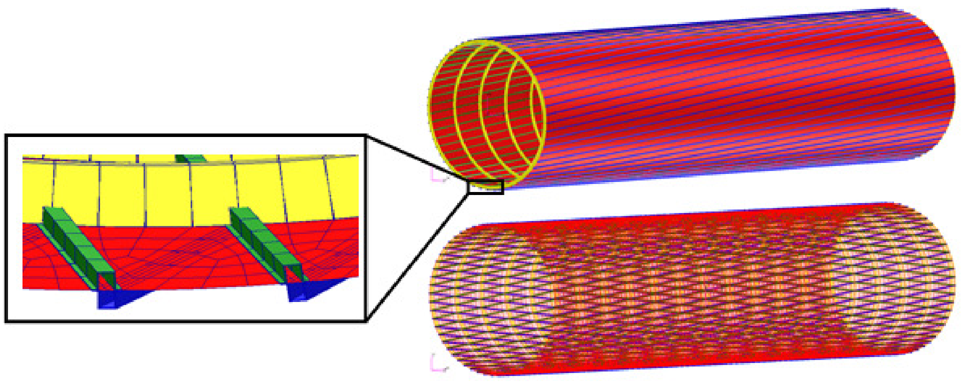


FIGURE 6.27: Example of fuselage barrel FE model

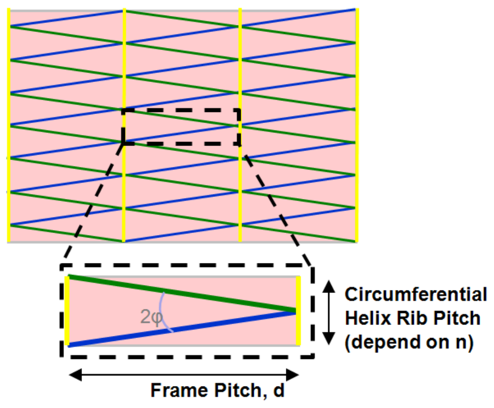


FIGURE 6.28: Skin bay geometry

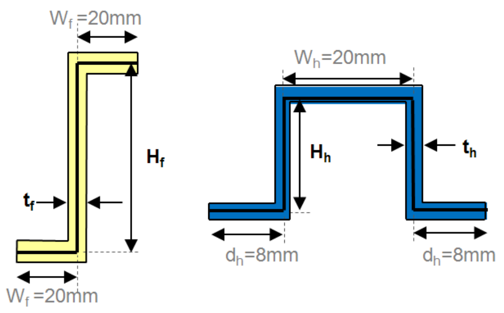


FIGURE 6.29: Circumferential ribs and helical ribs

The design variables were chosen to vary the geometry of the helical stiffeners and frames, the skin thickness, and the frame pitch without altering the triangular shape of the skin bay geometry. The seven optimization parameters are:

1. h, the skin thickness in mm
2. n, the number of helical rib pairs around the circumference of the barrel
3. t_h, the helical rib thickness in mm
4. H_h, the helical rib height in mm
5. d, the circumferential frame pitch in mm
6. t_f, the circumferential frame thickness in mm
7. H_f, the circumferential frame height in mm

These parameters are varied between the maximum and the minimum bounds listed in Table 6.8. The design variables are shown in Fig. 6.28-6.29. By altering the frame pitch, the height of the triangular skin bay is affected. Similarly, the number of helical ribs changes the width of the base of these triangular bays. Consequently, these two variables change the area and the angle 2φ of these skin bays, and thus are - along with the skin thickness - instrumental in influencing the buckling behaviour of the structure. The rib and frame geometries are also affecting the buckling of the fuselage globally and locally. Stiffer reinforcements on the edges of the skin bays reduce global and skin bay buckling. Although stability is expected to be the critical failure mode for the fuselage structure, the variables are also affecting the strength of the structure. The composite material fails if it is strained beyond a maximum value. Finally, the fuselage has to have a certain stiffness in bending and in torsion to avoid excessive global deformations in flight. The design variables are varied within the bounds shown in Table 6.8 to generate fuselage structures, which are then evaluated with respect to the mentioned failure modes.

TABLE 6.8: Design variables and space

Symbol	Design variable	Lower bound	Upper bound
Z_1	Skin thickness (h)	0.6 (mm)	4.0 (mm)
Z_2	No. of helix rib pairs around the circumference (n)	50	150
Z_3	Helix rib thickness (t_h)	0.6 (mm)	3.0 (mm)
Z_4	Helix rib height (H_h)	15 (mm)	30 (mm)
Z_5	Frame pitch (d)	500 (mm)	650 (mm)
Z_6	Frame thickness (t_f)	1.0 (mm)	4.0 (mm)
Z_7	Frame height (H_f)	50 (mm)	150 (mm)

An upward gust load case at low altitude and cruise speed is applied to the modelled fuselage barrel and depicted in Fig. 6.30. At one end of the barrel, bending, shear, and torsion loads are applied while the opposite end is fixed. These loads are applied via rigid multipoint constrains, which force a rigid barrel end. While floors are not modelled, the masses from the floors are applied at the floor insertion nodes. Finally, the structural masses are applied to the skin shell elements via mass densities.

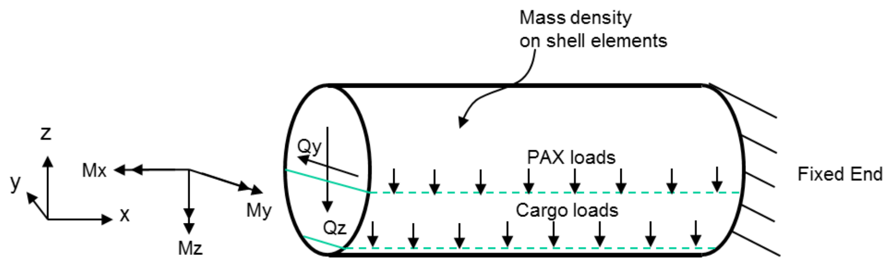


FIGURE 6.30: Load application

The optimisation constraints are strength, stiffness, and stability. The corresponding optimisation responses extracted from the FE models are the largest strains (tensile and compressive strains in the frames and in the helical ribs; tensile, compressive and shear strains in the skin), the critical buckling load, and the stiffness of the fuselage.

The results are normalised before further analysis. The mass per meter is computed for each model and then normalised against the largest mass per length of the 101 DOE fuselage models. Margins of safety (MS) are computed for the strains, the stiffness and the buckling results. A margin of safety is a normalization with respect to the allowable quantity, which measures whether the structures passes or fails when the load is imposed. Strains should be below an allowable limit strain. The strain margin of safety MS_ε is computed as follows:

$$MS_\varepsilon = \frac{\varepsilon_{max}}{\varepsilon} - 1 \geq 0 \tag{6.22}$$

where ε is the computed strain and ε_{max} the maximum allowable strain. Stiffness, on the other hand, should be larger than the allowable limit stiffness. Therefore the margin of safety for stiffness MS_S is computed as follows:

$$MS_S = \frac{S}{S_{min}} - 1 \geq 0 \tag{6.23}$$

where S is the computed stiffness and S_{min} the minimum allowable stiffness. Margins of safety can be positive or negative. A positive margin of safety shows that the computed value found in the structure does not violate the allowable value, and thus the structure is acceptable. A negative margin of safety, on the other hand, shows that the computed value violates the allowable value. Hence the structure fails and should be redesigned. The normalisation of the studied results allows for an easy comparison and, in the case of the margins of safety, a ready detection of failed fuselage geometries.

The 101-DOE FE models corresponding to the 101 DOE training data are generated using the automated multiparametric global fuselage barrel FE tool. Since a large number of similar FE models are generated and analysed, a mesh convergence study is not done for each individual model, but rather for one representative model. The resulting mesh, called the *DOE mesh*, is applied to all 101 DOE FE models. This is done for time efficiency reasons. The subsequently obtained optima are validated via FEM. A mesh convergence study is then done for each fuselage model corresponding to an optimum. Furthermore, smeared orthotropic material properties, rather than laminates consisting of discreet orthotropic plies, are applied to the skin in each model. The stacking sequence of the laminate is not optimised to avoid additional integer design variables. Once an optimum satisfying the constraints is found, a new design with a realistic skin ply stacking sequence can be determined.

Using HyGP methodology, the 101 obtained response sets are then used to build the metamodels for the structural responses, which are the normalised responses of strength, stiffness, stability and the fuselage barrel weight. In case I, Section 6.7.2.1, only the strength responses are taken into account in order to generate an optimal fuselage geometry; this is done because the generation of the buckling results via FEM take remarkably longer than the generation of the strength results. In a second case, case II (Section 6.7.2.2), the stability, strength, and stiffness responses are used to generate an optimal fuselage structure.

6.7.2.1 Case I

The explicit expressions for the responses related to tensile strain, compressive strain, shear strain and weight of the fuselage barrel are built by HyGP. As example, the expression for tension strain is given below:

$$\begin{aligned} f_{ts} = {} & -1.68356 + 1.25543\,\mathrm{Z}_1 + 0.690658\,\mathrm{Z}_3 \\ & - 0.005447\,\mathrm{Z}_2\,\mathrm{Z}_3 - \frac{0.266889\,\mathrm{Z}_1}{\mathrm{Z}_3} + \frac{4.00324\,\mathrm{Z}_2}{\mathrm{Z}_5} + \frac{0.00664789\,\mathrm{Z}_4}{\mathrm{Z}_3} \\ & - \frac{0.500119965039\,\mathrm{Z}_1 \left(\frac{868.596\,\mathrm{Z1}}{\mathrm{Z}_2} - 0.38115\,\mathrm{Z1}\,\mathrm{Z}_4 + \frac{3.14228\,\mathrm{Z1}^2}{\mathrm{Z}_3} + \frac{0.0000033183\,\mathrm{Z}_1\,{\mathrm{Z}_2}^2\,{\mathrm{Z}_4}^5}{{\mathrm{Z}_6}^3\,{\mathrm{Z}_7}^3} \right)}{\mathrm{Z}_2} \end{aligned} \tag{6.24}$$

where Z_i with $i = 1, \ldots, 7$ are the design variables detailed in Table 6.8 (see "Symbol" column).

The optimisation is validated by analysing the optimum fuselage geometry with FEM. The fuselage response predicted via optimisation and the response obtained via FEM are compared. The optimisation result is acceptable if the critical margin of safety predicted via optimisation is within 0.10 of the same margin obtained via FEM. Furthermore, the critical margin should be non-negative. If the first optimum does not fulfil these requirements, the optimisation results are improved iteratively until the requirements are met.

Two optimisation loops are required to reach a satisfying result when only strength constraints are used. Table 6.9 compares the responses predicted via optimisation and the corresponding responses obtained using the FEM.

TABLE 6.9: Optima obtained using strength constraints

Model	Tensile strain (MS)	Compressive strain (MS)	Shear strain (MS)	Normalised mass
Reference: Model nr. 52 (DOE mesh)	1.08	0.41	1.94	0.18
Optimum I (DOE mesh)	0.52	0.02	1.35	0.11
Optimum I (converged)	0.36	**-0.09**	1.21	0.11
Prediction I (HyGP+GA)	**0.02**	**0.00**	**1.42**	**0.10**
Optimum II (converged)	0.54	0.04	1.54	0.12
Prediction II (HyGP+GA)	**0.03**	**0.01**	**1.64**	**0.11**

From the 101 DOE FE models, model number 52 is the most weight efficient geometry that fulfils the strength constraint. The responses for model 52 clearly show the optimisation potential. The critical margin of safety of 0.41 is in compressive strain with a normalised weight of 0.18. When designing an aircraft part, one aims at a critical margin of safety close to zero so as to minimise weight. Furthermore, it is desirable that the optimisation predicted margins of safety be smaller, or conservative, with respect to the actual fuselage response determined via FEM so as to minimise design changes after the optimisation. The responses obtained via the optimisation loops discussed below show a reduction in the margins of safety and the corresponding reduction in normalised weight.

The response predicted for the first optimum (Prediction I) matches the DOE mesh response (Optimum I (DOE mesh)) better than the converged mesh response (Optimum I (converged)). The optimum parameter set is modelled in FE first with the mesh used for the DOE. This mesh is called the *DOE mesh*. Subsequently the same optimum parameter set is modelled with a mesh resulting from a convergence study for this particular geometry and yielding converged results. This mesh is called the *converged mesh*. The two structural responses obtained with FE are then compared to the response predicted via the optimisation.

The first optimum has a normalised weight of 0.10 with a predicted critical margin of safety of 0.00 for compressive strains. Using the DOE mesh FEM, the critical compressive strain margin of safety is 0.02. In the case of the DOE mesh, the critical margin is positive and the predicted margin is conservative. The optimisation predicts accurately and precisely the critical structural response of the DOE mesh model. This is expected as the DOE was established with the same mesh.

However, in the case of the converged mesh, the critical margin is negative, -0.09. The strains obtained with the converged mesh are higher than those obtained with the DOE mesh, which shows that the actual fuselage structure obtained via the first optimisation loop fails in compression. This should be avoided even though the optimisation and the FE margins are close. Furthermore, the non-critical strain margins obtained via FEM do not match the predicted margins and some predicted margins are non-conservative independent of the mesh. A predicted non-conservative margin of safety is to be avoided. However, in this optimization a non-conservative margin of safety is acceptable for non-critical constraints as these have a smaller impact on the design as the critical constrains. Due to the failure in compressive strain, the first optimum is rejected, and the converged mesh results of the first optimum geometry flow into the next optimization loop so yielding the second optimum.

The second optimum (Prediction II) yields a critical margin of safety in compressive strain, which is conservative with respect to the corresponding converged mesh FE model results (Optimum II (converged)). The predicted compressive strain margin is 0.01, while the margin obtained via FEM is 0.04. This is an excellent result since the margins are less than 0.10 apart. The non-critical tensile and shear strain margins are predicted to be 0.03 and 1.64 respectively, while the FEM yields corresponding margins of 0.54 and 1.54 respectively. The predicted tensile strain margin is conservative, whereas the predicted shear strain margin is not. Since the shear strain margin is the least critical margin, this non-conservative prediction is acceptable. The second optimum fulfils the requirement that the critical margin of safety predicted via optimisation be within 0.10 of the margin obtained via FEM. Furthermore, the critical predicted margin is conservative as it is smaller than the margin obtained via FEM. At the end of the two optimisation loops, a low weight optimum with accurate predictions of the critical constraints was therefore obtained.

Table 6.10 shows the geometric parameters of the two optima found and the reference design when only the strength responses are used for the optimisation. A weight efficient optimum - Optimum II - has been reached by minimising the number of frames and ribs and thus generating large skin bays.

TABLE 6.10: Design variable values and strength constraints for found optima

Design	Skin thickness h (mm)	Nr. of helix rib pairs n	Helix rib thickness t_h (mm)	Helix rib height H_h (mm)	Frame pitch d (mm)	Frame thickness t_f (mm)	Frame height H_f (mm)
Reference: DOE Model nr. 52	2.33	109	0.65	23.40	627.50	1.51	66.80
Optimum I	2.08	60	0.60	27.90	627.70	1.00	50.00
Optimum II	2.28	60	0.66	27.90	627.70	1.00	50.00

Through the optimisation loops, the triangular skin bay area increases by 82% - almost doubles - from model 52 (DOE Model nr. 52) to the second optimum (Optimum II). This large bay area is due to the decrease of the number of helix rib pairs to 60, which is close to the lower bound of 50, and the increase of the frame pitch to 627.70 mm, which is close to the upper bound of 650 mm. The resulting skin bays are large triangular skin bays with a base width of 209.44 mm, a height of 627.70 mm and an angle between the crossing helical ribs of 2φ=18.94 degrees. The applied loads are low enough that the structure required to bear them can have large skin bays when only the strength requirement is considered.

The helical ribs have a tall and slender hat-shaped cross section. The second optimum yields helical ribs, which have a thickness of 0.66 mm, which is close to the minimal allowed valued of 0.60, and a height of 27.90 mm, which is close to the maximum allowed value of 30 mm. This leads to a large moment of inertia and thus to a high bending stiffness. The circumferential frames, on the other hand, carry smaller loads than the helical ribs. They are also less instrumental in preventing fuselage bending. Therefore, the resulting frames are thin and small both dimensions reaching the minimal bounds of 1.0 mm and 50.0 mm respectively. When only the strength response is used in the optimization, a fuselage barrel with large skin bays, few thin and tall helical ribs, and few thin and small circumferential frames is reached.

6.7.2.2 Case II

In addition to the strength constraint, the stability and stiffness responses of the 101 DOE models are used to generate an optimum in this second case. The predicted responses (Prediction III) and the FE responses (Optimum III (converged)) of the third optimum are shown in Table 6.11.

TABLE 6.11: Optimum obtained with strength, stiffness and stability constraints

Model	Tensile strain (MS)	Compressive strain (MS)	Shear strain (MS)	Buckling (MS)	Torsional stiffness (MS)	Bending stiffness (MS)	Normalised mass
Optimum III (converged)	0.62	0.08	1.09	**-0.07**	1.21	0.89	0.29
Prediction III (HyGP+GA)	**0.20**	**0.23**	**1.27**	**0.00**	**1.21**	**0.89**	**0.29**

The response predicted via optimisation is again verified with a FEM simulation of the optimum geometry. The third optimum displays a predicted critical margin in buckling of 0.00 with a normalized weight of 0.29. The buckling margin obtained with the converged FE model is -0.07. Although, the difference between the predicted and the FEM margins is less than 0.10, the fuselage fails via global buckling as shown with the FEM. The optimization does not predict this failure.

The compressive strain margin of safety is no longer the critical margin when stability is considered. In fact, for this particular geometry, the tensile strain is more critical than the compressive strain. The predicted tensile strain margin of 0.20 is conservative when compared the 0.62 margin obtained via FEM. The predicted compressive and shear strain of 0.23 and 1.27, respectively, are not conservative compared to the compressive strain margin of 0.08 and the shear margin of 1.09 obtained via FEM. This is not desirable, but acceptable as these are not the critical margins.

The predicted stiffness margins correspond to the margins obtained via FEM. The torsional stiffness margin is 1.21, while the bending stiffness margin is 0.89. Stiffness is not a critical constraint for this design optimization.

Although the critical buckling margin of safety determined via FEM is negative, no additional optimization iterations have been performed due to time constraints. If additional optimization loops were to be performed, the following goals would be pursued: all margins of safety should positive, the critical margin of safety prediction should be conservative with respect to FE result, and the predicted critical margin should be within 0.10 of the FE result.

The parameter set for the final optimum geometry is as listed in Table 6.12. Stability requirements lead to smaller skin bays as larger panels buckle at a lower load than smaller panels. The skin bay area of the third optimum geometry decreases by 68% compared to the skin bays of the second optimum. The number of helix ribs increase to 150, which is the upper bound for this variable. The frame pitch decreases to 501.50 mm, which is

TABLE 6.12: Design variable values for optimum obtained with stength, stiffness and stability constraints (Optimum III)

Design	Skin thickness h (mm)	Nr. of helix rib pairs n	Helix rib thickness t_h (mm)	Helix rib height H_h (mm)	Frame pitch d (mm)	Frame thickness t_f (mm)	Frame height H_f (mm)
Optimum III	1.71	150	0.61	27.80	501.70	1.00	50.00

close to the lower bound of 500.00. The resulting skin bays are small triangular skin bays with a base width of 83.78 mm, a height of 501.70 mm and a shallow angle between the crossing helical ribs of 2φ=9.55 degrees. These small and shallow skin bays are excellent against buckling. Also, the normalized weight increases remarkably from 0.11 for the second optimum to 0.29 for the third optimum. Stability is a weight driving factor.

The hat shaped helical ribs remain tall and thin with a thickness of 0.61 mm and a height of 27.80 mm. The z-shaped circumferential frames remain unchanged from the third optimum with a thickness of 1.0 mm and a height of 50.0 mm. At first sight, these thin stiffeners are surprising when stability is considered. A thickening of the stiffeners is expected to avoid local buckling of the webs and flanges. These thin stiffeners are direct result of the modelling technique. Helical ribs and circumferential frames are modelled with beam elements, which are unable to represent local buckling in the webs and in the flanges. Only global beam buckling can be investigated with beam elements. Therefore, stiffeners would have to be modelled more precisely with shell elements or analysed via hand calculations to detect local buckling. These local buckling analyses are expected to generate thicker stiffeners.

6.7.2.3 New design

The final study generated a new design based on the third optimum. Since the smeared material properties were used for all studied FEM models, the laminate ply stacking sequence was not configured. A realistic skin laminate is now applied to the skin of the third optimum to generate a new practical design. Using a standard CFRP ply thickness of 0.125 mm, the skin thickness increases from 1.71 mm in the third optimum model to 1.75 mm. The same ply material properties were used as in the optimization work. The structural responses of the new design are shown in Table 6.13.

Compared to the FEM results corresponding to the model of the third optimum, the lay-up of the new design slightly increases stability, although the margin of safety in

TABLE 6.13: New design response

Model	Tensile strain (MS)	Compressive strain (MS)	Shear strain (MS)	Buckling (MS)	Torsional stiffness (MS)	Bending stiffness (MS)	Normalised mass
New design (converged)	1.15	0.19	1.31	**-0.04**	1.25	0.81	0.29

buckling is still negative with -0.04. All other margins are positive. Considering the small negative buckling margin of safety, it is expected that a small change such as an increase in skin thickness could be sufficient to obtain a zero or positive margin of safety. A preliminary light weight design, which fulfils the stability, strength and stiffness requirements, can be produced from this optimization result.

6.7.2.4 Remark

When only the strength constraints are used, the optimization results are such that all margins of safety are positive, the predicted critical margin of safety is smaller than the FE margin of safety, and the critical predicted margin of safety was within 0.10 of the FE result. Thus the optimization result obtained from the strength constraints is valid and leads to a structurally sound preliminary fuselage design.

Including the stability and stiffness constraints in addition to the strength constraints, the optimization results fulfil the condition that the predicted and the FE margin of safety results are within 0.10 of each other. Unfortunately, the actual critical margin is negative leading to a premature buckling of the fuselage structure. When stability, strength, and stiffness constraints are used, the optimization leads to a design requiring structural adjustments to be viable. Two major factors are expected to improve the results: an increase in data points and individual convergence studies for each model. Due to time constraints, these improvements could not be implemented in this optimization. Knowing these error sources, the resulting optimum and predicted responses of the structural behaviour are of satisfactory accuracy. Also the new design resulting from the optimization is of good quality where only small design changes are required to obtain a weight efficient structure that fulfils the strength, stiffness and stability constraints. The performed optimisation then represents a good tool for early design stages.

6.7.3 Results with MLSM-based optimisation

The optimisation shown in the previous sections was orginally meant to be based on metamodels generated by the MLSM technique available in the commercial optimisation software HyperStudy (Toropov et al. 2005, Alt 2009, Zeguer and Bates 2011). A few attempts on "Case I" however did not produce satisfying results, as the normalised mass returned was negative. In the following these attempts are detailed and confirm that HyGP-based optimisation can produce accurate results when other techniques fail.

6.7.3.1 MLSM metamodels building

Using the 101-point DoE defined in Section 6.7.2, the moving least squares method (MLSM) was used to generate two sets of metamodels for the objective (normalised mass) and the constraints (tensile strain MS, compressive strain MS, shear strain MS), using respectively 1st (set A) and 2nd order (set B) polynomials as a mathematical base. In both cases gaussian weighing was used, and closeness of fit was set to 5 (mid value in range $[1, 10]$ allowed by HyperStudy). In Table 6.14 the coefficients of determination R^2 of the resulting metamodels on the building data set are reported.

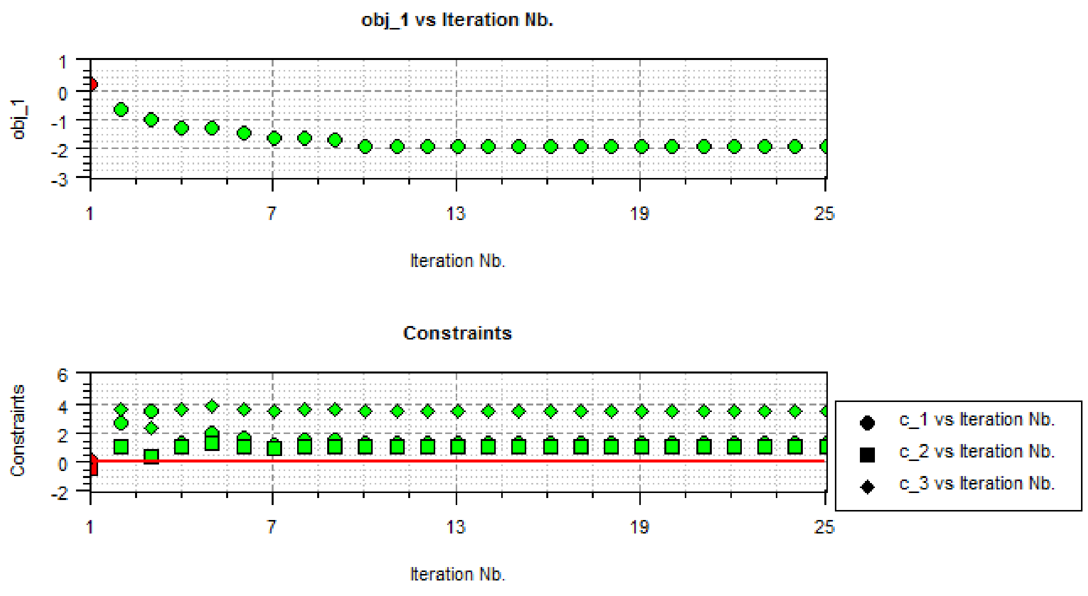
TABLE 6.14: Value of the coefficient of determination R^2 of case I MLSM metamodels on building data set

Response	Set A 1st Order MLSM R^2	Set B 2nd order MLSM R^2
MS tensile strain	0.9958747	0.9988923
MS compressive strain	0.9977960	0.9993751
MS shear strain	0.9995940	0.9998596
Normalised mass	0.9877087	0.9942394

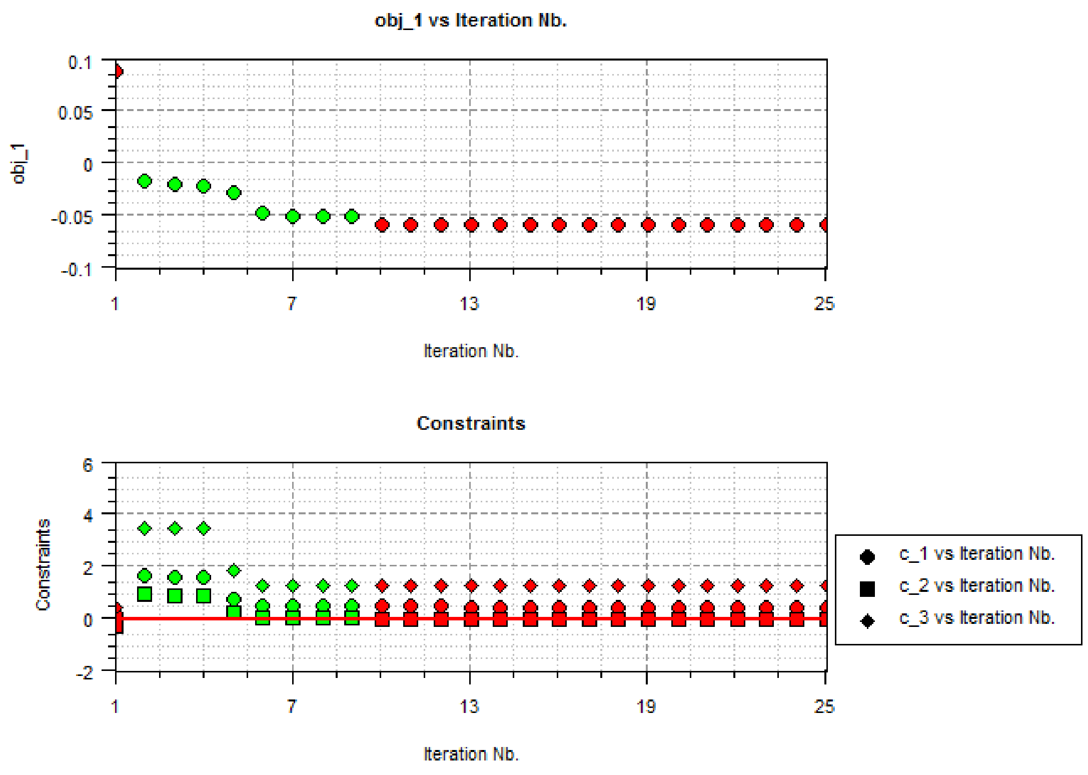
The GA algorithm coded in HyperStudy was used to search for the optimal configurations of the 7 input parameters leading to the minimum normalised mass on the condition that the margins of safety MS were all not negative. The convergence history is shown in Fig. 6.31. The found optima are shown in Table 6.15 (input parameters rounded to the first decimal place).

TABLE 6.15: Value of the coefficient of determination R^2 of case I MLSM metamodels on building data set

Set	h mm	n ()	t_h mm	H_h mm	d mm	t_f mm	H_f mm	t. strain MS	c. strain MS	s. strain MS	norm. mass
A	1.9	50	0.6	15.0	650.0	1.0	50.0	0.45	-0.01	1.23	**-0.06**
B	4.0	150	0.6	15.0	650.0	1.0	50.0	1.22	0.94	3.38	**-1.95**



(A) Set A



(B) Set B

FIGURE 6.31: GA convergence histories for optimisation with set A and set B metamodels (images generated by HyperStudy: *obj_1* stands for normalised mass, objective; *c_1*, *c_2* and *c_3* for the margins of safety MS, constraints)

The optima found by GA were of course not accepted, due to the negative normalised mass (in bold). The high coefficients of determination R^2 of MLSM metamodels reported in Table 6.14 were then ascribed to overfitting, for both first and second order MLSM regression. The GA search was therefore led to a design point where mass metamodel did not prove reliable, as physically inconsistent. Due to the failed MLSM-based optiimisation, HyGP was then given a try with the results detailed in the previous sections. It is important to note that in HyGP metamodelling "physical" inconsistency like the one experienced for MLSM mass regression could have been tackled using the approach described in Section 5.4 of the previous chapter.

6.7.4 Conclusion

In order to optimise a composite anisogrid fuselage barrel design, a 101-point DOE has been developed according to an extended uniform Latin hypercube design. Each data point corresponds to the response from FE simulations of a fuselage barrel. An automated tool which generates and analysis the fuselage barrel models was created for this study. Using these training data sets, the global metamodels, which are the explicit expressions of the sought structural responses as a function of the design parameters, have been built using HyGP methodology described in Chapter 5. The parametric optimization of the fuselage barrel was performed using Genetic Algorithm (GA) to obtain the best design configuration in terms of weight savings subject to stability, strength and strain requirements. The optimal solution and predicted structural responses have been verified by a FE simulation of the optimal lattice fuselage barrel.

Two optimal structures have been determined. The first structure only fulfils the strength requirement and yields a light weigh fuselage with few and thin helical ribs, large skin bays, and few circumferential frames. The second structure complies with the strength, stability and stiffness requirements and thus is a heavier structure with smaller skin bays and more stiffeners. The analysis allowed to identify stability as a driving factor for the skin bay size and the fuselage weight. The second optimal fuselage structure found requires only small design change to yield an acceptable preliminary design.

It is concluded that the use of the global metamodel-based approach has allowed to solve this optimisation problem with sufficient accuracy as well as provided the designers with a wealth of information on the structural behaviour of the novel anisogrid design of a composite fuselage. Preliminary tests have also proved the superior quality of HyGP metamodels with respect to MLSM ones.

6.8 Aerodynamic optimisation of NASA rotor 37 compressor rotor blade

NASA rotor 37 is a representative transonic axial-flow compressor rotor that has been used extensively in computational fluid dynamics (CFD) community to test optimisation algorithms and validate CFD codes (Dunham et al. 1998, Duta and Giles 2006, Ameri 2010).

Different approaches have been used to optimise the blade design under different constraints. For example, Samad and Kim (2008) used a three-dimensional RANS[6] solver to generate the data sets and built global response surfaces using second order polynomials: the Pareto front with respect to pressure ratio and adiabatic efficiency was then found using a multi-objective genetic algorithm coupled with an SQP optimiser. Shahpar et al. (2008) used Multipoint Approximation Method (MAM) to find an optimal blade design. The approach used by Oyama and Obayashi (2002) is also interesting (although for NASA rotor 67 optimisation) as no metamodels were used to reduce the computational cost: the minimum entropy design was found evaluating through direct simulations the points selected by an evolutionary algorithm. Duta and Giles (2006) used adjoint code to study the sensitivity of the mass flow to the twist of the midheight section of the blade.

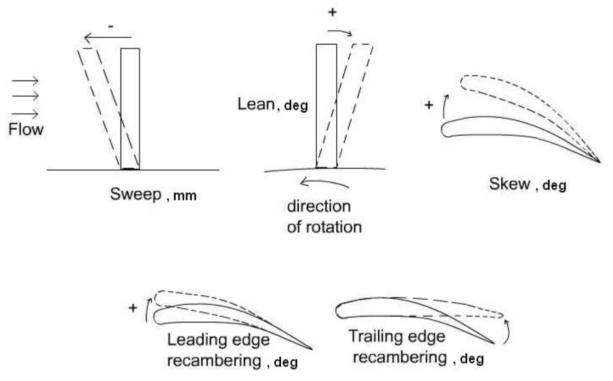
The aerodynamic optimisation of NASA rotor 37 compressor blade was considered a good test case for HyGP. First of all because to the best of the author's knowledge such optimisation has never been attempted using global metamodels generated by genetic programming. Secondly, the problem exhibits a relatively high dimensionality, with 25 input variables, and so in general it is a challenging problem due to the curse of dimensionality. It is important to note that only few examples have been found in literature of the application of genetic programming to design spaces of more than ten dimensions. Nordin et al. (1999) successfully solved a 40 input-variable data-mining problem without any dimensional reduction, whereas Smits et al. (2005) and Vladislavleva (2008) reduced a 23 input-variable problem to a lower dimensionality using a GP-based sensitivity analysis tool before generating the final model using genetic programming.

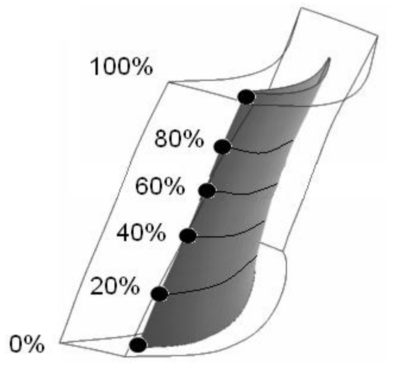
In the following sections the optimisation process and the resulting optimal blade are described. To assess HyGP metamodels reliability, the optimisation was repeated using global metamodels generated by MLSM.

[6]Reynolds-Averaged Navier Stokes system of equations.

6.8.1 Problem description

The blade parameterisation was done using five engineering parameters available in Rolls-Royce PADRAM code (Shahpar and Lapworth 2003): axial movement of sections along the engine axis (variable XCEN, in mm), circumferential movements of sections (variable DELT, in degrees), solid body rotation of sections (variable SKEW, in degrees), and leading/trailing edge recambering (variables LEMO and TEMO, in degrees). A sketch showing the physical meaning of the input variables is given in Fig. 6.32A. These design parameters define the shape of a bidimensional airfoil on five stations along the blade span, at 20%, 40%, 60%, 80%, and 100% (tip) of the overall span, as shown in Fig. 6.32B. The airfoil at the blade root was fixed (station 0%). The total number N of independent design variables is 25. B-spline interpolation was then used through the control stations along the span to generate smooth design perturbations in the radial direction.


Flow
-
Sweep , mm
+
Lean, deg
direction of rotation
+
Skew , deg
+
Leading edge recambering , deg
Trailing edge recambering , deg
(A) Deformation modes



(B) Control stations along blade span

FIGURE 6.32: NASA rotor 37 blade parametrisation

The optimisation problem was to find the values of the 25 parameters that maximise the adiabatic efficiency η of the blade, defined in Eq. (6.25):

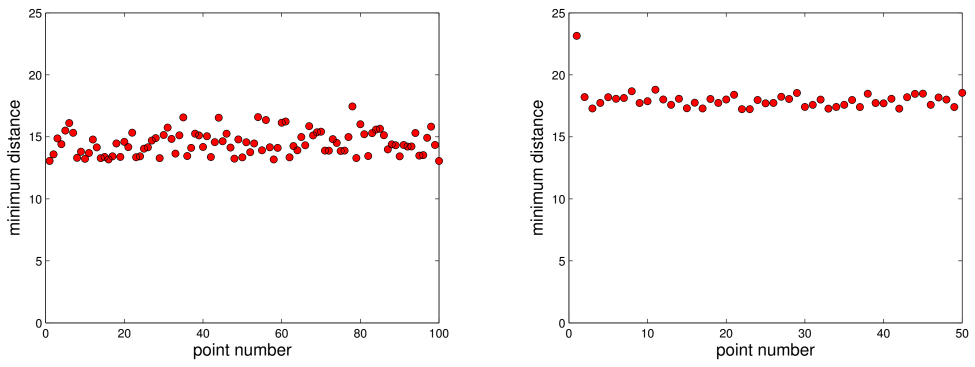
$$\eta = \frac{\left(\dfrac{P_{outlet}}{P_{inlet}}\right)^{\frac{\gamma-1}{\gamma}} - 1}{\dfrac{T_{outlet}}{T_{inlet}} - 1} \tag{6.25}$$

where $P_r = \frac{P_{outlet}}{P_{inlet}}$ is the pressure ratio and $\frac{T_{outlet}}{T_{inlet}}$ is the temperature ratio between outlet and inlet, respectively; γ is fluid specific heat. Constraints were defined on pressure ratio and mass flow through the rotor: a maximum perturbation of 0.5% with respect to the baseline pressure ratio $P_{r0} = 2.15$ and to the baseline mass flow $\dot{m}_0 = 20.1\ kg/s$ was imposed.

6.8.2 Input data and GP settings

The original range of the design variables ($[-5, 5]$ mm for XCEN and $[-0.5, 0.5]$ degrees for the other variables DELT, SKEW, LEMO and TEMO) was scaled to $[1.0, 11.0]$. In the scaled design space, a DoE made of 100 points was randomly generated. In order to improve the quality of the random design, a constraint on the minimal distance between points was imposed: the space-filling properties of the scaled DoE are shown in Fig. 6.33A through a plot of the minimum distance of each point to neighbouring DoE points. The average minimum distance is 14.49, the minimum distance standard deviation is 0.97.

For each DoE point, a few CFD simulations were performed with Rolls-Royce SOPHY (Shahpar 2005) to compute the corresponding values of efficiency, pressure ratio and mass flow rate. The data were used as HyGP building data set.



(A) building data set (B) additional data set C

FIGURE 6.33: Minimum distance between points in building data set (A) and additional data set for model penalisation (B) for efficiency, pressure ratio and mass flow symbolic regression

The constraint on pressure ratio P_r was recast in normalised form using two inequalities:

$$c_1 = \frac{Pr}{1.005\ P_{r0}} \leq 1 \tag{6.26}$$

$$c_2 = \frac{0.995\ P_{r0}}{P_r} \leq 1 \tag{6.27}$$

where P_{r0} is the baseline pressure ratio. The constraint on the mass mass flow $\dot{m}$ was similarly recast and normalised:

$$c_3 = \frac{\dot{m}}{1.005\ \dot{m}_0} \leq 1 \tag{6.28}$$

$$c_4 = \frac{0.995\ \dot{m}_0}{\dot{m}} \leq 1 \tag{6.29}$$

where $\dot{m}_0$ is the baseline mass flow.

Efficiency was also reformulated, to turn the maximisation problem (maximisation of efficiency η) into a minimisation problem (min η'):

$$0 \leq \eta \leq 1 \Rightarrow \eta' = 2 - \eta \Rightarrow 1 \leq \eta' \leq 2 \tag{6.30}$$

The scaled 100-point DoE was fed as a building data set into HyGP to generate the five metamodels of c_1, c_2, c_3, c_4, η' defined in Eqs. (6.26, 6.27, 6.28, 6.29, 6.30): 2 metamodels for pressure ratio, 2 for mass flow and 1 for the reformulated efficiency.

Preliminary tests with standard HyGP settings, not shown here, resulted in really irregular response surfaces that could not be used for optimisation. Metamodels featured low generalisation ability, typical of overfitting. The exclusion of the protected division from the primitives somewhat improved metamodel smoothness, but the introduction of the penalisation defined in Section 5.4, Chapter 5 with $p = 3$ proved to be decisive. An additional data set C made of 50 points uniformly distributed in the scaled design space (latin hypercube DoE) was used to bias the search for reliable metamodels of the reformulated efficiency η' and the four constraints. The minimum distance between points in C is plotted in Fig. 6.33B: the average minimum distance between points is 17.99, the minimum distance standard deviation is 0.85. On this additional data set metamodels of η' returning values lower than 1.1 were penalised, as well as metamodels for c_1, c_2, c_3, c_4 giving values lower than 0.5.

In the next sections the optimum found using HyGP metamodels is compared with the one returned by MLSM metamodels. In Table C.17 in Appendix C the settings used in HyGP experiments are reported.

6.8.3 Result of the optimisation performed with HyGP and GA

HyGP metamodels were explored using Multiobjective GA with Adaptive Range (AR-MOGA) from Rolls-Royce optimisation software SOFT (Shahpar 2002) to find the values of the 25 input parameters that optimise the rotor efficiency within the given constraints. Among the suboptima found, two designs were selected: their accuracy with respect to the responses provided by CFD simulations at the same design point is assessed in Tables 6.16 and 6.17.

TABLE 6.16: Validation of HyGP optimum 1 found by GA

	HyGP model	CFD	ϵ_{rel} (%)
η'	1.125	1.138	-1.14%
c_1	0.988	0.981	0.71%
c_2	0.994	**1.008**	-1.39%
c_3	0.999	0.999	0%
c_4	0.997	0.990	0.71%

TABLE 6.17: Validation of HyGP optimum 2 found by GA

	HyGP model	CFD	ϵ_{rel} (%)
η'	1.091	1.134	-3.79%
c_1	0.996	0.973	2.36%
c_2	0.999	**1.016**	-1.67%
c_3	0.952	**1.002**	-4.99%
c_4	0.993	0.987	0.61%

In Tables 6.18 and 6.19 the actual values of the efficiency η, the pressure ratio P_r and the mass flow $\dot{m}$ returned by CFD simulations at the two optima described in Tables 6.16 and 6.17 are compared to their corresponding values in the baseline design.

TABLE 6.18: Optimum 1: values of η, P_r and $\dot{m}$ in baseline design and HyGP optimised design (values by CFD)

	Baseline	Optimised	rel. variation (%)
η	0.857	0.862	+0.58%
P_r	2.15	2.12	**-1.39%**
$\dot{m}$ [kg/s]	20.1	20.19	+0.5%

TABLE 6.19: Optimum 2: values of η, P_r and $\dot{m}$ in baseline design and HyGP optimised design (values by CFD)

	Baseline	Optimised	rel. variation (%)
η	0.857	0.866	+1.05%
P_r	2.15	2.11	**-1.86%**
$\dot{m}$ [kg/s]	20.1	20.24	**+0.70%**

As shown by Tables 6.18 and 6.19, the errors on efficiency, pressure ratio and mass flow is generally under 2%. Optimum 1 stands out for a smaller violation of the pressure ratio constraint (-1.39% variation with respect to baseline pressure ratio against a maximum of 0.5%) with respect to optimum 2, where a higher efficiency is gained (+1.05%) at the cost of more severe violations on pressure ratio and mass flow constraint (-1.86% with respect to baseline pressure ratio, +0.70% with respect to baseline mass flow. In both cases the maximum variation allowed was 0.5%).

6.8.4 Comparison with optimum found by MLSM and SQP

NASA rotor 37 blade optimisation was repeated using MLSM to generate the metamodels for the objective and the four constraints, whereas SQP was used to find the optimum. The same 100-point DoE described in Section 6.8.2 was used. All points were used for metamodel building. As a result, the closeness of fit was not optimised: it was instead set to the maximum value (100) in order to encourage local accuracy (Toropov et al. 2005, Loweth et al. 2011). That was possible as the data are not affected by noise, so smoothing was not required. A second order polynomial was chosen as basis for the MLSM.

In Table 6.20 the values of η', c_1, c_2, c_3, c_4 - Eqs. (6.26, 6.27, 6.28, 6.29, 6.30) - at the optimum found are compared to the responses returned by CFD simulations. In Table 6.21 the resulting efficiency, pressure ratio and mass flow at the optimum are compared with the same parameters in the baseline design.

TABLE 6.20: Validation of MLSM optimum found by SQP

	MLSM	CFD validation	ϵ_{rel} (%)
η'	1.132	1.132	-0.05%
c_1	0.990	0.987	0.24%
c_2	1.000	**1.003**	-0.29%
c_3	0.986	0.995	-0.94%
c_4	1.000	0.995	0.54%

TABLE 6.21: Values of η, P_r and $\dot{m}$ in baseline design and in MLSM optimised design (values by CFD)

	baseline	optimised	rel. variation (%)
η	0.857	0.868	1.28%
P_r	2.15	2.13	**-0.78%**
$\dot{m}$ [kg/s]	20.10	20.11	+0.03%

The comparison of the optima found by SQP on MLSM metamodels with optimum 1 found using HyGP metamodels (Table 6.18) shows that MLSM allowed for a larger increase in the blade efficiency (1.28% against 0.58% produced by HyGP coupled with GA - Table 6.18) with a less severe violation of the constraint imposed on pressure ratio (-0.78% against -1.39% obtained by GA with HyGP metamodels).

The better performance of MLSM can be explained by the better accuracy of the MLSM technique with respect to HyGP. In particular, the fact that the optimum was not located on the design space boundary contributed to the superior accuracy of MLSM. Harewood et al. (2007) reported on the possible lack of accuracy in MLSM if the optimum is located on the boundary of the design space. Also, the smoothness of the metamodels resulted in the better performance of SQP search algorithm with respect to the non-deterministic GA search. It is worth noting neither optimisation process was able to improve the blade efficiency without violating at least a constraint.

The optimal set of blade parameters found through HyGP and MLSM metamodels are reported in Tables 6.22 and 6.23: the corresponding blade shapes are compared to the shape of the baseline blade design in Fig. 6.34.

TABLE 6.22: Blade parameters in optimum 1 found using HyGP metamodels

Station	XCEN (mm)	DELT (°)	SKEW (°)	LEMO (°)	TEMO (°)
0%	0.00000	0.00000	0.00000	0.00000	0.00000
20%	3.83924	-0.44331	-0.42574	0.15586	-0.07605
40%	1.04283	-0.37682	-0.32360	-0.24108	0.47043
60%	1.78793	0.38174	0.05784	0.00936	-0.10840
80%	-2.38414	-0.18487	0.32866	-0.39907	0.38807
100%	1.63545	0.27269	-0.39470	0.02017	-0.14352

TABLE 6.23: Blade parameters in the optimum found using MLSM metamodels

Station	XCEN (mm)	DELT (°)	SKEW (°)	LEMO (°)	TEMO (°)
0%	0.00000	0.00000	0.00000	0.00000	0.00000
20%	0.01355	0.01992	-0.04297	-0.02942	-0.10893
40%	-1.09720	-0.03056	-0.04254	-0.13860	-0.06543
60%	0.59583	0.01364	0.05910	-0.11524	0.04775
80%	2.29663	-0.02520	0.07393	-0.50000	0.03837
100%	2.08573	-0.11509	0.05538	-0.19514	-0.02887

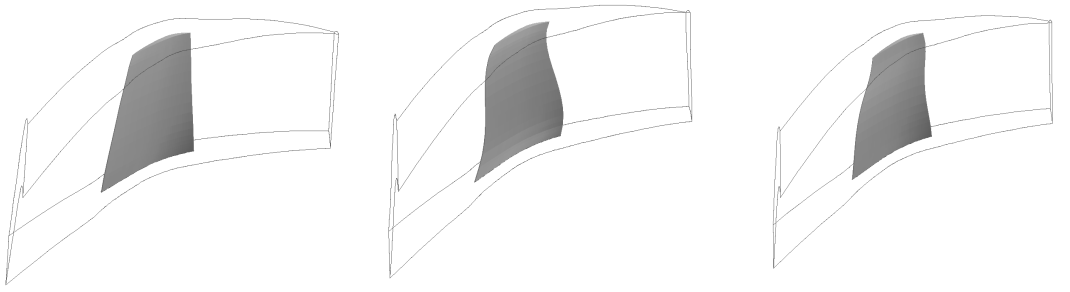

(A) baseline (B) HyGP-optimised blade design (C) MLSM-optimised blade design

FIGURE 6.34: Baseline and optimised blades

6.8.5 Computational cost

The experiments required for the optimisation of NASA rotor 37 blade were performed on a Linux machine equipped with a 2.27 MHz Intel Xeon processor. The independent HyGP runs were launched sequentially and no parallelisation was exploited in the fitness evaluation. HyGP settings for all 5 experiments (model inference of efficiency and the four constraints) are reported in Table C.17 in Appendix C.

The time required for the completion of each HyGP run is shown in Fig. 6.35. The average time for a complete GP run was 4.5 hours (16159 seconds).

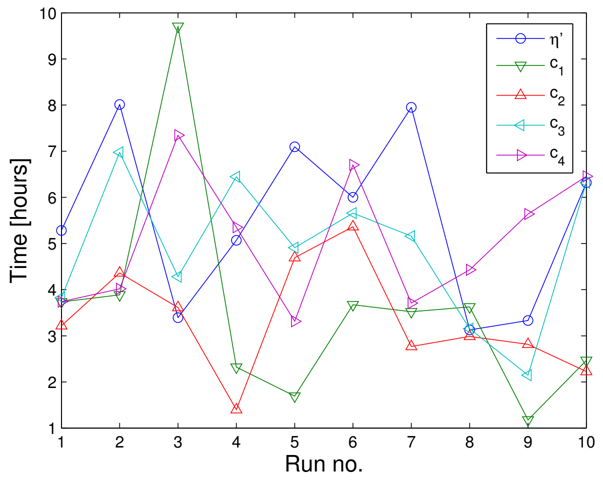


FIGURE 6.35: Time in hours for the generation of η', c_1, c_2, c_3, c_4

The time required to generate MLSM metamodels is as a matter of fact not comparable with HyGP, as HyperStudy produces the required MLSM models in less than 10 minutes on the same machine. The high computational cost of hybrid genetic programming is however a known issue, which can probably be tackled through massive parallelisation of the coefficients tuning and fitness evaluation stages. Limiting the tuning process to classes of individuals of specific phenotypical traits could also be a promising strategy, provided that such an approach does not affect the quality of the returned metamodels.

6.9 Conclusion

In this chapter the application of genetic programming to several modelling and optimisation problems has been presented. Data generated both by numerical simulations and real experiments have been modelled using HyGP, the genetic programming implementation introduced and validated in Chapter 5.

The level of accuracy and generalisation ability achieved by the generated models suggests that HyGP can be used for industrial optimisation problems. The quality of the metamodels has appeared to be critically dependent on the size of the building data set.

In the jet pump problem (Section 6.5) HyGP successfully returned an accurate and explicit metamodel that was accepted for industrial use. In Section 7.4.5, Chapter 7 it will be further shown that HyGP in this case performs better than an established implicit metamodelling technique as gaussian processes.

In the bread baking oven design optimisation (Section 6.6) the reduced size of the building data set had detrimental effects on the quality of the model. The use of the penalisation described in Section 5.4, Chapter 5 however has proved an effective and inexpensive way to make up for missing data and cope with the effect of the curse of dimensionality.

The fuselage barrel optimisation (Section 6.7) showed the limits of MLSM technique and proved that HyGP can be superior to other metamodelling techniques in some scenarios.

The comparison with MLSM on the NASA rotor 37 blade aerodynamic optimisation (Section 6.8) has shown that neither HyGP-based nor MLSM-based optimisation is able to lead to an efficiency improvement with respect to the baseline design without violating the constraints. Despite CFD validation showed that HyGP metamodels are slightly less accurate than MLSM ones in the optimal points found, the process has confirmed the usability of HyGP for industrial optimisation problems. The main advantage of HyGP metamodels over MLSM ones is their explicit form, which affords easy and inexpensive implementation of Monte Carlo sensitivity analysis.

HyGP appears therefore as a valid tool for industrial optimisation problems. Its main drawback is its high computational cost. Execution in parallel mode affords some reduction in these costs, but it is undeniable that higher computational efficiency has to be reached to encourage the use of HyGP. In Appendix A an insight into HyGP code structure is given and different execution modalities described.

Chapter 7

Search for factorised solutions in genetic programming

The experiments shown in the previous chapter prove that hybrid genetic programming can be very expensive in terms of time and computational resources required. Moreover, the generated metamodels may suffer from limited interpretability, being in most cases linear combinations of the primitives.

In this chapter a new strategy to increase the accuracy and the interpretability of the solutions is explored. To assess the advantages and the drawbacks of generating a global explicit expression via genetic means, a comparison on the same regression test functions introduced earlier is carried out with an implicit technique, gaussian processes.

7.1 Evolutionary advantage of linear combinations of terms

A characteristic behaviour emerges from the analysis of the experiments described in Chapter 5 and 6: HyGP frequently produces linear combinations of terms to approximate the true underlying function. This tendency can probably be explained in two ways. First of all, a linear combination may allow an acceptable approximation even from the beginning of the evolution, giving an evolutionary advantage with respect to other structures, and it can be improved by the addition of small terms, which nonetheless most of the times do not improve generalisation ability but result in overfitting. On the opposite, more complex structures may be penalised at early stages of the evolution by selection pressure, as to return accurate approximations they have to undergo more radical changes and are not likely to be improved by size growth normally occurring throughout the evolution, at least not in the same way linear combinations do.

Secondly, linear combinations are probably more resilient to crossover and mutation, which are known to be mostly destructive (see Section 3.1.5.5, Section 4.4.3, Section 4.4.4.3), and therefore they survive and progressively eliminate the good but not yet mature building blocks containing highly non-linear functional primitives.

Reducing the evolutionary advantage of individuals made of linear combinations of terms appears then as a possible way to explore different mathematical structures. Consequently, it might reduce premature convergence, lead to improvements in solutions' accuracy, and reduce the computational cost of the evolution. In fact, linear combinations contain in general more numerical coefficients to be tuned than highly non-linear individuals made of few terms, so on average they may be expected to incur higher tuning costs.

In the previous chapters it has been shown that the proliferation of linear combinations can be curbed by different strategies. Promising but not mature individuals may be indirectly protected from aggressive selection by using a multiobjective fitness function penalising "excessive" size. The problem in this case is however defining which size is to be considered "excessive", in the absence of any knowledge regarding the correct structure of the true function to be found. Secondly, introns may be used to provide neutral crossover and mutation points, allowing the exploration of new shapes and structures and relieving selection pressure (see Section 4.4.3, Chapter 4). Further protection may also be imposed by restricting the classes of individuals that are allowed to take part to selection, as done by Hornby (2006), who imposed that only individuals that have gone through a comparable amount of genetic specialisation (parameter defined as "age", measured in number of generations) can compete in a selection process (see Section 5.3.6, Chapter 5).

In the following a strategy that aims at protecting highly non-linear individuals, based on the idea of a "factorisation bonus", is presented and its effect on accuracy, average size and average computational cost of the individuals assessed. In addition, a complementary strategy based on editing is introduced that aims at reducing the number of nested divisions that frequently degrade GP individuals generalisation ability.

7.1.1 Factorisation bonus

The "factorisation bonus" is defined as a fitness value reduction (in the sense of quality improvement) granted to individuals whose mathematical structure has specific desirable features. The bonus is not based on a measure of the accuracy of the individual on the building data set but exclusively on the shape of the individual's syntax tree and its content. More in detail, the bonus is granted to the individuals that have at low depths of their tree structure primitives that may be considered "non-linear", with the aim of promoting factorised expressions and reducing the hegemony of non-factorised expressions as linear combinations typically are. In the experiments shown in the following sections, the primitives that are considered "non-linear" are multiplication, division (protected), sine, cosine, exponential, logarithm, hyperbolic tangent and reciprocal.

In order to award the bonus, the depth d_{tree} of the complete tree to be evaluated and the depth d of the "factorising" operation at the lowest depth in the tree (if found) are recorded. During the fitness evaluation stage then the fitness value of each individual is computed taking into account the value of d. If a factorising primitive is found relatively close to the root node, the factorisation bonus is awarded as a reduction to a tenth of the actual fitness value of the tree, as shown in Eq. 7.1:

$$F(i,t) = \begin{cases} 0.1 * F(i,t) & \text{if} \quad d < \min(0.2 * d_{tree}, 6) \\ F(i,t) & \text{if} \quad d >= \min(0.2 * d_{tree}, 6) \end{cases} \tag{7.1}$$

where the function $F(i,t)$ reported on the right of the curly bracket is the fitness function defined in Eq. 5.2, Section 5.2.5, Chapter 5. The "threshold depth" for the factorising operation found to be awarded the bonus is defined as the minimum of two parameters: a fifth of the tree depth ($0.2 * d_{tree}$) and a value set to 6. Preliminary tests have shown that using only the first does not affect the selection process after a few generations, due to the normal size growth of the individuals. On the other hand, using only a constant value (6 in this case) is not effective in the first generations of the evolution due to the reduced size of the individuals.

The "factorisation bonus" approach is interesting for its potential to reduce the size of the individuals and at the same time to support the exploration of complex mathematical structures.

7.1.2 Editing

Editing is a long exploited strategy, reported in Koza (1992) and also used more recently to reduce introns, as seen in Section 4.5.1, Chapter 4. In the following however a specific editing strategy applied in conjunction with the factorisation bonus approach is presented and its effect on generalisation ability assessed.

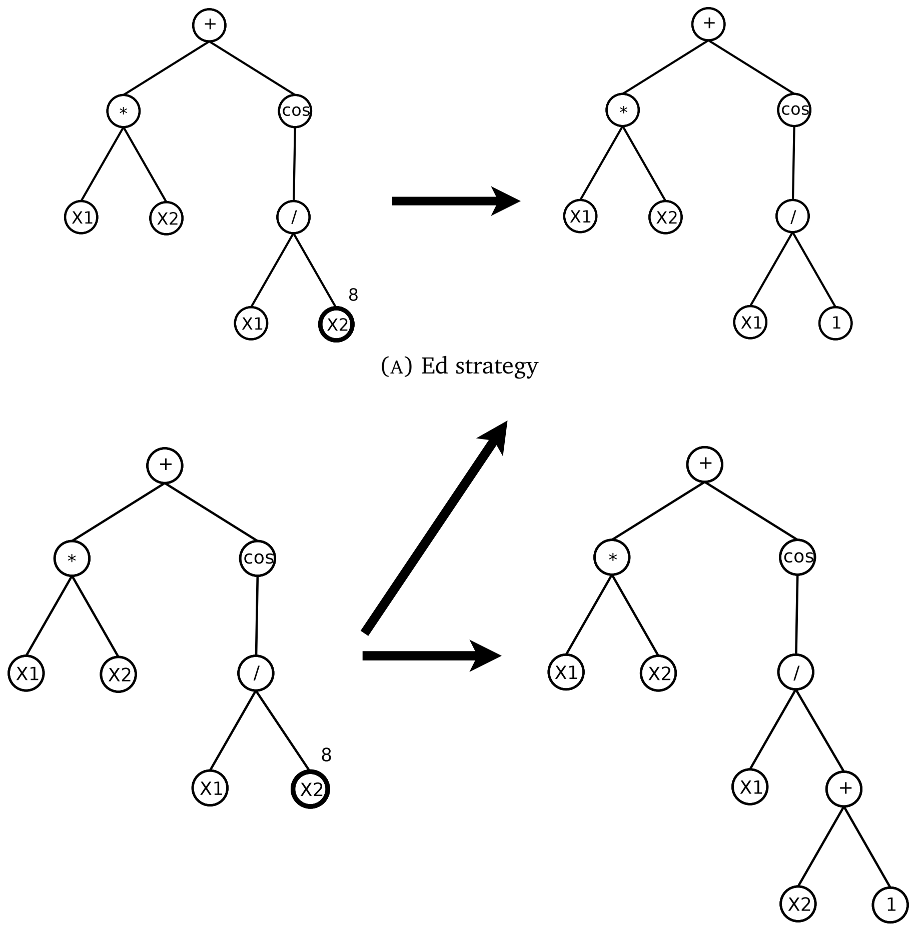
The editing strategy has been developed to reduce genetic programming tendency to generate numerical constants through nested protected divisions, which are not effectively penalised by the multiobjective fitness function defined in Eq. 5.2, Section 5.2.5, Chapter 5. Typically subtrees made of nested divisions terminates with the subtree A/X_j where A is a general subtree and X_j is a variable node. Those terms are particularly harmful for the generalisation ability of an individual as, although they return defined values during training thanks to the repairing action of protected division, they may lead to undefined behaviours (asymptotes) on the test data set.

Two strategies are put forward to tackle this problem, which will be called "Ed" and "Ed2", and they are both applied to the ancestors, in other words to the individuals before parameters insertion (see Section 5.2.2, Chapter 5). They both detect the presence of blocks A/X_j in the individual and check if 0 belongs to the domain of the denominator X_j of such blocks. If so, the strategy "Ed" replaces the variable with a constant value (1.0), which will later undergo tuning. The strategy "Ed2" instead randomly selects whether to replace the variable with a constant or adding a constant (1.0) to the variable, in order to minimise the risk of incurring an undefined case. An example of how these two strategies act on an ancestor is shown in Fig. 7.1.

7.2 Methodology and test problems

The strategies described in the previous section were tested following the same methodology described in Section 5.3.1, Chapter 5, used for all the experiments of that chapter.

The three test functions that proved most difficult for the hybrid genetic programming code described in Chapter 5 were used as test problems: Kotanchek, Salustowicz, and RatPol2D (see Section 5.3.1.1, Chapter 5). In addition, a new, four-dimensional test function taken from Korns (2011), which will be called "Korns P10" and is reported in Eq. 7.2, was used for symbolic regression tasks.



(A) Ed strategy

(B) Ed2 strategy

FIGURE 7.1: Editing strategies in action on node 8 of the tree represented on the left: the trees indicated by the arrow represent the edited individual in case 0 is in the domain of variable X_2

$$f_5(z_1, z_2, z_3, z_4) = 0.81 + 24.3 * \frac{2.0\, z_1 + 3.0\, z_2^2}{4.0\, z_3^3 + 5.0\, z_4^4} \tag{7.2}$$

The function “Korns P10” was selected as proxy of the class of “fairly simplistic formulas”, for which “current state-of-the-art symbolic regression systems suffer poor accuracy” or “fail to return a champion with the correct formula” (Korns 2011, p. 130). The importance of testing genetic programming implementations on this class of “intractable” problems (Korns 2011, p. 130) was detailed in Section 5.3.1.1, Chapter 5.

The training and validation data set for Kotanchek, Salustowicz, and RatPol2D test functions are defined in Section 5.3.1.2, Chapter 5. As for Korns P10 test function, the same domain bounds used by Korns (2011) on the four input variables were used. The building and validation data sets are defined in Table 7.1.

TABLE 7.1: Training and validation data sets for Korns P10 test function. In the middle column the domain used for optimal latin hypercube sampling is defined. In third column the validation data set is defined using the notation $Z_i = [a : dx : b]$. This means that values of variable Z_i are sampled from a to b with a step dx.

test function	building data (OLH DoE)	validation data (full factorial DoE)
Korns P10	300 points $-50 \leq Z_i \leq 50$ $i = 1, 2, 3, 4$	4096 points $Z_i = [-50 : 14.28 : 50]$ $i = 1, 2, 3, 4$

In Fig. 7.2 the Korns P10 test function is plotted for $-50 \leq Z_i \leq 50$ with $i = 1, 3$, $Z_2 = 0$ and $Z_4 = 1$.

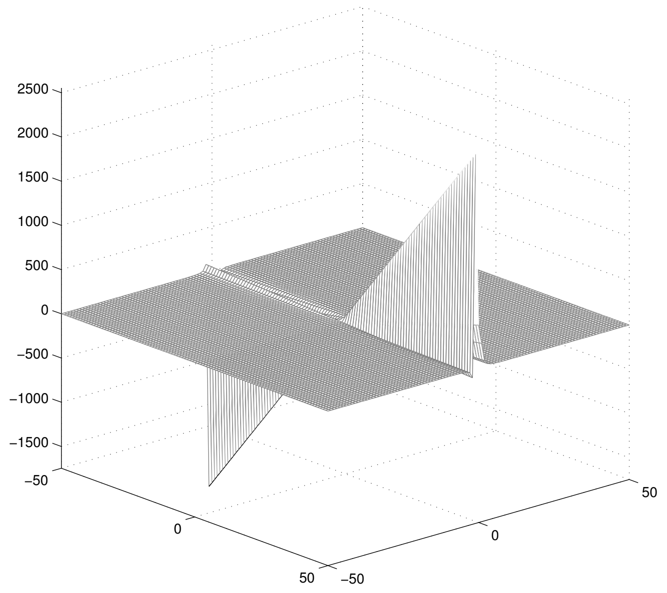


FIGURE 7.2: KornsP10 function plotted for $Z_2 = 0$ and $Z_4 = 1$

7.3 Experiments

Eight different experiments were performed for each test function using as baseline genetic programming implementation the *omegalim* version described in Section 5.3.0.6, Chapter 5. In three of them factorisation bonus was enabled, and in two of these three the editing strategies "Ed" and "Ed2" superimposed to the factorisation approach. These experiments were then repeated adding $shift$ to the functional primitives. To assess the effect of the new strategies on the baseline genetic programming implementation, in the following sections the results are compared to the ones returned by *omegalim* and *omegalim_shift* implementations. The experiments performed and the naming convention used is reported in Table 7.2.

TABLE 7.2: Plan of experiments to assess the effect of factorisation bonus and editing strategies Ed and Ed2

No.	experiment name	features enabled
1	omegalim	
2	omegalim_shift	shift primitive
3	omegalim_F	factorisation bonus
4	omegalim_shift_F	factorisation bonus, shift primitive
5	omegalim_Ed_F	factorisation bonus, Ed
6	omegalim_shift_Ed_F	factorisation bonus, Ed, shift primitive
7	omegalim_Ed2_F	factorisation bonus, Ed2
8	omegalim_shift_Ed2_F	factorisation bonus, Ed2, shift primitive

7.3.1 Kotanchek test problem

The same GP parameters listed in Table 5.16 of Section 5.3.5.1, Chapter 5 were used. The pathologies of the individuals generated by the different implementations on building and validation data sets are reported in Table 7.3.

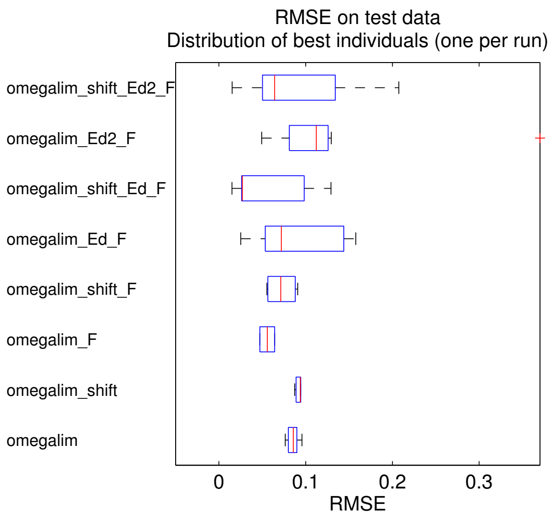
TABLE 7.3: Kotanchek test problem: pathologies on building and validation data sets

Kotanchek	Building data set				Validation data set			
			RMSE				RMSE	
	%∞	%bad	median	IQR	%∞	%bad	median	IQR
omegalim	0	0	6.155E-02	1.529E-02	0	0	8.571E-02	9.749E-03
omegalim_shift	0	0	5.754E-02	2.055e-02	70	0	9.391e-02	5.297e-03
omegalim_F	0	0	4.296E-02	2.856E-02	80	0	5.575E-02	1.696E-02
omegalim_shift_F	0	0	5.354E-02	3.063E-02	60	0	7.129E-02	3.159E-02
omegalim_Ed_F	0	0	5.072E-02	2.116E-02	30	0	7.202E-02	9.050E-02
omegalim_shift_Ed_F	0	0	5.157E-02	6.323E-02	40	0	2.716E-02	7.207e-02
omegalim_Ed2_F	0	0	6.259E-02	4.151E-02	30	0	1.122E-01	4.462E-02
omegalim_shift_Ed2_F	0	0	5.438E-02	3.454E-02	20	0	6.418E-02	8.386E-02

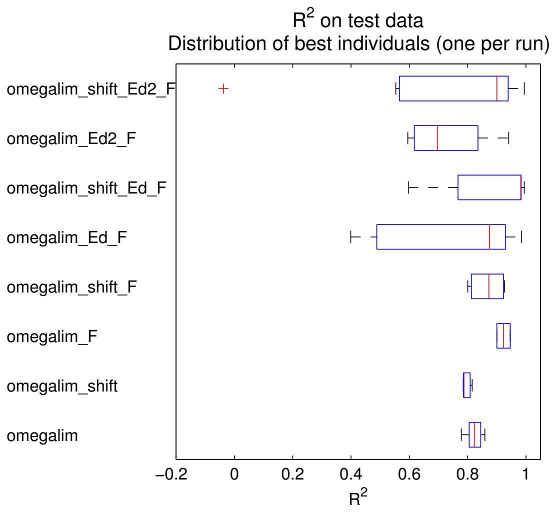
Omegalim implementation appears the most robust, in the sense that all the best individuals generated from each run are defined on the validation data set. The isolated use of the factorisation bonus appears to reduce the generalisation ability of the generated individuals, performance that is however improved by the superimposition of Ed and Ed2 editing strategies.

The boxplots of $RMSE$ and R^2 reported in Fig. 7.3 show an interesting effect of the factorisation bonus.

All the experiments where the factorisation bonus was enabled returned a best individual with lower RMSE than the baseline *omegalim* and *omegalim_shift* implementations. In particular, a higher RMSE (and R^2) interquartile range is observed in the experiments featuring the combined use of factorisation bonus and editing strategies Ed or Ed2. There



(A) RMSE



(B) R^2

FIGURE 7.3: Kotanchek test problem: RMSE and R^2 boxplots. Individuals with RMSE>100 as well as undefined ones are not included

are also a few differences in the median of the RMSE (and R^2) distributions, particularly evident for the experiment *omegalim_shift_Ed_F*. These differences however are not recognsed as statistically significant by Kruskal-Wallis test on the validation data set (p-value=0.30422)[1].

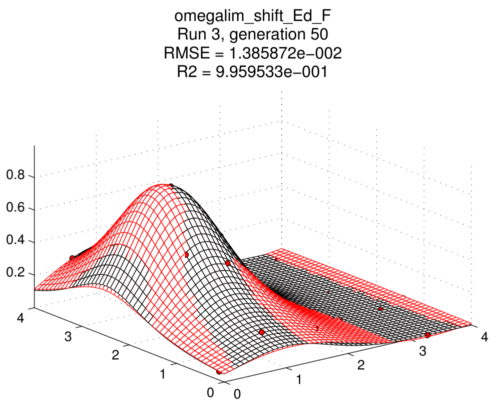
The expression of the overall best model, generated by *omegalim_shift_Ed_F*, is reported below:

$$\tilde{f}(z_1, z_2) = -0.0103772 + (-36.3242\,\mathrm{z}_1 - 48.3108 + 15.5212\,\mathrm{z}_1\mathrm{z}_1)^2/$$
$$((40.8610\,\mathrm{z}_1 - 90.0379 - 25.8195\,\mathrm{z}_1\mathrm{z}_1)^2 + (-69.1777\,\mathrm{z}_2 - 0.256008\,\mathrm{z}_2\mathrm{z}_1 + 17403.4)^2) \tag{7.3}$$

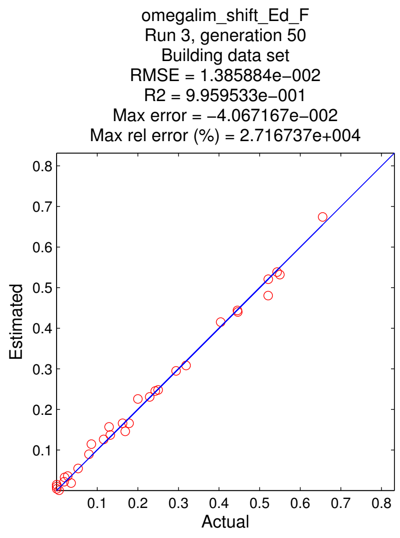
The effect of the factorisation bonus can be appreciated comparing the expression reported above with Eq. 5.25, expression of the best one returned by *omegalim* (Section 5.3.5.1, Chapter 5). The model in Eq. 7.3 is plotted superimposed to the original Kotanchek test function in Fig. 7.4A, together with the actual vs. estimated response plots on building and validation data sets (Fig. 7.4B and Fig. 7.4C).

The superior generalisation ability of the best model generated by *omegalim_shift_Ed_F* (Eq. 7.3) compared to the best one generated by *omegalim* can be ascribed to the correct identification of Kotanchek function mathematical structure. In this case the factorisation

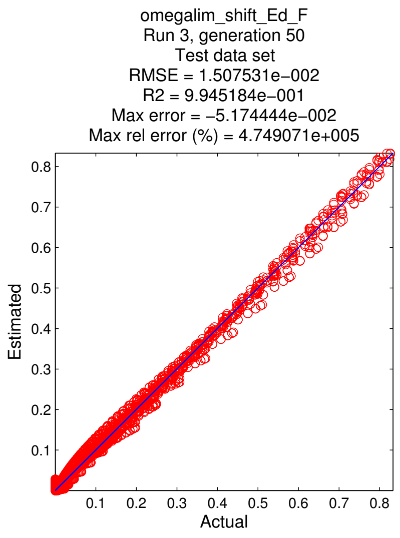
[1]The ANOVA test agrees with such conclusion, with p-value = 0.3326.



(A) Comparison between GP generated model (in red) and Kotanchek function (in black)



(B) Performance on training data set



(C) Performance on test data set

FIGURE 7.4: Kotanchek test problem: best individual generated by *omegalim_shift_Ed_F*

bonus successfully avoided the generation of a linear combination of terms, something that instead happened with *omegalim* as shown by Eq. 5.25 in Section 5.3.5.1, Chapter 5.

The analysis of the average size of archive individuals against the generations, plotted in Fig. 7.5, reveals that the use of the factorisation bonus does not consistently affect the average size of the individuals in the archive.

An interesting effect that is linked to the use of factorisation bonus is the runtime reduction, as shown in Fig. 7.6. Runtime reduction can not be completely ascribed to the reduced size of the individuals, as it has been shown that no consistent effect is produced by the factorisation bonus on the average size of the archive individuals. So probably the factorisation bonus affects computational overhead indirectly, making parameters tuning

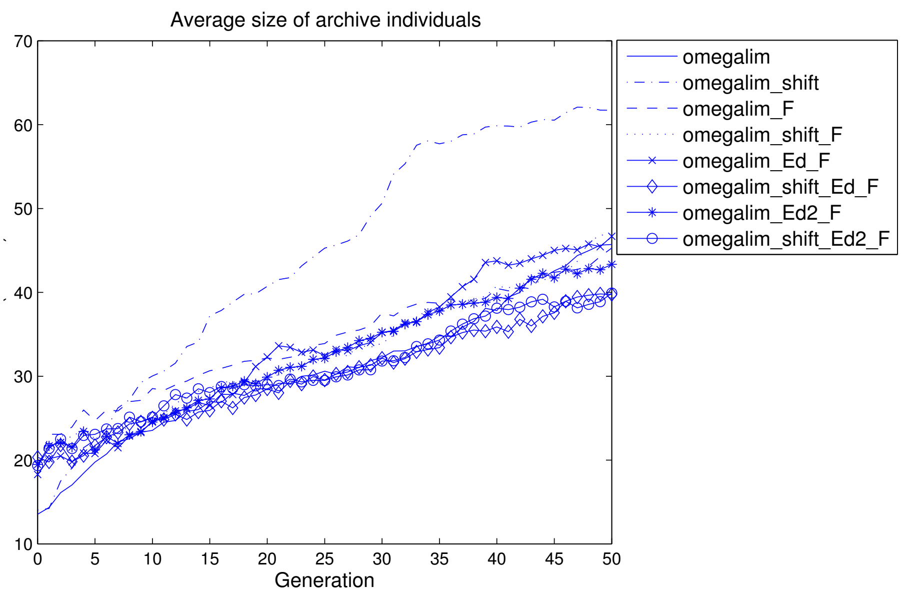


FIGURE 7.5: Kotanchek test problem: average size trend within the archive

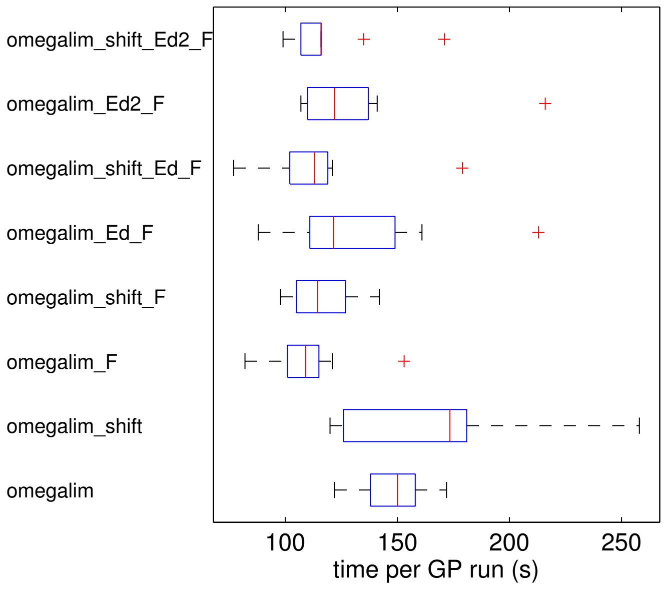


FIGURE 7.6: Kotanchek test problem: boxplots of best individuals' runtime

less expensive, maybe thanks to the better mathematical structure of the individuals it allows for.

7.3.2 Salustowicz test problem

All the experiments were performed using the same parameters listed in Table 5.19 of Section 5.3.5.2, Chapter 5. The pathologies the individuals generated by the different implementations suffer from on building and validation data sets are reported in Table 7.4.

TABLE 7.4: Salustowicz test problem: pathologies on building and validation data sets

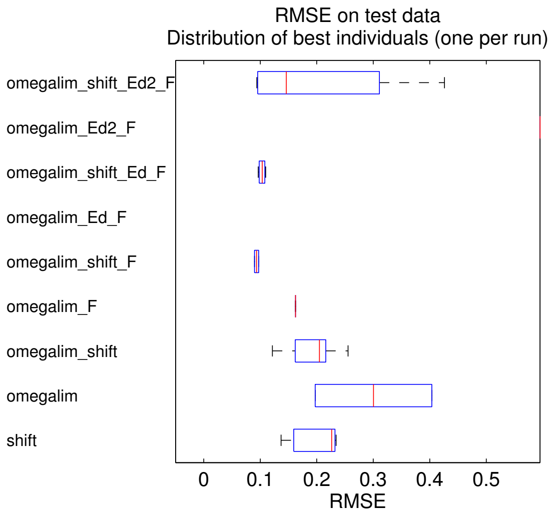
Kotanchek	Building data set				Validation data set			
			RMSE				RMSE	
	%∞	%bad	median	IQR	%∞	%bad	median	IQR
shift	0	0	1.200E-01	1.199E-01	60	10	2.263E-01	7.309E-02
omegalim	0	0	1.702E-01	7.376E-02	80	0	3.004E-01	2.065E-01
omegalim_shift	0	0	1.757E-01	6.902E-02	30	0	2.047E-01	5.408E-02
omegalim_F	0	0	1.655E-01	6.948E-02	90	0	1.624E-01	0.000E+00
omegalim_shift_F	0	0	7.803E-02	4.500E-02	80	0	9.346E-02	7.170E-03
omegalim_Ed_F	0	0	1.217E-01	9.306E-02	100	0	n/a	n/a
omegalim_shift_Ed_F	0	0	1.034E-01	2.984E-02	70	0	1.034E-01	9.872E-03
omegalim_Ed2_F	0	0	1.218E-01	7.111E-02	90	0	5.951E-01	0.000E+00
omegalim_shift_Ed2_F	0	0	9.864E-02	6.237E-02	50	10	1.462E-01	2.156E-01

The reduced number of individuals defined on the validation data set is not unexpected, as similar generalisation issues were observed in the experiments described in Section 5.3.5.2, Chapter 5. *Omegalim_shift* and *omegalim_shift_Ed2_F* are somewhat more robust than the other implementations, whereas *omegalim_Ed_F* gives the worst results with no individuals defined on the validation data set. Despite most of the individuals generated using the factorisation bonus are not defined on the validation set, the individuals that are indeed defined show in some cases higher accuracy than the ones produced by the baseline experiments *shift*, *omegalim* and *omegalim_shift*, as shown by the RMSE and R^2 boxplots in Fig. 7.7. This is the case of *omegalim_shift_F*, *omegalim_shift_Ed_F* and *omegalim_shift_Ed2_F*. The consistently positive effect of the *shift* primitive on accuracy has also to be acknowledged.

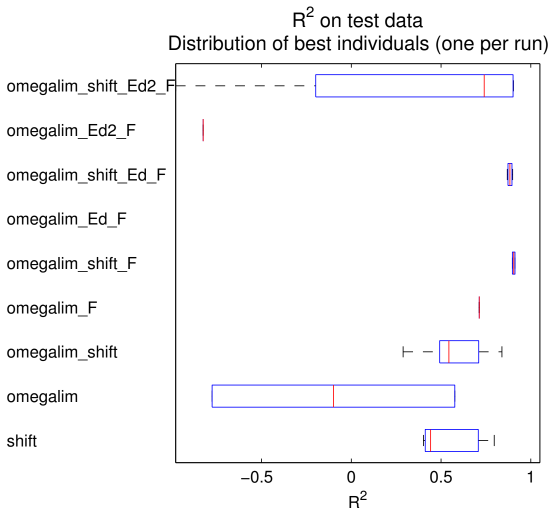
However, according to the p-value returned by Kruskal-Wallis test on the validation data set (0.099677) there is no significant evidence of a difference in the medians of the samples produced by the various experiments. Curiously, the ANOVA test supports the opposite conclusion (p-value = 0.0080031).

The best model was generated by *omegalim_shift_F*. Its expression is reported below:

$$\begin{aligned}\tilde{f}(\mathrm{z}_1) = &-0.000189778+\\ &\frac{113.724\,\sin(-1.61627\,\mathrm{z}_1)+116.524\,\sin(1.63860\,\mathrm{z}_1)}{-62.2514+39.5659\,\mathrm{z}_1-7.07145\,\mathrm{z}_1\,\mathrm{z}_1}\end{aligned} \tag{7.4}$$



(A) RMSE



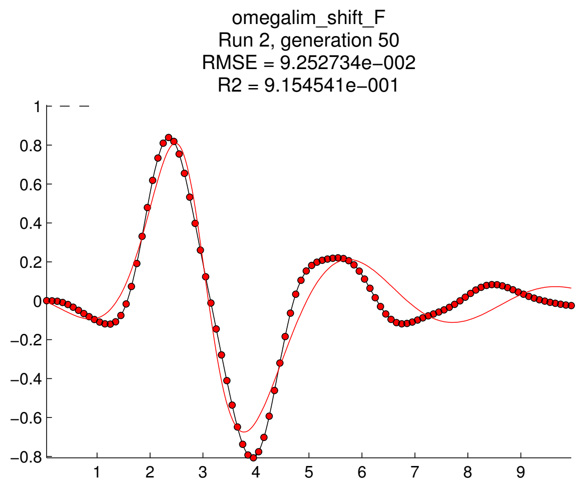
(B) R^2

FIGURE 7.7: Salustowicz test problem: RMSE and R^2 boxplots. Individuals with RMSE>100 as well as undefined ones are not included

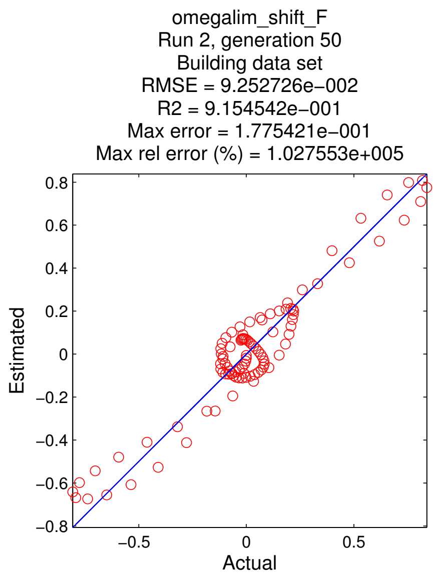
The expression reported above should be compared to Eq. 5.26, best individual returned by *shift* (see Section 5.3.5.2, Chapter 5), to appreciate the difference in size.

The returned model in Eq. 7.4 is superimposed to Salustowicz function plot in Fig. 7.8A. In Fig. 7.8B and 7.8C are shown the actual responses versus the responses estimated using the best model generated by *omegalim_shift_F* (Eq. 7.4).

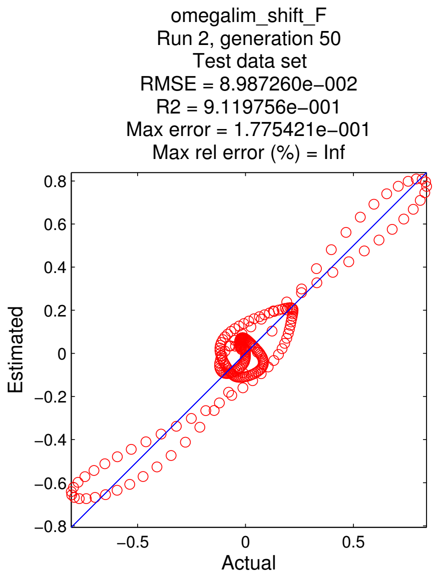
No consistent effect on the average size of archive individuals and on runtime is achieved using the factorisation bonus, with or without the editing strategies, as shown by the plots of the average archive individuals' size as a function of generation and of runtime boxplots in Fig. 7.9.



(A) Comparison between GP generated model (in red) and Salustowicz function (in black)

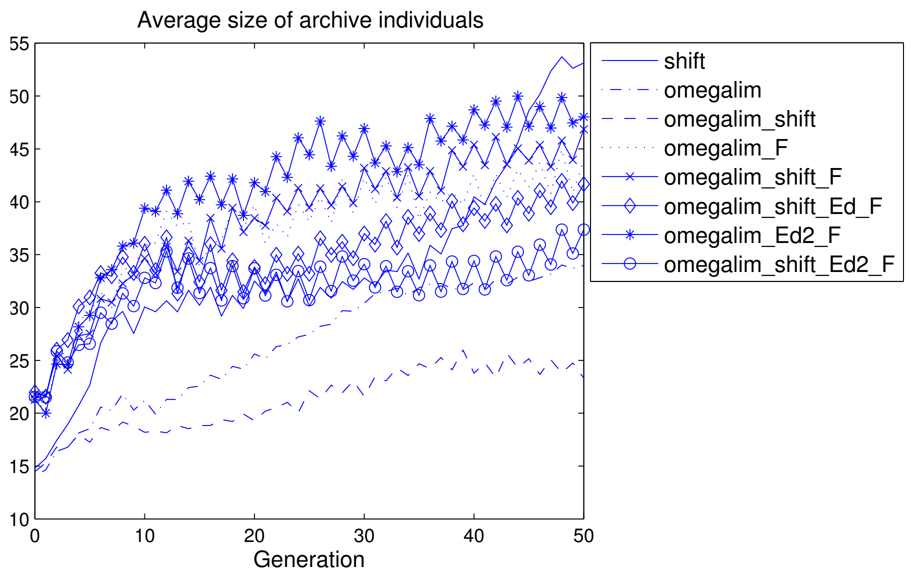


(B) Performance on training data set

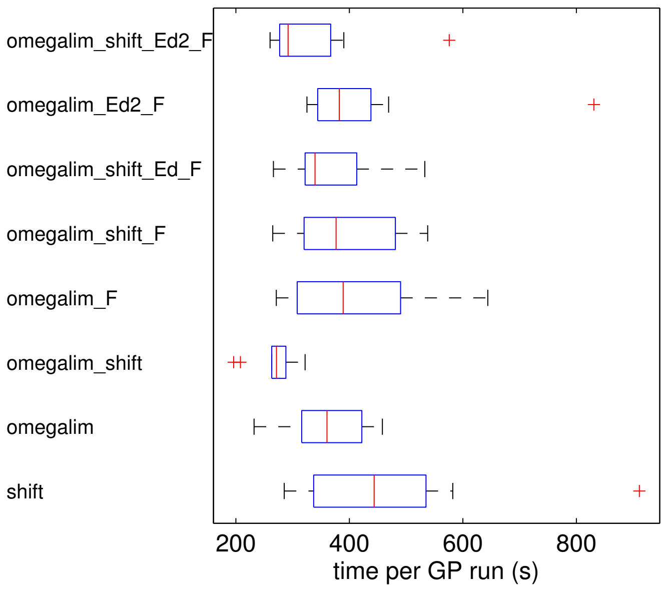


(C) Performance on test data set

FIGURE 7.8: Salustowicz test problem: best individual generated



(A) Size evolution



(B) Runtime

FIGURE 7.9: Salustowicz test problem: average size trend within the archive and boxplots of best individuals' runtime

7.3.3 RatPol2D test problem

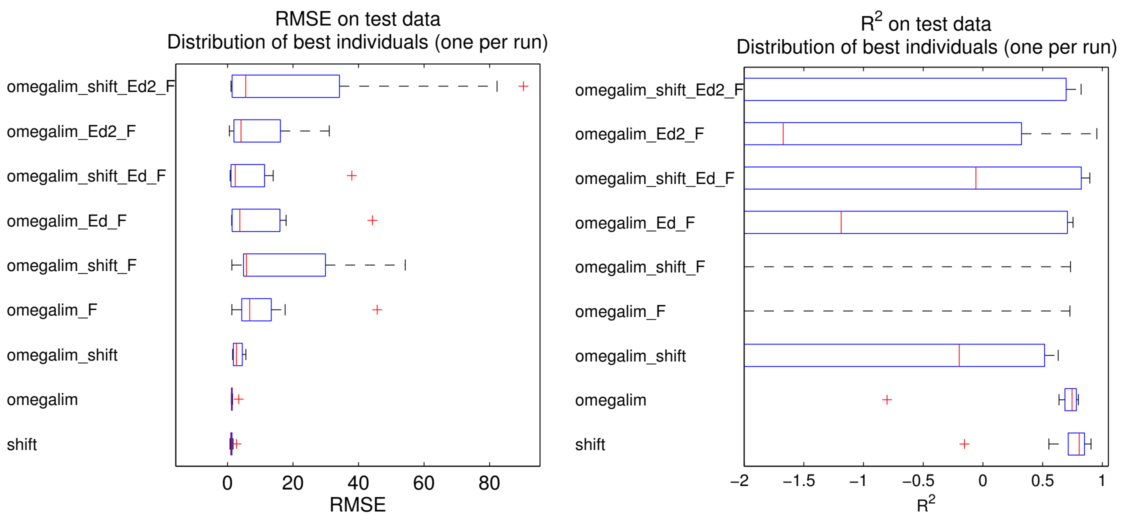
The same GP parameters listed in Table 5.21 Section 5.3.5.3, Chapter 5 were used for the experiments. The pathologies of the individuals generated by the tested implementations on building and validation data sets are reported in Table 7.5.

TABLE 7.5: RatPol2D test problem: pathologies on building and validation data sets

RatPol2D	Building data set				Validation data set			
			RMSE				RMSE	
	%∞	%bad	median	IQR	%∞	%bad	median	IQR
shift	0	0	4.279E-01	3.367E-01	0	10	1.117E+00	3.586E-01
omegalim	0	0	4.864E-01	2.994E-01	0	10	1.279E+00	2.357E-01
omegalim_shift	0	0	2.275E-01	3.505E-01	0	60	2.685E+00	2.736E+00
omegalim_F	0	0	5.514E-01	9.525E-02	0	20	6.752E+00	9.020E+00
omegalim_shift_F	0	0	6.358E-01	2.216E-01	0	10	5.752E+00	2.498E+01
omegalim_Ed_F	0	0	6.460E-01	1.286E-01	0	10	3.756E+00	1.460E+01
omegalim_shift_Ed_F	0	0	5.196E-01	2.242E-01	0	20	2.354E+00	1.022E+01
omegalim_Ed2_F	0	0	5.480E-01	3.363E-01	0	20	4.131E+00	1.418E+01
omegalim_shift_Ed2_F	0	0	6.819E-01	4.039E-01	0	0	5.537E+00	3.277E+01

No particular differences in the number of undefined individuals on the validation data set were recorded, with the exception of the implementation *omegalim_shift* for which 6 individuals out of 10 are undefined on the validation data set.

The boxplots of $RMSE$ and R^2 samples are reported in Fig. 7.10.



(A) RMSE (B) R^2

FIGURE 7.10: RatPol2D test problem: RMSE and R^2 boxplots. Individuals with RMSE>100 as well as undefined ones are not included

Omegalim and *shift* appear the most robust implementations, as they show far smaller $RMSE$ and R^2 interquartile ranges on both building and validation data sets. However, their exploration ability proves slightly less effective than *omegalim_shift_Ed_F* and *omegalim_Ed2_F*, implementation that returned the best individual. According to the p-value returned by the Kruskal-Wallis test (0.00099349) there is evidence of significative difference between $RMSE$ medians[2]. The p-values resulting by pairwise comparison of the $RMSE$ samples reported in Table 7.6 confirm that *omegalim*, *shift* have statistically better median than the other implementations except *omegalim_shift_Ed_F*.

TABLE 7.6: RatPol2D test case: *p*-values resulting from pairwise comparison using Wilcoxon rank sum test. To compare two implementations, start from the row with the name of the first one; then read along the row until the column with the second implementation's name is found. If a white box is reached, keep reading down the column until the row with the second implementation's name is found

RatPol2D	median	shift	omegalim	omegalim_shift	omegalim_F	omegalim_shift_F	omegalim_Ed_F	omegalim_shift_Ed_F	omegalim_Ed2_F	omegalim_shift_Ed2_F
shift	1.117E+00									
omegalim	1.279E+00	0.06								
omegalim_shift	2.685E+00	0.02	0.02							
omegalim_F	6.752E+00	0.00	0.00	0.15						
omegalim_shift_F	5.752E+00	0.00	0.00	0.11	1.00					
omegalim_Ed_F	3.756E+00	0.00	0.02	1.00	0.42	0.19				
omegalim_shift_Ed_F	2.354E+00	0.20	0.81	0.93	0.23	0.14	0.28			
omegalim_Ed2_F	4.131E+00	0.02	0.11	0.68	0.28	0.11	1.00	0.72		
omegalim_shift_Ed2_F	5.537E+00	0.00	0.01	0.45	0.90	0.84	0.60	0.27	0.41	

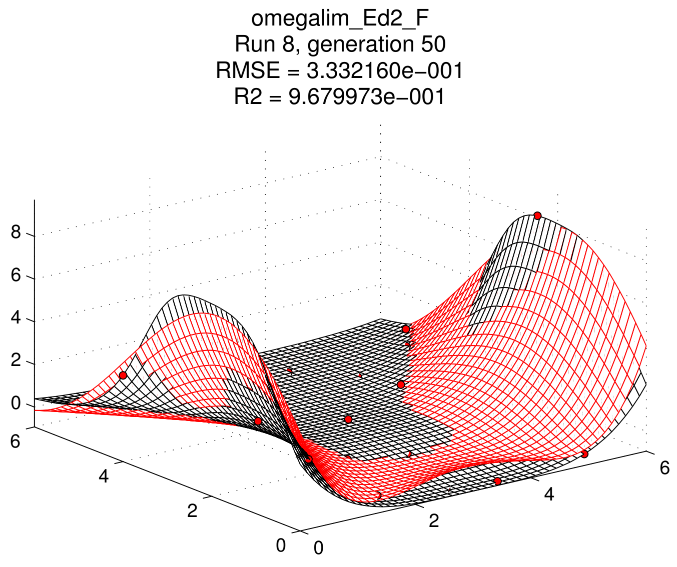
The best model was generated by *omegalim_Ed2_F* and is reported below:

$$\begin{aligned}\tilde{f}(z_1, z_2) = {}& -0.352041 + (532.069 - 72.9607\,\mathrm{z}_2)/\\ & (598.117/(52.1197 + 8.27273\mathrm{z}_1\,\mathrm{z}_1 - 49.3048\,\mathrm{z}_1 - 5.24910\,\mathrm{z}_2\mathrm{z}_2 + 27.3669\,\mathrm{z}_2))^2\end{aligned} \tag{7.5}$$

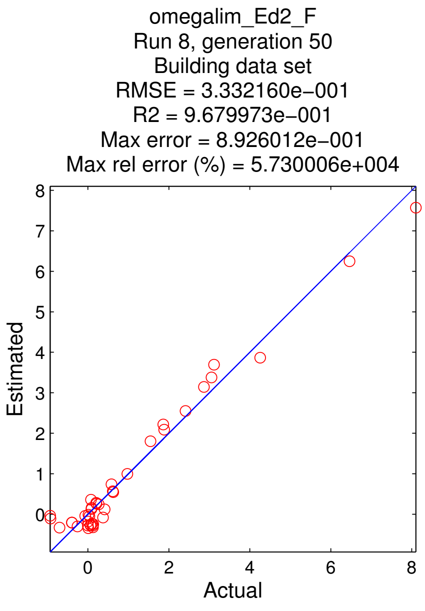
The model in Eq. 7.5 is plotted superimposed to RatPol2D test function in Fig. 7.11A. In Fig. 7.11B and 7.11C the actual reponse is plotted versus the response approximated by the model on building and validation data sets.

The evolution of the average size of archive individuals shown in Fig. 7.12A show an interesting behaviour: all the implementations in which the factorisation bonus was enabled experience a smaller average growth rate than the implementations where the bonus was not used (*shift*, *omegalim* and *omegalim_shift*). The reduction in size, due to the difference in the mathematical structure of the models, is easily appreciated if

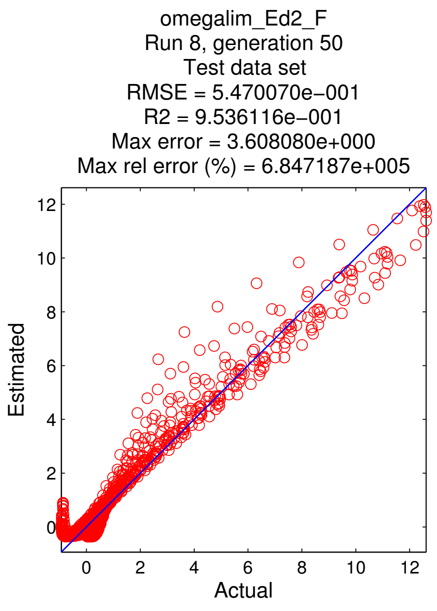
[2]ANOVA test on test data set does not back such conclusion (p-value = 0.14447).



(A) Comparison between GP generated model (in red) and RatPol2D function (in black)



(B) Performance on training data set

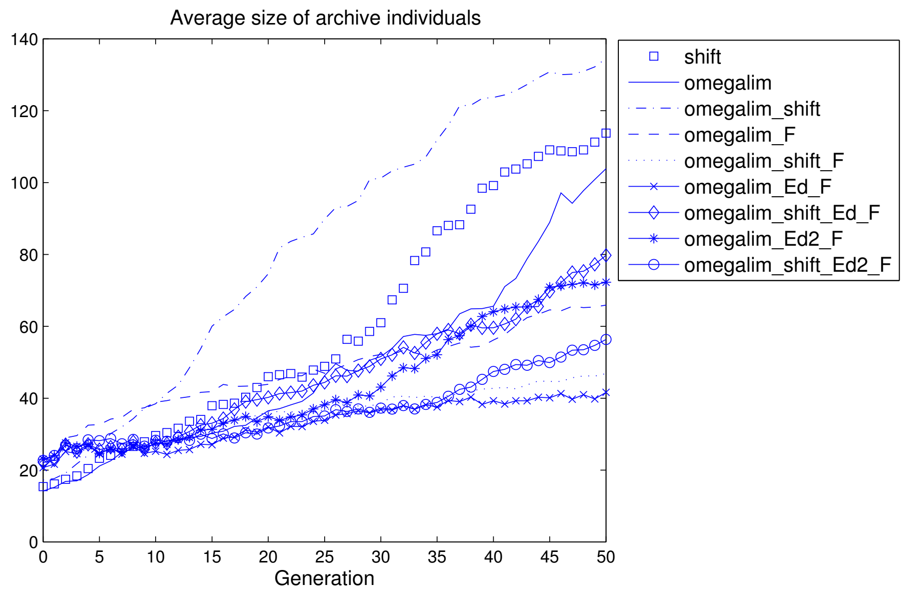


(C) Performance on test data set

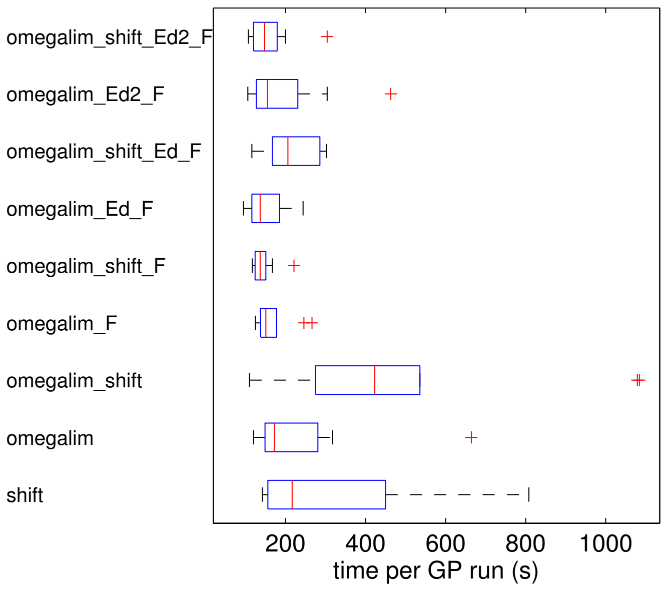
FIGURE 7.11: RatPol2D test problem: best individual generated

Eq. 7.5 is compared with Eq. 5.28 and Eq. 5.27 (best individuals returned respectively by *omegalim* and *shift*, see Section 5.3.5.3, Chapter 5).

In Fig. 7.12B runtime boxplots are shown for the experiments performed: the experiments where the factorisation bonus was enabled, no matter if editing was enabled, show on average a smaller runtime with respect to the ones not exploiting the bonus. Therefore, it seems that for this case, differently from the previous ones, there is a correlation between the use of factorisation bonus and a reduction of the computational cost of the evolution.



(A) Size evolution



(B) Runtime

FIGURE 7.12: RatPol2D test problem: average size trend within the archive and boxplots of best individuals' runtime

7.3.4 KornsP10 test problem

The main parameters used for the experiments are listed in Table 7.7.

TABLE 7.7: KornsP10 test problem: main parameters used in the experiments

Population size:	400
Generations:	200
Primitives:	+,-,*,/(protected), square, cube, sin, cos, tanh, exp, log
Fitness cases:	300

The pathologies the individuals generated by the different implementations suffer from on building and validation data sets are reported in Table 7.8.

TABLE 7.8: KornsP10 test problem: pathologies on building and validation data sets

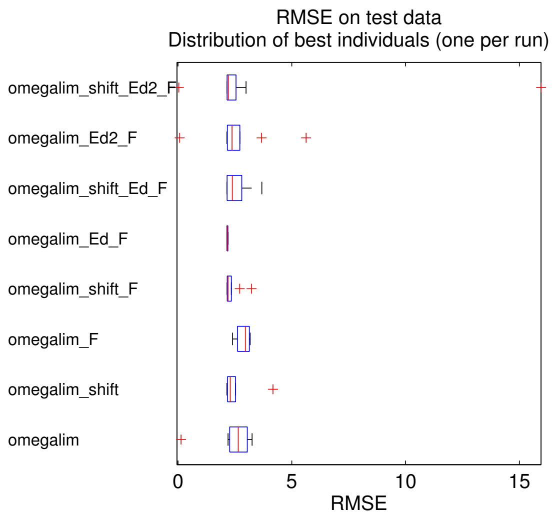
KornsP10	Building data set				Validation data set			
			RMSE				RMSE	
	%∞	%bad	median	IQR	%∞	%bad	median	IQR
omegalim	0	0	3.310	2.587	10	0	2.647E+00	7.774E-001
omegalim_shift	0	0	4.214	1.681	40	0	2.294E+00	3.574E-01
omegalim_F	0	0	4.960	1.656	60	0	2.963E+00	5.205E-01
omegalim_shift_F	0	0	3.283	2.039	10	0	2.198E+00	1.810E-01
omegalim_Ed_F	0	0	4.888	2.442	40	0	2.167E+00	4.234E-02
omegalim_shift_Ed_F	0	0	3.574	1.544	0	0	2.384E+00	6.483E-01
omegalim_Ed2_F	0	0	3.453	2.820	0	0	2.374E+00	5.507E-01
omegalim_shift_Ed2_F	0	0	4.601	1.142	10	0	2.214E+00	3.913E-01

The results shown in Table 7.8 reveal that the use of factorisation bonus alone increased the number of undefined individuals on the validation data set with respect to the baseline implementation *omegalim*. The combined use of the bonus with one of the editing strategies partly solved the problem.

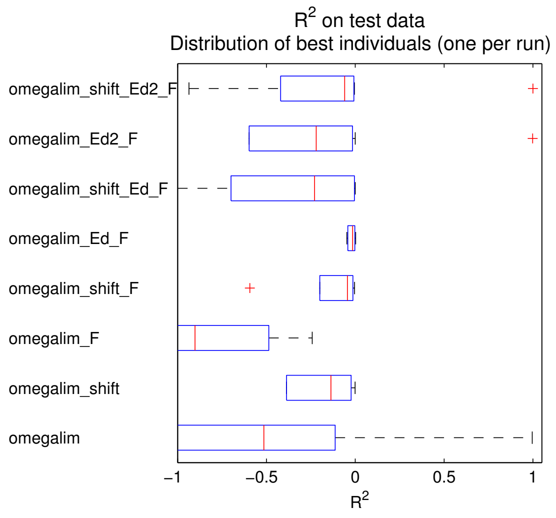
The boxplots of $RMSE$ and R^2 distributions referring to the validation data set are reported in Fig. 7.13.

The boxplots show that in general the quality of the best solutions generated by all implementations is quite poor, with the exception of the best three individuals generated by *omegalim*, *omegalim_Ed2_F* and *omegalim_shift_Ed2_F*. At a general level, the combined use of factorisation bonus and one of the editing strategies resulted in a slight improvement in the median of the $RMSE$ and R^2 distribution. Such improvement is however recognised as not statistically significant by Kruskal-Wallis test on the validation data set (p-value = 0.15249)[3].

[3]ANOVA tests backs this result with p-value = 0.89231.



(A) RMSE



(B) R^2

FIGURE 7.13: KornsP10 test problem: RMSE and R^2 boxplots. Individuals with RMSE>100 as well as undefined ones are not included

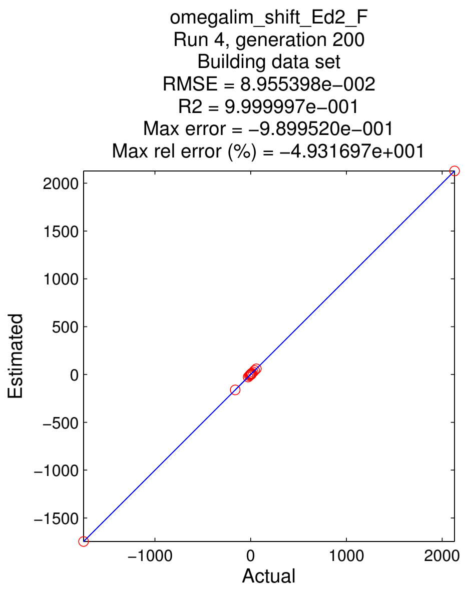
The expression of the best model generated by *omegalim_shift_Ed2_F* is reported below:

$$\begin{aligned}\tilde{f}(\mathrm{z}_1,\mathrm{z}_2,\mathrm{z}_3,\mathrm{z}_4) &= 0.809396 + (-36.6710\,\mathrm{z}_2)/\\ &(((((147.049\,\mathrm{z}_4\,\mathrm{z}_4 - 2.01659\,\mathrm{z}_4)^2) - (128.716\,\mathrm{z}_4 + (-222.307\,(\mathrm{z}_2/(\mathrm{z}_3\mathrm{z}_3)))))/ \\ &(-32.5875\,(263.633\,\mathrm{z}_2))) - (-150.732\,(\mathrm{z}_3/(\mathrm{z}_2/(\mathrm{z}_3(\mathrm{z}_3/-74.6861))))))\end{aligned} \tag{7.6}$$

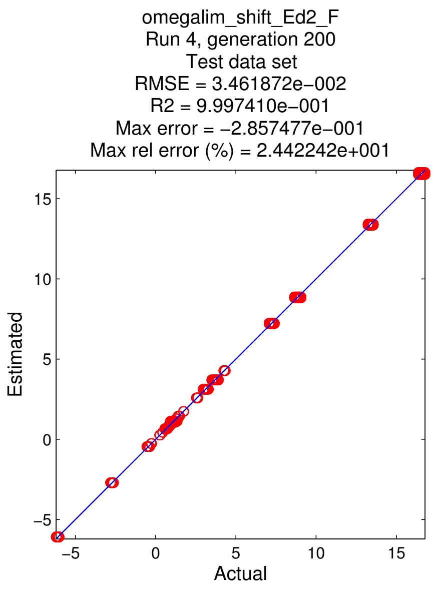
In Fig. 7.14B and Fig. 7.14C are shown the actual Korns P10 output values on the validation data set versus the responses estimated using the approximated model in Eq. 7.6 on the same points.

In Fig. 7.15A are shown the average size growth of archive individuals during the evolution, while in Fig. 7.15B boxplots show the runtime distribution for each experiment.

A first look at the size growth reveals that the combined use of factorisation bonus and one of the two editing strategies results in a smaller growth rate, reducing the average size of the archive individuals by approximately a third by the end of evolution with respect to the implementations where these two strategies were not enabled. A correlation between the combined use of factorisation bonus with an editing strategy and the computational cost reduction is confirmed by boxplots in Fig. 7.15B. In particular it can be noted that *omegalim_Ed_F*, the implementation with the smallest average size of archive individuals, is also the one having the lowest runtime median.

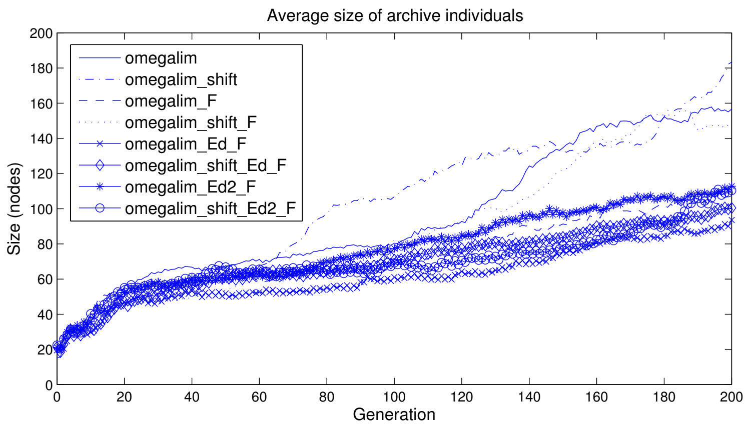


(A) Performance on training data set

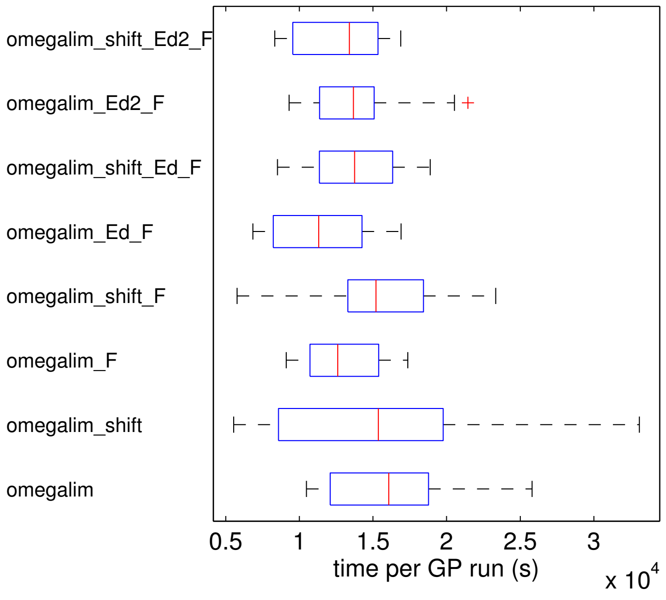


(B) Performance on test data set

FIGURE 7.14: KornsP10 test problem: best individual generated



(A) Size evolution



(B) Runtime

FIGURE 7.15: KornsP10 test problem: average size trend within the archive and boxplots of best individuals' runtime

7.4 Comparison with Gaussian Processes

The experiments run so far show that the generation of a global explicit expression via genetic means has undoubtedly a cost, in terms both of computing time and generalisation ability. To assess if such cost is bearable and to check HyGP maturity, the same regression tests have been performed using an implicit technique. Gaussian Processes (Section 1.2.4.3, Chapter 1) have been considered for the comparison, in particular the MatLab implementation written by Rasmussen and Nickisch, freely downloadable from Rasmussen (2006) (see also Rasmussen and Williams (2006), Rasmussen and Nickish (2010)). Precious help in the selection of the covariance functions (see Section 1.2.4.3, Chapter 1) and in results validation was provided by Mr. James Lloyd from the University of Cambridge.

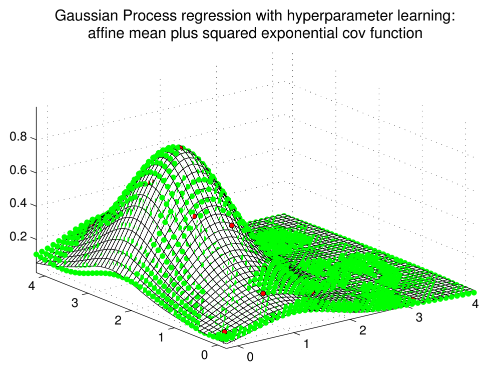
In the following the settings used and the coefficient of determination R^2 obtained considering the pointwise mean output of the gaussian process for each regression test function are reported. The gaussian process hyperparameters, required to define the covariance function (see Section 1.2.4.3, Chapter 1) and the mean of the process, have been optimised through a conjugate gradient method as implemented in Rasmussen (2006), following the Bayesian model selection approach (Rasmussen and Williams 2006, p. 108).

7.4.1 Kotanchek test problem

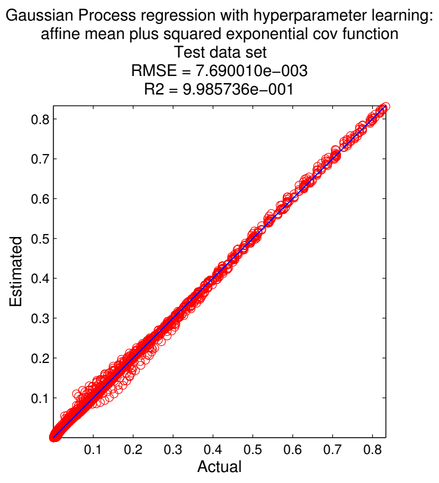
The Gaussian Process was set up using an affine mean and a squared exponential covariance function (Rasmussen and Williams 2006, p. 83) with equal length scales for the two input variables (called function *covSEiso* in Rasmussen (2006)). A maximum of 100 function evaluations was imposed for the conjugate gradient optimiser. The building data set (40 points) and validation data set (2025 points) detailed in Table 5.2, Chapter 5 were used. In Fig. 7.16 the pointwise mean output generated by the conditioned Gaussian Process are shown superimposed to the true underlying function. The actual vs. estimated output plot is also shown. The coefficient of determination returned was $R^2 = 0.9985736$.

7.4.2 Salustowicz test problem

As for the previous case, the Gaussian Process was set up using an affine mean and a squared exponential covariance function with equal length scales for the two input variables (*covSEiso*). A maximum of 100 function evaluations was imposed for the conjugate gradient optimiser. The building data set (100 points) and validation data set (221 points)



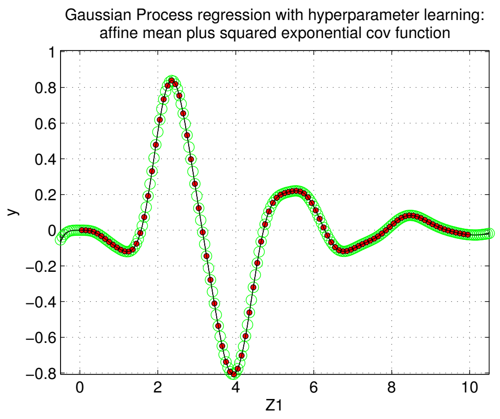
(A) Comparison between Gaussian Process reponses (in green) and Kotanchek function (in black)



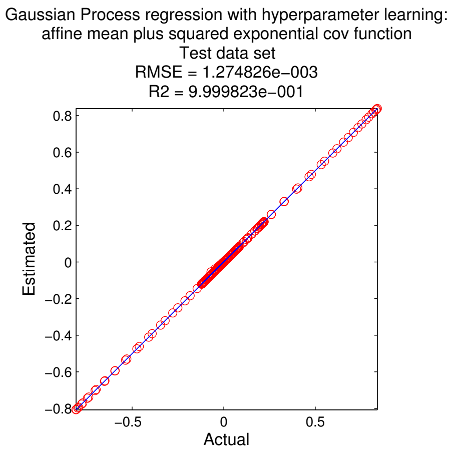
(B) Performance on test data set

FIGURE 7.16: Kotanchek test problem: Gaussian Process response

detailed in Table 5.2, Chapter 5 were used. The pointwise mean generated by the Gaussian Process are shown in Fig. 7.17 superimposed to Salustowicz function. The coefficient of determination achieved was $R^2 = 0.9999823$.



(A) Comparison between Gaussian Process reponses (in green) and Salustowicz function (in black)

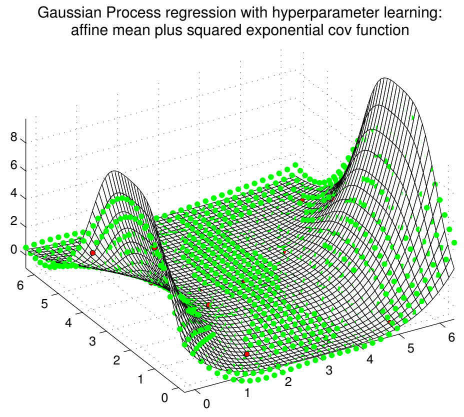


(B) Performance on test data set

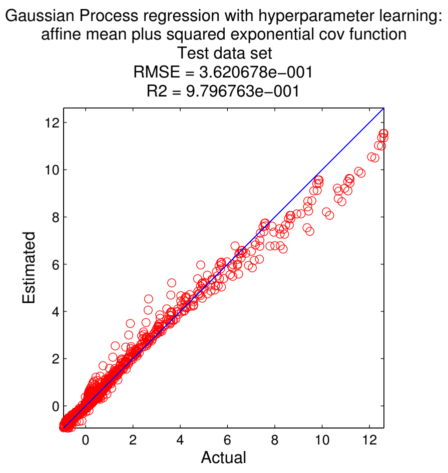
FIGURE 7.17: Salustowicz test problem: Gaussian Process response

7.4.3 RatPol2D test problem

The Gaussian Process was set up using an affine mean and a squared exponential covariance function with equal length scales for the two input variables (*covSEiso*). A maximum of 100 function evaluations was imposed for the conjugate gradient optimiser. The building data set (40 points) and validation data set (1156 points) detailed in Table 5.2, Chapter 5 were used. In Fig. 7.18 the pointwise mean response generated by the conditioned Gaussian Process and the actual vs. estimated output plot are shown. The resulting coefficient of determination was $R^2 = 0.9796763$.



(A) Comparison between Gaussian Process reponses (in green) and RatPol2D function (in black)



(B) Performance on test data set

FIGURE 7.18: RatPol2D test problem: Gaussian Process response

7.4.4 Korns P10 test problem

Due to the presence of asymptotes Korns P10 function appeared very challenging for gaussian processes (Snelson et al. 2004). For this reason different covariance functions were used, and also a training data set purged of points characterised by excessive outputs (outliers, defined as points whose output absolute value is larger than 50) was attempted. A maximum of 1000 function evaluations was imposed for the conjugate gradient hyperparameter optimiser. The building data set (300 points) and the validation data set (4096 points) defined in Table 7.1 were used.

The tests with different functional structures for the mean and the covariance functions led to coefficient of determination R^2, defined on the pointwise mean output of the

Gaussian Process, reported in Table 7.9. In addition to the already introduced squared exponential covariance function with uniform length scales (*covSEiso*), the *rational quadratic covariance function with automatic relevance determination (ARD) distance measure* was used (Rasmussen and Williams 2006, p. 106). This covariance structure features length scales independently tuned and is indicated as *covRQard*. The *GPSS MAE* covariance functions were found automatically using the Gaussian Process Structure Search code written by Mr. Lloyd (Lloyd 2012).

TABLE 7.9: KornsP10 test problem: obtained R^2 for different covariance functions. In bold the best R^2 returned.

Mean	Covariance function	Outliers removed?	R^2	Hyperparameters
Zero	constant + covSEiso	NO	-0.293146	3
Zero	constant + covRQard	NO	-0.293338	7
Zero	constant + covSEiso	YES	**-0.002457**	3
Zero	constant + covRQard	YES	-1.746069	7
	GPSS MAE level 2	YES	-2.124714	6
	GPSS MAE level 3	YES	-2.007970	8
Affine	constant + covRQard	YES	-1.941058	12

7.4.5 Jet pump problem: comparison of HyGP and Gaussian Process

An additional test was performed on a real life industrial problem, in particular the modelling of the jet pump entrained flow rate described in Section 6.5, Chapter 6. As noted previously, the practical need of a user-friendly metamodel motivated the adoption of HyGP. The test with Gaussian Process allows to check if there could have been some gain in accuracy had the final user not expressively asked for an explicit metamodel. As in Korns P10 case, a few different sets of mean and covariance function were tested, reported in Table 7.10 together with the resulting $RMSE$ and R^2 computed on the same validation data set introduced in Section 6.5.

TABLE 7.10: Jet pump test problem: obtained R^2 for different covariance functions. In bold the best R^2 returned.

Mean	Covariance function	$RMSE$	R^2	Hyperparameters
Zero mean	covSEiso	16.63110	0.9950176	2
Zero mean	covSEard	14.76978	0.9960705	4
Zero mean	constant + covRQard	13.60173	0.9966674	6
Affine mean	constant +covRQard	13.59126	**0.9966725**	10

7.4.6 Discussion of the results

The Korns P10 test case confirms that Gaussian processes are not ideal when the response to be modelled varies over many orders of magnitude, at least in their conventional form (Snelson et al. 2004). There are some strategies, like *warping* (Rasmussen and Williams 2006, p. 92) (Snelson et al. 2004) that allow to cope with these scenarios, but these approaches are not standard and anyway imply searching for a well-behaved monotonic warping function.

In these scenarios, the search for mathematical structures performed by HyGP appears as a valid approach, as not constrained nor degraded by the smoothness properties or the scale of variation of the underlying function. The comparison between the R^2 obtained on the four benchmarks and the real industrial case by the two techniques shows indeed that not only is HyGP effective when no assumptions on the smoothness of the response can be made but it can also provide metamodels of quality comparable to gaussian process-generated metamodels. In fact, for Korns P10 function and the jet pump problem HyGP outperformed Gaussian Processes.

TABLE 7.11: RMSE and R^2 errors computed on validation data set for the given test problems. Best results for each test case are highlighted in bold

Test function	Gaussian Process R^2	HyGP R^2
Salustowicz	**0.999982**	0.911976
Kotanchek	**0.998574**	0.994518
RatPol2D	**0.979676**	0.953612
Korns_P10	-0.002457	**0.999741**
Jet pump	0.996672	**0.997664**

It may be objected that the wrong selection of the covariance function or the presence of multiple local marginal likelihood suboptima in the space of hyperparameters can be blamed for the not optimal performance of Gaussian Processes in some of the previous cases (Rasmussen and Williams 2006, p. 115). These claims are grounded, but are intrinsic to modelling with Gaussian Processes. Tackling these issues implies facing the problem of how to lead a search for a structural part of a model, and therefore restates the importance and validity of population-based regression techniques, able to robustly search for the optimal mathematical structures and coefficients of a model. In this sense, genetically-based search strategies for the automatic selection of Gaussian Processes covariance functions could be an interesting and promising direction of research.

7.5 Conclusion

In this chapter a strategy to protect against selection pressure highly factorised metamodel made of highly non-linear functions has been introduced, called “factorisation bonus”. The experiments performed show that the use of the factorisation approach leads to better approximations of the test functions than *omegalim* and *omegalim_shift*, implementations that already represent an improvement with respect to the basic HyGP engine presented in Chapter 5. The improvement in accuracy can be ascribed to the ability of the factorisation bonus of protecting mathematical structures that otherwise would get lost during the early stages of the evolution.

Factorisation appears then similar to the ALPS strategy (Hornby 2006) in the sense that not mature but highly factorised individuals, deemed promising, are granted a special protection during selection that allows them to compete with mature individuals, likely to have high accuracy due to overfitting. A confirmation of that is the fact that all the best models produced by *omegalim* and *omegalim_shift*, where the factorisation bonus was not enabled, are linear combinations of terms or even polynomials. The use of the factorisation bonus leads instead always to the generation of highly factorised and compact expressions. The bonus also seems to increase the exploration ability of GP, probably as a result of the protection granted to factorised individuals, but at the same time affects negatively robustness. Unfortunately a clear and consistent difference in the effect of the two editing strategies did not emerge, although the importance of the two strategies can not be denied considering that in three cases out of four the best model was generated exploiting the factorisation bonus in combination with Ed or Ed2.

The lack of statistical evidence of the better performance of the implementations featuring the factorisation bonus is due to the reduced robustness with respect to the baseline implementations, *omegalim* and *omegalim_shift*.

The experiments also show that for Kotanchek, RatPol2D and Korns P10 test functions the use of the factorisation bonus allows for a general reduction in the median runtime. In this sense the bonus is able to reduce the computational cost of the evolution. For Salustowicz test function the outcome, in terms of accuracy of the best individual and average runtime, is contradictory. The best model returned represents an improvement with respect to the baseline implementations. However, in this case the factorisation bonus failed to protect the correct structure, although multiplication was among the factorising primitives under control.

The wrong identification of the Salustowicz function mathematical structure proves that the factorisation bonus approach, although already effective, needs further exploration to express its full potential. So far it has appeared biased towards the protection of only divisions. A more effective strategy should be developed, probably establishing a hierarchy of functional primitives and define tournaments based on this hierarchy. The ALPS approach may provide useful inspiration. To reduce the number of HyGP parameters, it would also be useful to develop a strategy to eliminate or automatically identify the best values of the numerical constants in the equation defining the factorisation bonus (Eq. 7.1).

In the final part of the chapter the comparison of HyGP with Gaussian Processes, an implicit metamodelling technique, has been described. Regression tests on benchmark functions and on a real-life industrial problem have proved that HyGP can produce metamodels of accuracy comparable or even better than Gaussian Process metamodels. In one case the better performance of HyGP is due to the fact that genetic programming does not require the function to be modelled to be smooth or varying on a limited range. Differently from Gaussian Process, HyGP returns the explicit expression of the metamodel.

Chapter 8

Conclusions and recommendations for future work

8.1 Summary of contributions

The research presented in this thesis has focused on the problem of generating accurate mathematical metamodels from data through an evolutionary technique called "genetic programming". The ability to produce a metamodel through data-driven techniques is key for the fast exploration of the behaviour of a system, as such analyses would be far more expensive or demanding if approached using more traditional methodologies, like direct simulations or analytical models.

A review of the most common metamodelling techniques nowadays used has shown that genetic programming has unique features. Some techniques, like kriging (gaussian processes), are able to generate accurate approximations, but not in an explicit form and under strict conditions regarding smoothness and the scale of output variation. Others, like response surface methodologies, are instead explicit, but require prior knowledge of the relationship between inputs and outputs to better fit the data. Genetic programming has appeared as the only metamodelling technique able to produce relatively accurate and explicit metamodels without requiring any prior knowledge on the mathematical relationship between inputs and outputs.

Much attention has been dedicated to the analysis of the main factors that affect negatively genetic programming performances, like introns, bloat and lack of variability. The review of the different solutions advanced by researchers has allowed to identify a minimum set of requirements that increases the likelihood of generating accurate and compact metamodels at a reasonable computational cost.

The tree-based, hybrid genetic programming implementation developed during the research activity, named HyGP, has stemmed from this analysis. Besides basic counter-measures to bloat, in HyGP a strategy has been implemented for handling metamodels' numerical coefficients not conventionally adopted in GP. The strategy, based on the deterministic insertion and tuning of numerical coefficients in HyGP individuals, allows to unambiguously assess if the low accuracy of a metamodel is ascribable to a wrong mathematical structure or to wrong values of numerical coefficients. Although the approach had already been introduced by Alvarez (2000), it has been rigorously tested here for the first time on a few benchmark regression problems, showing that it performs better than the standard genetic programming paradigm (see Section 5.3.3, Chapter 5). The hybrid approach has also been further improved introducing a mechanism to limit the overfitting issues caused by $sine$ and $cosine$ primitives (see Sections 5.2.2 and 5.3.0.6, Chapter 5).

A completely new strategy has been developed to increase the generalisation ability of the metamodels evolved by HyGP (see Section 5.4, Chapter 5). The strategy allows the user to define through a set of inequality constraints the desired behaviour of the ideal solution on a set of points in the design space. The survival and evolution of a metamodel with enhanced generalisation ability is encouraged penalising the probability of the metamodels not satisfying the specified inequalities of being selected for the reproduction stage. The main advantage of the approach is that it does not necessarily require extra computational cost for data gathering, as prior knowledge of the system under study can be exploited to define the desired behaviour.

To test HyGP performances, a few experiments on benchmark cases and real-life industrial metamodelling and optimisation problems have been performed (see Chapter 6). The results obtained show that HyGP is able to produce metamodels of accuracy comparable to and in some scenarios better than other metamodelling techniques: Table 8.1 summarises the regression and optimisation problems where HyGP outperformed established techniques.

The main advantage of HyGP metamodels is that they are returned as a symbolic expression, so they can be used for inexpensive sensitivity analysis through Monte Carlo methods and in general be processed using analytical tools. An additional approach has been introduced to reduce the size and increase the accuracy of HyGP metamodels, based

TABLE 8.1: List of regression and optimisation problems where HyGP outperformed other metamodelling techniques. The technique in bold returned the most accurate metamodel (regression) or better solution (optimisation) for the problem defined in first column.

Problem	Techniques		Section and Chapter
Regression of Rosenbrock function	**HyGP**	PCE	Section 5.5.2, Chapter 5
Regression of Kotanchek function	**HyGP**	PCE	Section 5.5.3, Chapter 5
10-bar truss optimisation	**HyGP**	MLSM	Section 5.6, Chapter 5
Fuselage barrel optimisation	**HyGP**	MLSM	Section 6.7.3, Chapter 5
Regression of Korns P10 function	**HyGP**	Gaussian Pr.	Section 7.4.4, Chapter 7
Jet pump regression problem	**HyGP**	Gaussian Pr.	Section 7.4.5, Chapter 7

on the protection of individuals containing highly non-linear functional nodes at low depths (Chapter 7).

8.2 Practical impact of the research

The research on genetic programming methodologies has been carried out with a twofold aim. On the one hand, ways to improve the traditional GP paradigm have been explored, obtaining the results described in the previous section. On the other hand, much work has been dedicated to make HyGP a user friendly and efficient metamodelling tool, which can be used with minimum effort by users without programming skills. This has been considered a necessary requirement for HyGP successful application in industry.

This vision has motivated the efforts to develop a few C++ parallelised versions of HyGP to make the most efficient use of the computing architecture available to the user (see Appendix A), in particular clusters. The range of applications described in Chapter 6 can give a hint about which aerospace companies have shown active interest for HyGP. The metamodels that were specifically generated and accepted for industrial use are:

1. Model of chromate diffusion process (Section 6.4, Chapter 6)

2. Model of supersonic jet pump entrained flow rate (Section 6.5, Chapter 6)

3. Model of a lattice aircraft fuselage barrel (Section 6.7, Chapter 6)

Finally, the Matlab scripts written to post-process the results produced by HyGP have also a considerable practical importance. They allow to compare HyGP experiments using statistical methods like Kruskal-Wallis test and Wilcoxon rank sum test, and therefore they can be used to test further improvements to the current HyGP implementation.

8.3 Recommendations on future work

At the end of this long dive into genetic programming, different recommendations for future work emerge, not only limited to HyGP. The general review of GP research has allowed to discover that definitions of major GP phenomena do not exist or, if they exist, they are not accepted by the whole GP community. Such definitions are important to compare different GP implementations and so they are vital for research in GP. General consensus is needed in the following areas:

- *a common definition of bloat should be introduced.*

  Some authors define bloat as code growth without any fitness improvement, other as disproportionate code growth with respect to fitness improvement. Also, some refer to the average size of GP individuals, other to the size of the best individual. In this sense, a unique parameter that comparing size and fitness was able to detect bloat would be useful to implement anti-bloat strategies. A possible idea of a "bloat quantifier" could be:

  $$I_b(t) = -\frac{\Delta\bar{e}/\bar{e}(t-1)}{\Delta\bar{s}/\bar{s}(t-1)} \tag{8.1}$$

  where $I_b(t)$ is the bloat "quantification" at generation t, $\Delta\bar{e}$ is the variation of the average error ($RMSE$ for example) from generation $t-1$ to t, $\bar{e}(t-1)$ is the average error of the population at generation $t-1$. At denominator, $\Delta\bar{s}$ is the variation of the average size in the population and $\bar{s}(t-1)$ is the average size at generation $t-1$.

- *a common set of test cases (benchmarks) should be defined.*

  The identification of common test cases would allow a direct comparison of GP implementations. The functions described in Affenzeller and Wagner (2004, p. 260-1) or the ones used by Vladislavleva (2008) could be selected for this purpose. For the particular difficulty of the problem, also the 10-bar truss optimisation problem described in Section 5.6, Chapter 5 could be a good candidate.

- *a common measure of computational cost should be agreed on.*

  A few definitions of the computational cost associated to a GP run have been presented in Section 3.1.6, Chapter 3. The number of node evaluations seems to be the best parameter as it takes into account also the effects of bloat.

- the importance of dimensionality analysis apparently has gone unnoticed in genetic programming, apart from the attempt made by Keijzer and Babovic (1999). Researchers have tried to increase the quality of GP metamodels reducing the number of input parameters of the problem (Vladislavleva 2008) through sensitivity analysis or clustering them in metavariables (Singh et al. 2007). However, if the physical nature of the input variables is known, the Buckingham theorem (Focken 1953) provides a powerful way to reformulate the metamodelling problem that does not require to neglect the effect of any variable. The approach is a promising and relatively unexplored way to improve the quality GP metamodels and is worth further attention in the future.

More precise recommendations can be given about possible ways to improve HyGP performances and its use of computational resources. A set of potential improvements easily implementable are:

- expressional complexity (Smits and Kotanchek (2004, p. 289) and Vladislavleva (2008, p. 90)) could be used instead of individual size to reduce bloat (see also Section 4.3.1, Chapter 4).

- all subtrees that compose each individual could be considered during evaluation and selection to increase population variability, as done in Smits and Kotanchek (2004) and Vladislavleva (2008). As all these subtrees are in any case evaluated, this approach would not result in an increase in RAM usage or computational overhead, while increasing the chance of finding a good model.

- the user should be given the possibility to insert in the initial population externally generated models, for example established analytical (fundamental) models ("warm start"). This would allow to bias the evolution using validated models and possibly to generalise them. In MacLean et al. (2005), Yin et al. (2007) this approach is mentioned as a promising way to boost the search. Such strategy is also used effectively in Schmidt and Lipson (2009a). An algorithm to convert a text expression into a syntax tree accepted by HyGP is required to implement this approach.

- the current parallel HyGP implementation could be modified according to the suggestions provided in Section A.4, Chapter A to make a more efficient use of parallel architectures. Furthermore, HyGP use could be further simplified if all the software performing metamodel generation, validation and post-processing was integrated in a unique framework. Python programming language seems to have optimal features to merge all these functionalities, as links with C++ code can be easily implemented.

The following are instead a list of promising research areas which deserve further exploration:

- despite the strategy introduced in Section 5.4, Chapter 5 can be effectively used to exploit prior knowledge of the system under analysis to improve the generalisation ability of HyGP metamodels, further research is needed to understand how it affects the evolution and how it can be improved. It would be interesting to explore the effect of other formulations of the penalisation term F_5.

- it would be interesting to analyse the phenomenon of bloat in HyGP, to see if in the absence of any size penalisation the same quadratic or sub-quadratic increase in size and linear in depth reported for standard GP (see Section 4.4.1, Chapter 4) would be observed. That would allow to assess the effect of the parameter insertion algorithm on bloat (see Section 5.2.2, Chapter 5). Furthermore, it would be interesting to study the combined effect of adaptive parsimony pressure or Pareto selection and the hybrid formulation featured by HyGP. Combined implementations of this kind have not been found in literature.

- finding a way to define "species" with common mathematical features (for example rational functions, polynomials, etc ...) inside a GP population would allow for the application of selection strategies commonly used in GAs, like *niching* or *restricted mating*. Clustering could then be used to select individuals from either different or similar species, to direct the exploration to interesting subregions of the design space.

- strategies to automate the selection of the optimal parameters both in selection and in genetic operations could reduce the bias imposed by the initial GP settings defined by the user. These strategies could also adapt the values of the different parameter according to the evolution stage (youth, maturity or stagnation). For example tournament size, elite size, crossover and mutation rates could dynamically change during the evolution.

- reducing the computational cost of evaluation and parameter tuning in HyGP is critical for its application to high dimensional problems, due to the curse of dimensionality. As programming style can affect performances, memory usage in HyGP code can probably be further optimised. A way to monitor RAM usage during the run could help implement more efficient ways to handle HyGP populations.

Appendix A

HyGP implementation details

This appendix focuses instead on HyGP implementation details, from the selection of the programming language to HyGP code structure. The simple strategy used to parallelise the code is presented and further development towards a more efficient parallelism are described.

A.1 Programming language selection

In general, a programming language should be chosen for its intrinsic capabilities to efficiently perform the operations described in the algorithm to be implemented. Many different and conflicting criteria make the identification of the optimal programming language for genetic programming algorithm implementation very challenging, and this maybe explains why many different languages have been used by GP researchers through the years.

First of all, execution speed is key, due to the high number of fitness evaluations: low-level programming languages may dramatically reduce execution time but at the cost of a more difficult implementation and often of a reduced portability.

Secondly, the programming language, the specific task the GP implementation aims to perform, and the GP representation[1] can be constrained by each other, as particular representation are suggested for particular tasks. For example, linear representation is often preferred for algorithm or control law generation, and in this case machine code is the best programming language in terms of speed[2].

[1]see Section 3.3, Chapter 2.
[2]see Section 3.3, Chapter 2.

Thirdly, portability is paramount. The possibility to develop, test and run a piece of software on the highest number of architectures without significant modifications is a necessary condition for its application on an industrial scale.

A.1.1 Machine code or object-oriented languages?

Table A.1 reports the results of a brief survey carried out before starting HyGP development to identify what programming languages had mostly been used for genetic programming implementations since the appearance of Koza's book (Koza 1992).

TABLE A.1: A list of a few genetic programming implementations, with corresponding programming language and year in which the original implementation was launched

Name	Progr. language	Author	Year
GP	LISP	Koza (1992)	1992
GPQuick	C++	Singleton (1993)	1993
AIM-GP (CGPS)	machine code	cited in Nordin et al. (1996 1999)	1994
Discipulus™	machine code	cited in Nordin et al. (1999) (www.rmltech.com)	
Code by Hollick et al.	C/C++	Hollick and Kuhlmann (1995)	1995
lil-gp	C	Zongker et al. (1996), Punch and Zongker (1998)	1995
SYSGP	C++	Brameier et al. (1998)	1998
Open BEAGLE	C++	Gagné and Parizeau (2002)	1999
Code by Alvarez	C++	Alvarez (2000)	2000
ParetoGP Toolbox	MatLab	cited in Vladislavleva (2008) Kordon and Lue (2004)	2001
HeuristicLab	C#, .NET	Wagner and Affenzeller (2002)	2002
ECJ	Java	Luke and Panait (2002a) Luke (2010)	2002
GPLab	MatLab	Silva (2003)	2003
TinyGP	Java, C	Poli et al. (2008)	2004
Code by Lew et al.	Java	Lew et al. (2006)	2006
Pyevolve	Python	Perone (2009)	2009
Eureqa	C++	Schmidt and Lipson (2009ab)	2009
HyGP	C++, Fortran	(this thesis)	2011

The range of programming languages as it appears from Table A.1 spans low-level programming code (machine code), higher-level programming languages, like C and LISP, and more abstract and object-oriented programming languages like C++, Java, Python and Matlab. No GP implementations in Fortran were found by the survey. Such a wide range of choice justifies a more detailed analysis on the advantages and weaknesses of different classes of programming languages for GP implementation.

A.1.1.1 Machine code

Low-level programming languages (for example, machine code or binary code) generally allow for high execution speed and optimised memory usage. These features are particularly important considering that a single GP run is generally computationally intensive and requires large amount of RAM. Nordin et al. (1999) reported that a speed-up of up to two order of magnitude can be achieved by machine code GP implementations with respect to similar GP implementations written in compiled programming languages (C, C++), as there is no need to convert high-level data structures to machine code during evaluation. Nordin et al. (1999) also observed that machine code allows for an efficient optimisation of the memory usage, which cannot be done by more abstract programming languages. AIM-GP, a GP implementation written in machine code, has been reported to perform extremely well on high-dimensional symbolic regression problems, with up to 40 input variables (Nordin et al. 1999, pag. 288).

The striking performances allowed by machine code however do not come without drawbacks. A first limitation is posed on the GP representation, as the only GP representation that can be handled or processed by the low-level operations on registers performed by machine code is the linear one. Flexibility is then somewhat penalised.

Secondly, in machine code GP the practical implementation of all the mechanisms required for fitness evaluation, genetic modification and primitives selection are so dependent on the target processor (Nordin et al. 1999, pag. 292) that a specific evolutionary engine has to be written for any particular target machine[3]. As a result, portability may be extremely limited and closure satisfaction may be a challenge, due to the complexity of correctly splitting the genotypes during mutation or crossover (Nordin et al. 1996 1999). Portability of the evolved program can be increased through machine code decompilation (Nordin et al. 1999, pag. 288-89), or the conversion in compilable C code of the best individual evolved by machine code GP (this is a feature of Discipulus™).

Thirdly, it should be reminded that machine code GP follows an imperative approach, and not a functional approach[4]. As a result, programs evolved by machine code GP are to be used like "black boxes", returning a certain set of outputs corresponding to the input provided by the user.

[3]for example, a version of machine code GP has been developed even for the Sony PlayStation (Nordin et al. 1999, pag. 281).

[4]see Section 3.3, Chapter 2.

Given the high execution speed, but, at the same time, the scarce portability, machine code is an optimal choice for writing linear GP implementations aiming at evolving algorithms or control laws in real time, on specific target machines and without the need of human intervention: for example, machine code GP has been successfully used for online control of autonomous robots (Nordin et al. 1999, pag. 297). However, the use of machine code as programming language appears less convenient for GP implementations performing symbolic regression tasks, as in such cases runtime reduction is not critical and the availability of an explicit mathematical expression may help data analysis and interpretation.

A.1.1.2 Higher level and object-oriented languages

Higher level programming languages like C or Fortran represent a good compromise between speed and portability. However, they still lack the versatility of object-oriented languages. The possibility to define *objects* and to establish a hierarchy among them (inheritance, polimorphism, etc. (Prata 2005)) undoubtedly simplifies the implementation of the GP algorithm (Barbosa and Bernardino 2011), which by its nature is organised on different independent levels (population, individual and fitness cases). The versatility granted by objects also allows more freedom in terms of GP representation: linear, tree or graph genotypes can easily be implemented.

If on the one hand the high level of abstraction and versatility of some object-oriented languages may ease GP implementation, on the other hand it may imply drastically slower GP execution speed. This is generally the case for GP software written in non-compiled or scripting languages like Python or MatLab. An indipendent symbolic regression test performed using data sampled from Rosenbrock function with a C++ and a Python GP implementation showed that the former runs at least two orders of magnitude faster than the latter. Similar conclusions about the use of Python for GP implementation were also reported by Perone (2009).

MatLab and any other proprietary programming languages introduce major obstacles to GP use at industrial scale, so they have not been considered.

A.1.2 Optimal selection for regression purposes

Among all programming languages described in the previous sections, C++ was recognised as the best compromise in terms of speed, versatility and portability. HyGP was then written in C++, although for computationally intensive operations Fortran routines were generally preferred. HyGP individuals' numerical coefficients[5] are indeed tuned by the Fortran SQP algorithm written by Madsen et al. (2002), called smoothly from C++ HyGP code.

The operating system selected for HyGP development, testing and standard use was Linux. The availability of open source editors, compilers and debuggers[6] played an important role in the decision (Vaughan-Nichols 2004), but far more important was the constantly increasing recognition of such operating system in high-performance computing sector. Fig. A.1 shows the composition of the world best 500 supercomputers (TOP500) in the period 1993-2010 in terms of operating system (Meuer et al. 1993): the percentage of supercomputers based on Linux grew from 0% to about 80% in approximately 12 years. As HyGP was conceived primarily as a research tool designed to be run on the widest range of high-performance computers, Linux appeared a correct choice. Moreover, Linux machines are widely used also in industry.

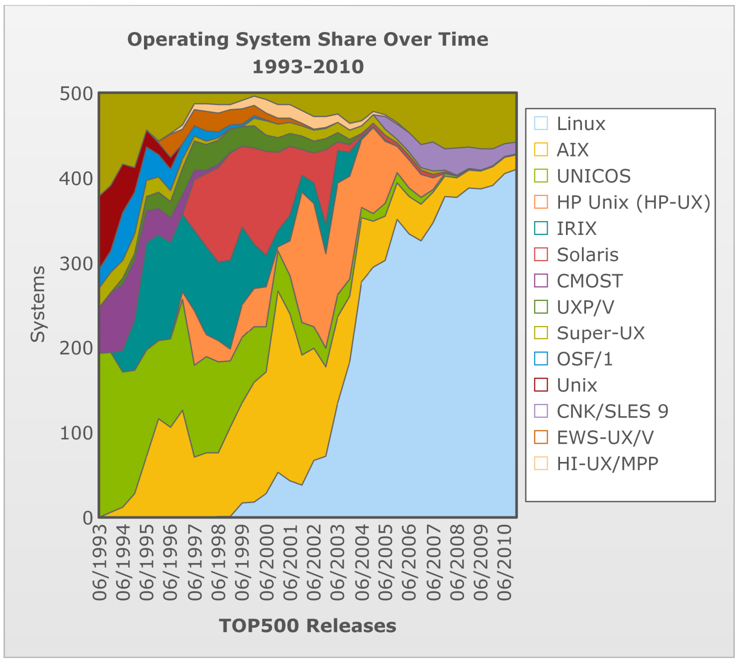


FIGURE A.1: Operating systems installed on the 500 most powerful supercomputers (period 1993-2010) (image reported under permission - http://www.top500.org).

[5]see Section 5.2.2, Chapter 5

[6]see for example compilers g++ and gfortran, debugger gdb and editor Eclipse IDE, all included or easily installable on any Linux distribution.

A.2 Basic structure of HyGP code

HyGP source code was developed trying to maximise the use of classes to enable code reuse and reduce code size, allow for easier testing and make future development easier. The main classes that were defined are:

- `RunParameters`: class that stores all the parameters required to set up a HyGP experiments, except the primitives (variables and functions) the building data set provided by the user or any additional data set to be used for problem specific knowledge exploitation (see Section 5.4, Chapter 5).
- `ProblemDefinition`: class that stores the primitives (variables and functions), the building data set provided by the user and any additional data set to be used for problem specific knowledge exploitation. A method for splitting the building data set and declaring which data subset has to be used for HyGP individual tuning and fitness evaluation is here implemented[7], as well as a simple method to compute statistical data on the building data set. All the functions required for variable initialisation are also contained in `ProblemDefinition`.
- `Reporter`: class containing all the methods used to print results to file. More information on the files produced as output can be found in Section B.3, Appendix B.
- `Population`: class containing the arrays of the addresses (pointer arrays) of the individuals that define the GP population. Instead of inserting and removing numerical coefficients from the individuals each generation, it was opted for keeping two lists of addresses, one corresponding to individuals without numerical coefficients and the other one to individuals with numerical coefficients. All the methods required for population random generation, population genetic modification (crossover and mutation), individual fitness evaluation, population sorting and termination criterion assessment were also implemented here.
- `Node`: base class defining the atomic entity composing a syntax tree. The class was adapted from Hollick and Kuhlmann (1995), as all the derived classes `Binary_Node`, `Unary_Node`, `Terminal_Var`, `Terminal_Const`. These derived classes are needed to allow syntax trees to represent mathematical expressions: `Binary_Node` is used to

[7]this method was used to apply the NestedDoE approach described in Section 5.3.0.5, Chapter 5.

code binary functions (FB), `Unary_Node` for unary functions (FU), `Terminal_Var` for variables (TV) and `Terminal_Const` for numerical coefficients (TC).

Each `Node` class stores pointers to other `Node` classes, so links between nodes can be established and syntax trees built. Each link has to be defined upwards and downwards: upwards to the "parent" node for which the current node is an argument, and downwards to define the arguments of the current node. For example `Binary_Node` has a pointer to the parent node and two pointers to the nodes defining the two arguments of its function. To ease legality checks in the syntax trees and to account for different numbers of downwards links, most of the methods for the base Node class are virtual (see polimorphism - (Prata 2005)), and specifically defined in the derived classes (for example the destructor or the node counter). The conventions followed for node numbering and for depth assignment in each syntax tree are shown in Fig. A.2.

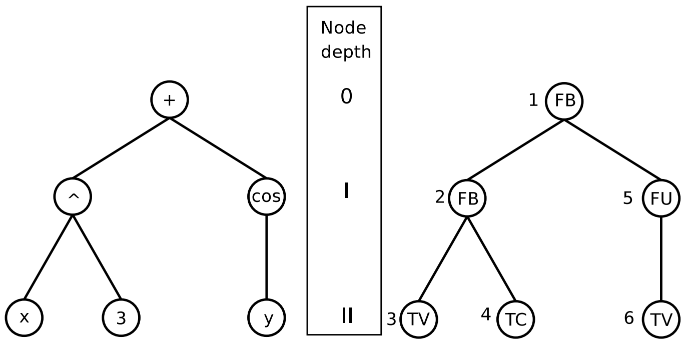


FIGURE A.2: Conventions for node numbering and depth assignment in HyGP syntax trees.

The top node of the tree, also called root node, is always assumed to be a binary function (`Binary_Node`). This assumpiton, inherited from Hollick and Kuhlmann (1995), does not reduce the variety of mathematical expressions that can be represented through a syntax tree[8].

Data acquisition from input file and numerical coefficient tuning were left to standard functions. More information on input file format can be found in Section B.1, Appendix B. A detailed flowchart with the operations performed and the commands used in HyGP main source file is provided in Fig. A.3-A.4 (to be compared with Fig. 5.1, which is the general HyGP algorithm).

[8]similar practical issues, which affect GP individuals genotype, are common in GP literature: for example in (Soule et al. 1996, p. 217) GP individuals' root node has to be a control statement.

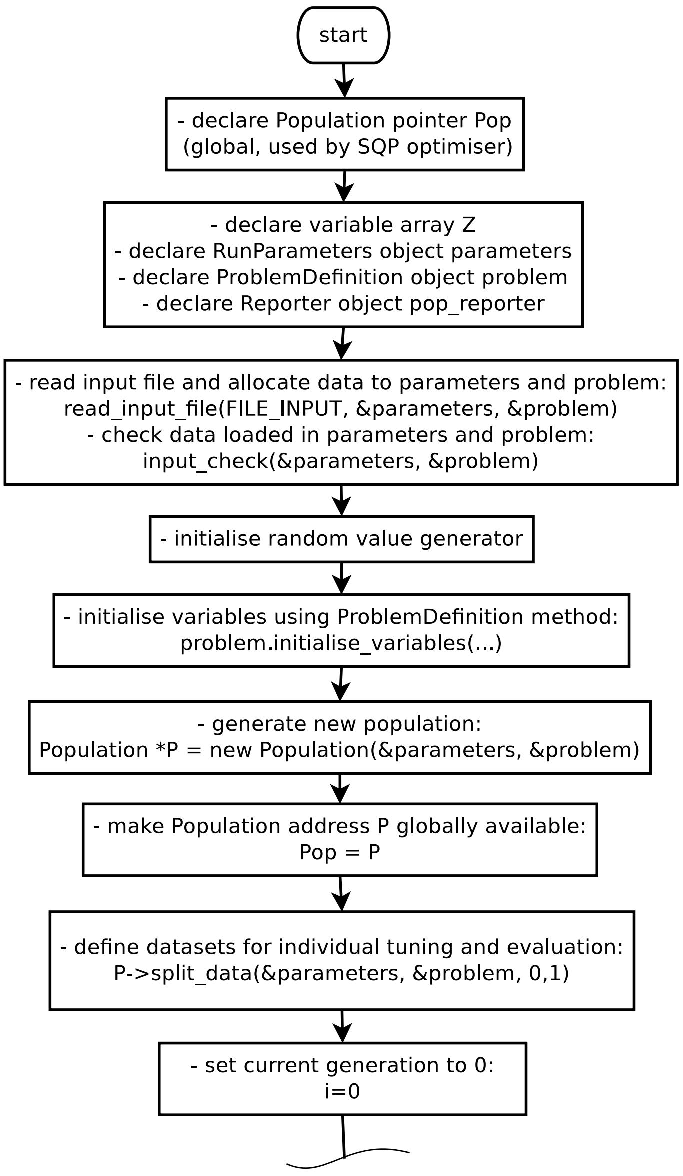

start
- declare Population pointer Pop
(global, used by SQP optimiser)
- declare variable array Z
- declare RunParameters object parameters
- declare ProblemDefinition object problem
- declare Reporter object pop_reporter
- read input file and allocate data to parameters and problem:
read_input_file(FILE_INPUT, &parameters, &problem)
- check data loaded in parameters and problem:
input_check(&parameters, &problem)
- initialise random value generator
- initialise variables using ProblemDefinition method:
problem.initialise_variables(...)
- generate new population:
Population *P = new Population(&parameters, &problem)
- make Population address P globally available:
Pop = P
- define datasets for individual tuning and evaluation:
P->split_data(&parameters, &problem, 0,1)
- set current generation to 0:
i=0

FIGURE A.3: HyGP flowchart - 1st part

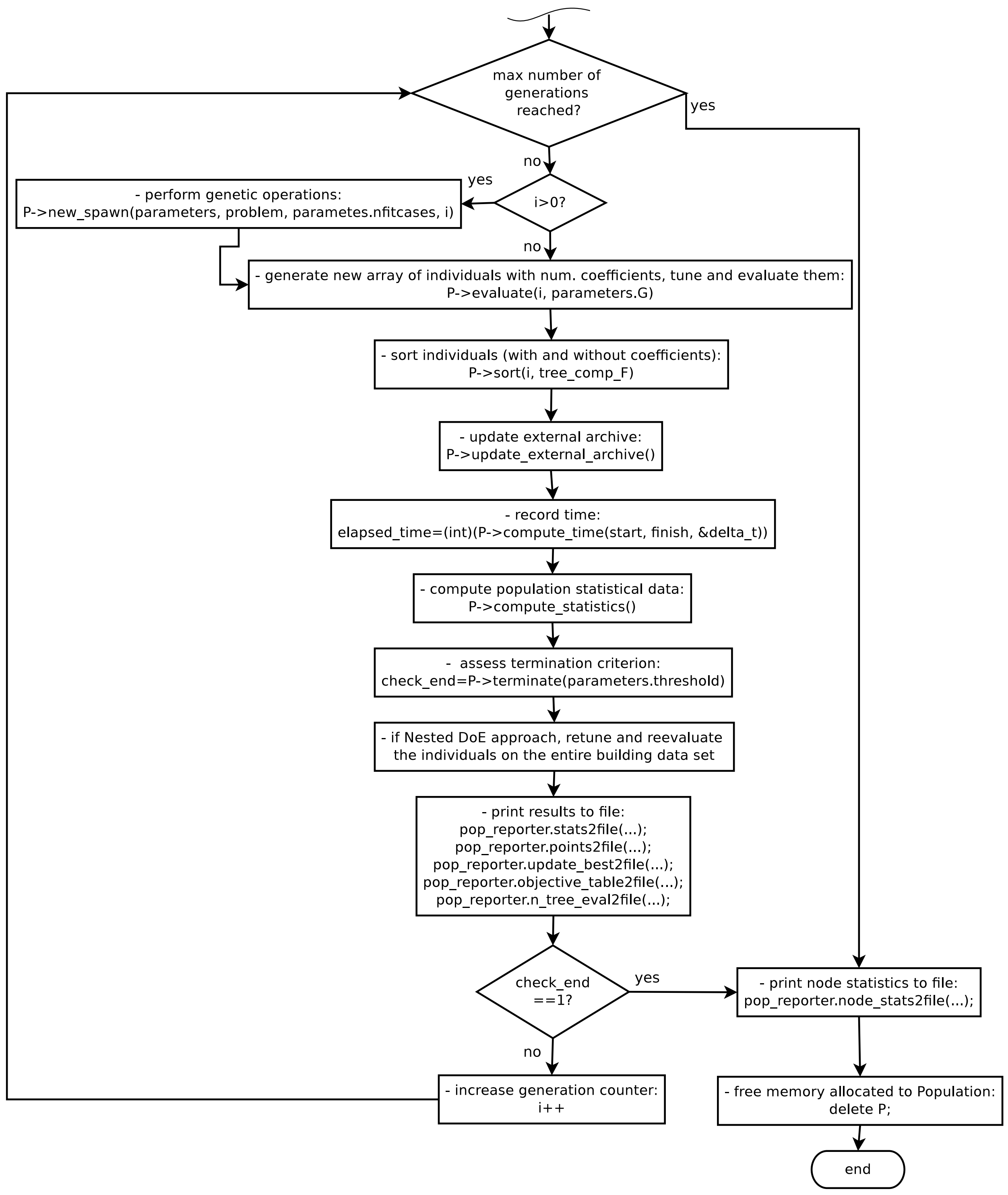

max number of generations reached?
yes
no
i>0?
yes
- perform genetic operations:
P->new_spawn(parameters, problem, parametes.nfitcases, i)
no
- generate new array of individuals with num. coefficients, tune and evaluate them:
P->evaluate(i, parameters.G)
- sort individuals (with and without coefficients):
P->sort(i, tree_comp_F)
- update external archive:
P->update_external_archive()
- record time:
elapsed_time=(int)(P->compute_time(start, finish, &delta_t))
- compute population statistical data:
P->compute_statistics()
- assess termination criterion:
check_end=P->terminate(parameters.threshold)
- if Nested DoE approach, retune and reevaluate the individuals on the entire building data set
- print results to file:
pop_reporter.stats2file(...);
pop_reporter.points2file(...);
pop_reporter.update_best2file(...);
pop_reporter.objective_table2file(...);
pop_reporter.n_tree_eval2file(...);
check_end ==1?
yes
- print node statistics to file:
pop_reporter.node_stats2file(...);
no
- increase generation counter:
i++
- free memory allocated to Population:
delete P;
end

FIGURE A.4: HyGP flowchart - 2nd part

A.3 HyGP parallelisation

HyGP first implementation was developed without any awareness of the potentiality of multiprocessor computers. After the first sequential executions on a desktop machine, it soon appeared clear that, given the computational cost of a single HyGP run, parallelisation would be a necessary prerequisite for HyGP application to academic and industrial modelling problems.

The idea to exploit parallelism is nowadays an established strategy to make efficient use of computational resources (Friedman 1991, Rogers and LaMarsh 1995, Simpson et al. 2001, Polynkin et al. 2008). Quoting Chapman et al. (Chapman et al. 2007, p. 3):

> "It is vital that application software be able to make effective use of the parallelism that is present in our hardware"

GP and more in general GA algorithms have features that make their parallelisation particularly easy and efficient. Multiple similar and independent operations are performed at different levels: fitness evaluation is an example of operation that can be executed in parallel at individual level, whereas the evolution of a single metamodel (GP) or design point (GA) is an example at population level. Such a hierarchical organisation brought Fogel to refer to artificial evolution as "an inherently parallel process" (Fogel 1994, p. 11). As reported in Zhang and Mühlenbein (1995), "the evolutionary approach differs from most other search techniques in that it makes a parallel search simultaneously involving hundreds or thousands of points in the search space". Schmidt and Lipson (2009a) also highlighted that the nature of genetic algorithms is intrinsically parallel.

Parallelisation is then a necessary step to fully exploit evolutionary algorithms' potentiality (Fogel 1994, Affenzeller and Wagner 2004, Kroo 2004, Winkler et al. 2007) and to make the otherwise excessive computational cost of evolutionary techniques manageable (Barbosa and Bernardino 2011).

Before showing how HyGP was parallelised, it is important to remind the difference between distributed GP and parallel GP. In distributed GP many populations (also called *islands*, *demes* or *villages* - Fogel (1994), Affenzeller and Wagner (2004)) are evolved independently although some migrations between them can be established to boost genetic variability, as described in Section 4.6, Chapter 4. The adjective "parallel" instead refers to the fact that a few operations of the GP algorithm are performed concurrently on different machines or processors. The two concepts are independent, although distributed GP massively benefits from the parallelised evolution of different populations.

A.3.1 Target architecture

As seen in the previous section, prerequisite for algorithm parallelisation is hardware parallelism, or the existence of multiprocessor architectures. The design of an efficient multiprocessor computer is a research field in itself and far beyond the scope of this work. It is however important to describe the main typologies of parallel architectures to understand how the parallelism in the hardware can be exploited so to match the intrinsic parallel nature of genetic algorithms.

Two basic ways of connecting processors in parallel exist, which differ according to the modality of access to computer memory. Shared memory systems, also called *symmetric multiprocessing systems* (SMPs), are architectures where many central processing units (CPUs) are connected to a single, shared memory unit. Being all the CPUs close to the memory, data are accessed quickly and networks are not needed. On the other hand, memory sizing is critical to guarantee that all processes are allotted enough memory for completing their task. A schematic diagram of a SMP system is shown in Fig. A.5.

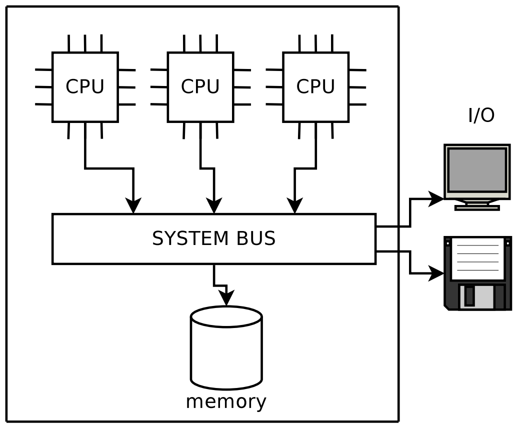


FIGURE A.5: Shared memory architecture or SMP

Distributed memory systems feature instead a memory unit for each processor. Multiprocessor architectures are assembled connecting the single systems made of processor and memory. These external (network) connections are however generally slower than the internal connections between processor and memory, so distributed memory systems are generally used when no massive data exchange among processors is required (all data needed by each processor is loaded into the memory and output exchanged at task completion). Fig. A.6 reports a diagram of a distributed memory system.

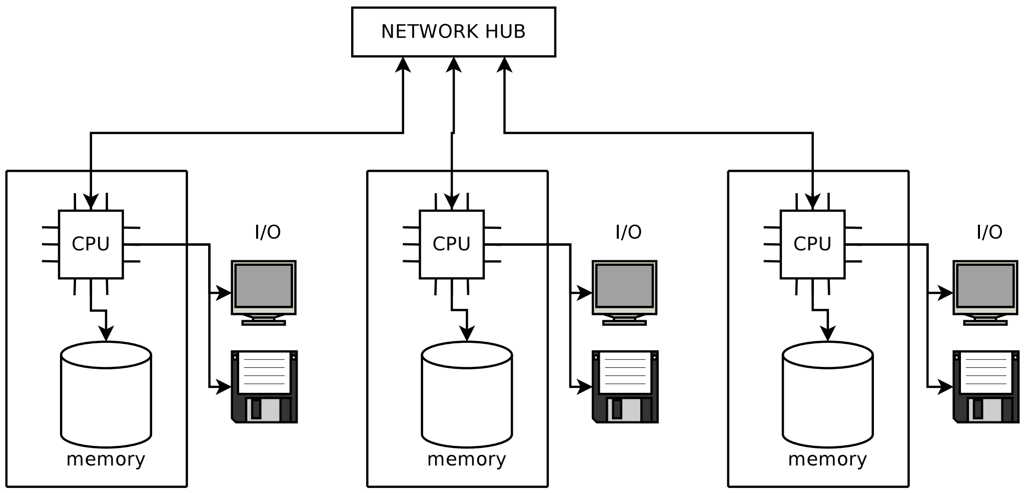


FIGURE A.6: Distributed memory architecture

All the multiprocessor architectures available nowadays are combinations of the previous basic systems. A typical parallel architecture that can be found in university and industry laboratories is the so-called "cluster", which can be defined as a distributed memory system composed of many shared memory machines or nodes, which are usually standard workstations connected by an off-the-shelf network. Code designed to be run on clusters may then exploit two levels of hardware parallelisation: among the nodes, typical of distributed systems, and among the cores in a single node. The former is referred to as "coarse grain parallelism", whereas the latter as "fine grain parallelism" (Garcke et al. 2003, Chapman et al. 2007). A cluster architecture is represented schematically in Fig. A.7.

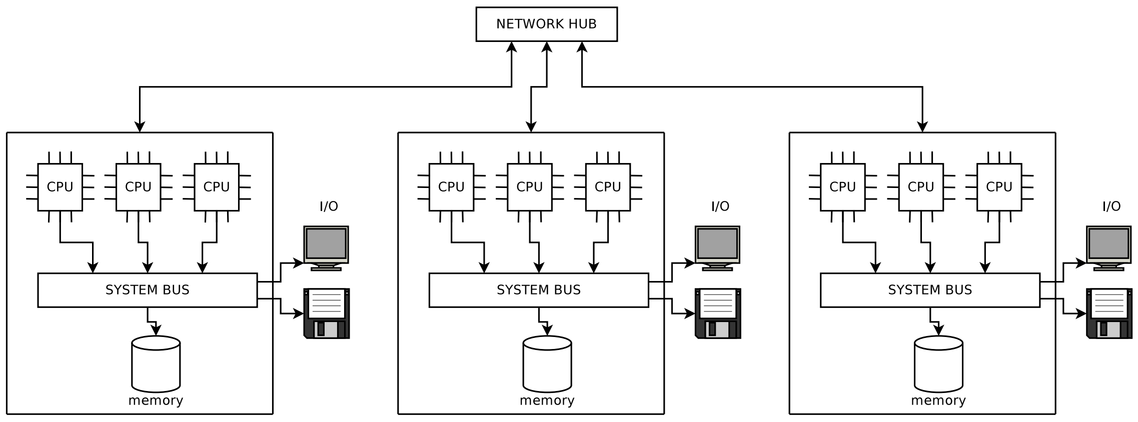
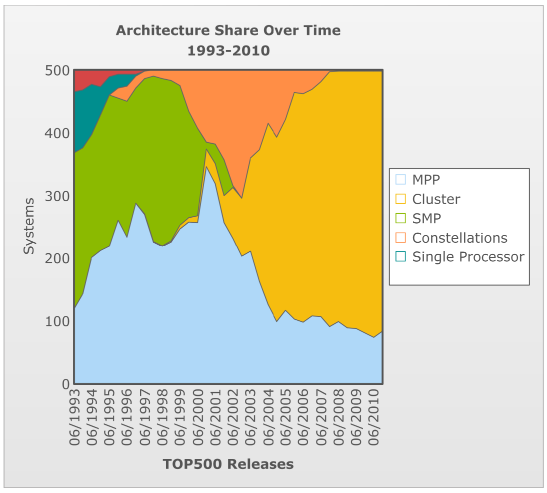


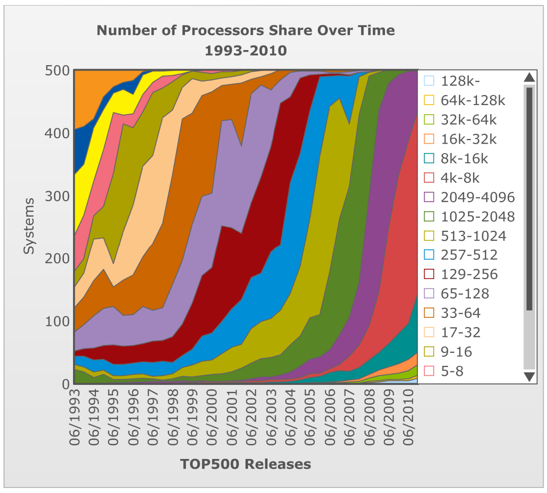
FIGURE A.7: Cluster architecture

Other parallel architectures are available, but they are still based on a combination of shared/distributed memory systems. Massively Parallel Processors (MPPs) for example are distributed memory architectures. Constellations are clusters of "large SMP nodes scaled such that the number of processors per node is greater than the number of nodes"[9].

A brief analysis of the systems used for the development of supercomputers may help identify the most efficient and cost effective parallel architecture that could be used in the near future in industry and academia: Fig. A.8A and Fig. A.8B report the architectures and the number of processors of the 500 most powerful supercomputers in the period 1993-2010 (Meuer et al. 1993). From the figures, it emerges that the cluster is by far the most common ways to build high performance computing machines nowadays. Appeared on the TOP500 list in 1998 (Meuer et al. (1993),Vaughan-Nichols (2004)), cluster architecture reached in November 2010 82.80% of the share of the most powerful supercomputers. Fig. A.8B reveals the massive increase in the number of processors used in these supercomputers, sign that the current trend to increase computing power relies on distributing computational jobs on a higher and higher number of processors. Parallel computation can exploit a range of 4000-8000 processors in the most typical cluster configuration.



(A) Multiprocessor architecture share

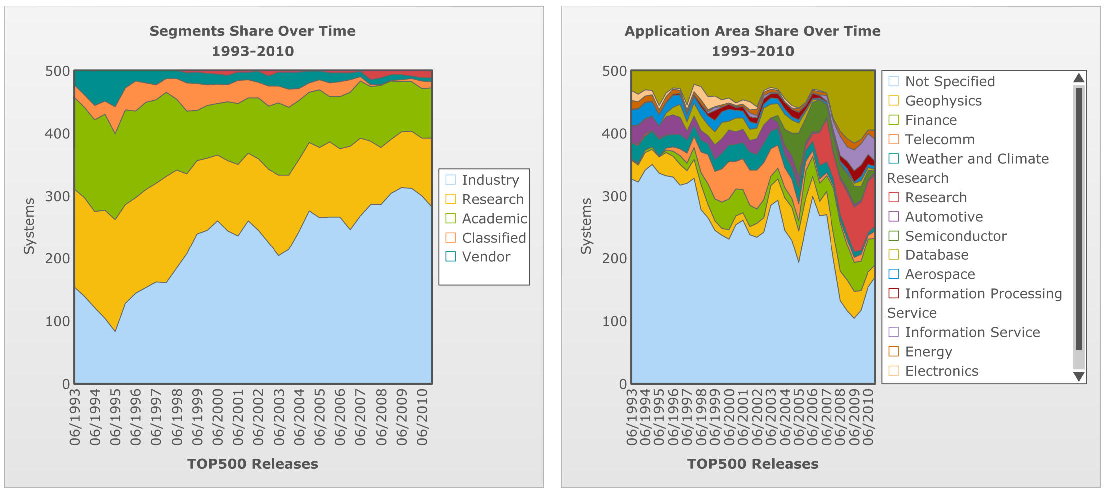


(B) Number of processors share

FIGURE A.8: Architecture and number of processors of the 500 most powerful supercomputers (period 1993-2010) (image reported under permission - http://www.top500.org).

Fig. A.9 shows the sectors and the application areas which supercomputers were typically used in during the period 1993-2010.

[9]excerpted from `http://www.clusterconnection.com/2009/06/cluster-or-constellation/` on April 29, 2012.



(A) segments share (B) application area share

FIGURE A.9: Segment and application area share of the 500 most powerful supercomputers (period 1993-2010) (image reported under permission - http://www.top500.org).

Cluster architectures then are and will be an efficient, reliable and cost effective way to build parallel machines. Therefore they were selected as the ideal target systems for the parallelised HyGP implementation. As a matter of fact, cluster architecture matches the requisites of HyGP experiments, as "clusters are effective for loosely coupled tasks" (Vaughan-Nichols 2004). HyGP was first parallelised on a SMP system (single node of a cluster), using a "fine grain" approach. A parallel HyGP version was also implemented on a distributed memory architecture (using a single processor per cluster node) to overcome the memory limitations imposed by the SMP configuration. Although a HyGP implementation exploiting both fine and coarse grain parallelism featured by the cluster architecture was not actually developed, in Section A.4 suggestions for the full exploitation of cluster parallelism are given.

A.3.2 Parallelisation for shared memory architectures

As described in Section 5.3.1, Chapter 5 HyGP experiments are made of a certain number of runs, which are independent evolution processes starting from a randomly initialised population of metamodels. As single runs do not exchange data during evolution, they can be easily executed in parallel, allocating a run to a single process. This parallelisation strategy was initially implemented for execution on a single multiprocessor cluster node, which can be considered a SMP machine.

The two most known languages for code parallelisation on SMPs are Pthreads and OpenMP (Hoeflinger 2006, Chapman et al. 2007, Grama et al. 2008). OpenMP can be defined as a set of directives or commands that are able to split or share operations among processors in shared-memory parallel computers (SMPs) (Hoeflinger 2006, Chapman et al. 2007, Barney 2011, Gustafson 2011) . It was introduced by a group of industries called the *OpenMP Architecture Review Board* (ARB) in the latter half of the 1990s[10] with the aim of making parallel programming easier and reduce portability problems. The set of OpenMP directives are today widely accepted, making parallelisation far less dependent on the architectures. One advantage of the OpenMP directive approach is that the parallelisation can be done step by step, retaining the original sequential code (Hoeflinger 2006, Chapman et al. 2007). Although Pthreads allows a greater degree of control and precision on process handling, it is also more complex and harder to learn than OpenMP. The basic control on memory and threads allowed by OpenMP was considered sufficient for the simple parallelisation task, so HyGP was parallelised using OpenMP.

Central concept in the operation allocation among processors is the “thread”, defined as a set of instructions that can be executed independently from others. HyGP evolutions are by independent, so each one of them was allocated to a “thread”. The instructions to compile and execute HyGP on a SMP system are given in Section B.2.2, Appendix B.

HyGP parallel execution on SMP systems is in general suggested for problems of reduced dimensionality and small building data sets. In this case, OpenMP allows HyGP to be run also on most multiprocessor desktop computers, both in parallel or sequential mode with no change to HyGP source code required (the original sequential HyGP source code is retained). For more computationally intensive problem, the amount of RAM memory available on the machine may not be enough for parallel execution of 10 or more runs, considering that the typical RAM usage for a single HyGP run is 4 to 6 GB, depending on the population size and depth limit. OpenMP cannot be run on a full cluster, as it is not an SMP machine, but OpenMP code development (Cluster OpenMP) may give this opportunity in the future (Hoeflinger 2006).

A.3.3 Parallelisation for distributed memory architectures

A distributed memory system was considered as the target architecture for a second parallel HyGP version. The aim was to tackle the memory constraints that make parallel symbolic regression of highly dimensional data not efficient on a typical SMP machine,

[10]The first version, written to be used with Fortran, was released in 1997.

unless the number of concurrent HyGP runs is drastically reduced, so losing the advantage of parallel computing. The same parallelisation strategy introduced in the previous section was followed, allocating one evolution per process.

As for SMP machines, many languages can be used to write a code exploiting the hardware parallelism of a distributed memory machine. MPI, or Message Passing Interface, is one of the most known set of directives for code parallelisation. However, it requires major changes in the sequential source code (Chapman et al. 2007). It has to be noted that, in general, higher costs for program development using MPI with respect to shared-memory parallel programming (OpenMP) have to be expected (Hoeflinger 2006).

HyGP implementation, on the other hand, does not require strict parallelisation: data transfer among different runs is not needed, as a distributed approach is not implemented. So parallelisation through MPI was not even needed. A compromise in terms of simplicity and fair usage of resources was found in the use of *job array* (Thornton 2010). A job array can be defined as a method to submit to a cluster a set of computational jobs with different initial settings. The fact that jobs are submitted as soon as the computational resources in the cluster become available solves the problem of lack of memory: in case of memory shortage processes (evolutions) are launched in sequential mode, otherwise evolutions are launched concurrently. Also, no HyGP source code modification was needed, as a job array is declared through a script that calls the code written for sequential execution.

In Section B.2.3, Appendix B the instructions on how to launch a job array are given.

A.4 Conclusion

In this appendix the structure of the genetic programming implementation used to generate the metamodels presented in the rest of the thesis, known as HyGP, has been described. Open source tools (operating system, compilers, parallelisation directives) were used in order to allow for the widest diffusion of the code[11]. Some computational issues, like efficient parallelisation, have been addressed during the development. As a result, HyGP allows for a set of independent runs to be launched in parallel on a single multiprocessor computer or on a cluster using job arrays.

However, only a single level of parallelism has been exploited so far. Future research will have to focus on more complex ways to fully exploit the double parallelism of clusters:

[11]HyGP is freely downlodable from `http://www.personal.leeds.ac.uk/~cnua/mypage.html`. More information on input file format and output generated is provided in Section B.1, Appendix B.

whereas a single evolution can be allocated per cluster node using MPI or job arrays, fitness evaluation can also be parallelised within each node. Being each cluster node an SMP system, OpenMP could be used to efficiently parallelise even fitness cases evaluations on the different processors composing the node. This approach could also pave the way to an efficient implementation of a distributed version of HyGP (in this case inter-node parallelisation through MPI directives is suggested, as evolutions should be performed concurrently).

The use of Graphics Processing Units (GPUs) (Harding and Banzhaf 2007) is also a new field of research from which dramatic reduction of fitness evaluation time can be expected.

Appendix B

Guide to HyGP use

This appendix shows how to use the genetic programming implementation developed during the research activity, called HyGP. The operations required to set up HyGP input file are described in Section B.1, while in Section B.2 it is shown how to compile and launch a HyGP experiment once the input file has been created. HyGP code was written in C++ and is supposed to be run in a Linux/Unix environment, although executables for Windows can be in theory obtained using different compilers from the ones assumed in the following.

The output produced by HyGP is described in Section B.3. A few Matlab scripts have been written to post-process the text expressions generated by HyGP (validation and results plotting), so to ease the interpretation of the results. All the images reported in this dissertaton were generated using these scripts, which are briefly described in Section B.5.

As the steps described in the following were relentlessly performed during the research activity to produce the metamodels shown in Chapter 5 and Chapter 6, a few suggestions to help the user get the most out of HyGP are given in Section B.4 .

B.1 Input file and evolution parameters

Listing B.1 shows the general format of a HyGP input file. The symbol “#” (number sign) is used to introduce comments in the input file (lines beginning with “#” are neglected by HyGP). Words in capital are labels used to assign HyGP parameters: the value set but the user has to follow the symbol “=” (equal sign).

The data in the input file is organised in three main sections:

1. evolution parameters

2. functional primitives available to HyGP

3. data matrix and matrices used to impose inequality constraints on metamodel output and derivatives

B.1.1 Evolution parameters

The main parameters for the evolution are declared in the first section of the input file (lines 2-34 in Listing B.1). Special care has to be taken in setting the values: a blank space has to be left between "=" and the given value, but no space has to be put between the name of the label and "=". For example NVAR=␣2 is correct, whereas NVAR␣=2 and NVAR␣=␣2 are not accepted.

The list of basic HyGP evolution parameters is reported in Tables B.1 (run set up and population initialisation), B.2 (genetic operators, fitness function and termination criterion) and B.3 (split DoE settings).

In Table B.4 the main parameters to set up inequality constraints on metamodel output and derivatives are shown. More details on this feature will be given in Section B.1.4.

LISTING B.1: Example of input file

```
# Evolution parameters
SEED= 20
NVAR= 2
MINRAND= −200
MAXRAND= 200
MAX_N_PERIODS= 1.5
STEP= .001
NFITCASES= 10
METHOD= 4
DEPTH_MAX= 4
DEPTH_MIN= 2
DEPTH_LIM= 50
p_FULL= .5
REPR_RATE= .2
CROSS_RATE= .4
MUT_RATE= .4
COMP_RATE= .0
NEW_RATE= .0
M= 200
G= 50
NORMALISED_ERR= 0
MINMAX= 0
W_COMPLEXITY= .01
W_N_CORRECTIONS= .1
W_SIZE= .001
W_FACTORISATION= .005
N_GUESSES= 2
SPLIT= 0
VALIDATING_LINES= 6
THRESHOLD= 1.0E−12
N_INEQUALITY0= 8
W_PEN_ORD0= 0.001
N_INEQUALITY1= 5
W_PEN_ORD1= 0.001
# List of operations
BINARY_FUN= add,sub,mult,sdiv
UNARY_FUN= pow1,square,cube,sin,cos,exp,nxp,inv
#
# variable 1 (Z1)   variable 2 (Z2)   corresp. output
    9.306026e−01     −5.733779e−01     2.071918se+02
         ...               ...               ...
   −1.786222e+00      1.259808e+00      3.805549e+02
# 0−order inequality constraints
0.0       0.0       .5        >
−1.0      0.0       90.0      >
 ...       ...       ...       ...
2.0       −2.0      3500      >
# first−order inequality constraints
0.0       0.0       1.0       0.0       0.0       <
...       ...       ...       ...       ...       ...
1.5       −1.5      1.0       0.0       2000      >
```

TABLE B.1: HyGP evolution parameters. Run set up and population initialisation

Parameter	Value	Notes
SEED	[int]	seed for random number generator. If set to -1 the seed is initiliased by using the computer clock. If a positive value is given, such value is used as seed
NVAR	[int]	number of independent variables for the problem
MINRAND	[float]	lower bound for the random number generator
MAXRAND	[float]	upper bound for the random number generator
STEP	[float]	increment for the random number generator (random values picked in [MINRAND:STEP:MAXRAND])
NFITCASES	[int]	size of the building data set (MUST be equal to the number of rows of the data matrix)
METHOD	[int]	type of method used for tree generation/initialisation (1 - unlimited, 2 - full, 3 - grow, 4 - ramped half and half)
DEPTH_MAX	[int]	maximum depth of a randomly generated parameter-less tree (initialisation)
DEPTH_MIN	[int]	minimum depth of a randomly generated parameter-less tree (initialisation)
p_FULL	[float]	percentage (normalised, from 0 to 1) of individuals to be generated using "full" method in case method "ramped half and half" is chosen. The rest is filled using "grow" method.
M	[int]	Population size (constant during the evolution)
G	[int]	Maximum number of generations

TABLE B.2: HyGP evolution parameters. Genetic operators, fitness function and termination criterion

Parameter	Value	Notes
REPR_RATE CROSS_RATE MUT_RATE	[float]	percentages (from 0.0 to 1.0) of new populations generated by reproduction, crossover and mutation. The sum MUST be equal to 1
DEPTH_LIM	[int]	maximum accepted depth of parameterless offspring produced by crossover and subtree mutation
COMP_RATE	[float]	percentage of new population generated "composing" existing individuals (see Section 5.3.0.2, Chapter 5)
NEW_RATE	[float]	percentage of new population generated from scratch (see Section 5.3.0.2, Chapter 5)
MAX_N_PERIODS	[float]	maximum number of periods of functions $sin(a * z1)$ or $cos(a * z1)$ in the given variable range (see Section 5.3.0.6, Chapter 5)
NORMALISED_ERR	[int]	error formulation. If 0 RMSE is used, if 1 the normalised version is used (see Section 5.3.0.7, Chapter 5)
MINMAX	[int]	MinMax formulation. If 0 a linear combination of objectives is used, if 1 the MinMax formulation is used (see Section 5.3.0.7, Chapter 5)
W_COMPLEXITY	[float]	weight for complexity objective
W_N_CORRECTIONS	[float]	weight for singularity objective
W_SIZE	[float]	weight for size objective
W_FACTORISATION	[float]	pressure for fractional expression generation
N_GUESSES	[int]	number of initial random guesses for SQP optimiser
THRESHOLD	[float]	desired accuracy on the building data set. The evolution is stopped when the best individual in a generation has a RMSE lower or equal to THRESHOLD

TABLE B.3: HyGP evolution parameters. Split DoE settings

Parameter	Value	Notes
SPLIT	[int]	switch for nested DoE technique (see Section 5.3.0.5, Chapter 5). Used if set to 1, not used if set to 0
VALIDATING_LINES	[int]	number of lines in the data matrix that are used as validation data set, starting from the first row of the matrix. All the other rows in the matrix are used for tuning

TABLE B.4: HyGP evolution parameters. Inequality constraints settings

Parameter	Value	Notes
N_INEQUALITY0	[int]	number of inequality constraints on metamodel output (number of rows of first additional matrix)
W_PEN_ORD0	[float]	weight of the objective defining metamodel output inequality constraints in fitness function
N_INEQUALITY1	[int]	number of inequality constraints on metamodel partial derivatives (number of row of second additional matrix)
W_PEN_ORD1	[float]	weight of the objective defining metamodel derivatives inequality constraints in fitness function

B.1.2 Available operations

The functional primitives that can be used in a HyGP run have to be declared in two separate classes: binary functions and unary functions. If more than a primitive is declared, a comma has to be used to separate the names. For example:

```
BINARY_FUN= add,sub

UNARY_FUN= square,cube,sin,cos,exp
```

The currently available operations are listed in Tables B.5 (binary functions) and B.6 (unary functions).

TABLE B.5: List of available HyGP binary operations

Symbol	operation	Notes
`add`	Addition	
`sub`	Subtraction	
`mult`	Multiplication	
`sdiv`	Protected division	It returns 1 if the denominator is zero (and increases the penalisation for unfeasible responses)
`spow`	Protected power	It returns 1 in case of undefined operation (and increases the penalisation for unfeasible responses)

TABLE B.6: List of available HyGP unary operations

Symbol	operation	Notes
`square`	Power of 2	
`cube`	Power of 3	
`sin`	Sine	
`cos`	Cosine	
`shift`	Shift operation	It adds a constant to the argument (translation operator)
`exp`	Exponential	
`nxp`	Negative exponential	1/exp(x), where x is the actual argument of the function
`neg`	Opposite	Attention! It spoils the effect of the algorithm used to penalise high frequency noise (Section 5.3.0.6, Chapter 5), as if the argument of $cos()$ or $sin()$ are multiplied by -1 the standard format $sin(\omega * x)$ or $cos(\omega * x)$ is not matched. It is better not to use it, SQP tuner can find negative arguments anyway.
`log`	Natural logarithm	Protected version: it returns a predefined large number (MAX_VAL) if the argument is not positive (and increases the penalisation for unfeasible responses).
`sinh`	Hyperbolic sine	
`cosh`	Hyperbolic cosine	
`tanh`	Hyperbolic tangent	
`abs`	Absolute value	
`inv`	Protected reciprocal	1/x, where x is the actual argument of the function. It returns 1 if x is zero (and increases the penalisation for unfeasible responses).

B.1.3 Data matrix

The set of m=NFITCASES fitness cases used as building data set has to be declared as a matrix, in which each row corresponds to a sample point (see definition of data matrix 1.3 in Section 1.2, Chapter 1):

$$\begin{array}{ccccc} x_{1,1} & x_{1,2} & \dots & x_{1,N} & y_1 \\ & & \dots & & \\ x_{i,1} & x_{i,2} & \dots & x_{i,N} & y_i \\ & & \dots & & \\ x_{m,1} & x_{m,2} & \dots & x_{m,N} & y_m \end{array} \tag{B.1}$$

The $m \times N$ data matrix is declared in the input file after the functional primitives (lines 40-42 in Listing B.1).

In case the nested DoE option is used (switch SPLIT set to 1 - see Table B.3), the first VALIDATING_LINES of the data matrix are used for fitness evaluation, whereas the rest are used for model building (parameters tuning using SQP algorithm).

B.1.4 Matrix defining the inequality constraints on metamodel output

The matrix is used to define the desired behaviour of the evolved metamodels' output on a set of given points in the form of an inequality constraint, according to the approach described in Section 5.4, Chapter 5. Each row of the matrix contains the coordinate of a point, a value and a sign that defines the preceding value as a lower or upper bound of the metamodel output in the point:

$$x_{i,1} \quad x_{i,2} \quad \ldots \quad x_{i,N} \quad y_b \quad \textit{inequality sign}$$

For example, if in a symbolic regression problem in two variables the metamodel output value has to be larger than 0.5 in the point $(0,0)$, but negative in $(2,2)$, the following lines should be used:

$$\begin{array}{cccc} 0.0 & 0.0 & 0.5 & > \\ 2.0 & 2.0 & 0.0 & < \end{array}$$

The matrix defining the inequality constraints on metamodel output must be defined after the data matrix (see lines 44-47 in Listing B.1). The number of rows must be equal to N_INEQUALITY0 (see Table B.4) and the number of columns must be equal to NVAR+2 (see Table B.1).

B.1.5 Matrix defining the inequality constraints on metamodel derivatives

This matrix is used to define the desired behaviour of the evolved metamodels' partial derivatives on a set of points, similarly to what has been done in the previous section. This feature was developed to penalise not acceptable rate of increase/decrease. Experiments performed so far showed ambiguous results, so further research is needed to get a conclusive assessment of this capability.

To impose that the partial derivative of the evolved metamodels evaluated in the direction defined by the vector $\{v_{i,1},\ v_{i,2},\ ,\ldots,\ v_{i,N}\}$ in the point $\{x_{i,1},\ x_{i,2},\ldots,\ x_{i,N}\}$ is smaller or larger than a given y'_b value the following line has to be used in the input file:

$$x_{i,1} \quad x_{i,2} \quad \dots \quad x_{i,N} \quad v_{i,1} \quad v_{i,2} \quad \dots \quad v_{i,N} \quad y'_b \quad \textit{inequality sign}$$

For example the line:

$$1.5 \quad -1.5 \quad 1.0 \quad 0.0 \quad 2.0 \quad > \tag{B.2}$$

means that the partial derivative along the direction $(1.0, 0.0)$ (first axis) evaluated in point $(1.5, -1.5)$ has to be larger than 2.0.

The matrix defining the inequality constraints on metamodel derivatives must be inserted after the data matrix (see lines 49-51 in Listing B.1). The number of rows must be equal to N_INEQUALITY1 (see Table B.4) and the number of columns must be equal to 2*NVAR+2 (see Table B.1).

Partial derivatives are computed in HyGP using a first order finite difference method, so they should be considered approximations of the real derivative value.

B.2 Compiling and linking

HyGP was mostly written in C++ but contains also parts in Fortran77. C++ was used to implement all the operations related to population handling (genetic operators, sorting, initialisation, fitness evaluation, etc). To allow for fast optimisation of the numerical parameters of the generated GP individuals the SQP algorithm developed in Fortran 77 by Madsen et al. (2002) was linked to the main C++ HyGP code.

HyGP is assumed to be run on a Linux operating system. A makefile based on g++ and gfortran compilers was written to ease compilation and linking of the source files. The makefile is listed in Listing B.2.

The code was written in order to run in sequential mode or in parallel mode (see Section A.3, Chapter A). As a result, according to the architecture available, the code can be run:

- sequentially on a single CORE, one evolution after the other (for standard desktops or laptops)
- in parallel on one machine (node), allocating one evolution to each CORE (for multiprocessor/multithread machines)

LISTING B.2: HyGP makefile

```
# Basic Makefile
all: gp

gp: master.o T.o M.o
# for Linux on feng-gps1
        gfortran -std='legacy' -o gp M.o T.o master.o
         -L/usr/lib/gcc/x86_64-redhat-linux/4.0.2/
         -lstdc++
         -fopenmp

M.o: ./genetic_code/SQP/MI0L2_c/MI0L2.FOR
        gfortran -std='legacy'
                -c ./genetic_code/SQP/MI0L2_c/MI0L2.FOR
                -o M.o

T.o: ./genetic_code/SQP/MI0L2_c/TI0L2.FOR
        gfortran -fsecond-underscore
         -c ./genetic_code/SQP/MI0L2_c/TI0L2.FOR  -o T.o

master.o: master.cpp
         # parallel compilation
         g++ -c -g parallel_master.cpp -o master.o -fopenmp
         # normal compilation
         # g++ -c -g master.cpp -o master.o

clean:
        rm M.o
        rm T.o
        rm master.o
        rm gp
```

- in pseudo-parallel on many nodes, one evolution for each NODE (for clusters with Sun Grid Engine - array jobs are used)

In the following the instructions to generate the executables for each execution mode will be described. A few conventions regarding the names of the directory will be assumed:

- `PROJECT_PATH` path where HyGP source code was saved to;
- `PATH_INPUT_FILE` is the input file location;
- `PATH_OUTPUT_DIRECTORY` is the path of the directory where the output files are written to. The directory is automatically generated by HyGP;
- `NO_RUNS` is the number of independent evolutions (runs) the HyGP experiment is made of.

Before compiling, make sure that the path of the library -lstdc++ (line 8 in Listing B.2) is correct for the system in use.

B.2.1 Compiling for sequential execution on a single processor computer

This is the simplest way to compile and run the GP code. Evolutions are run one after the other on a single processor. To generate the executable, first open the makefile and in the group of instructions following `master.o:` leave uncommented only the line below "normal compilation", as shown below:

```
master.o: master.cpp
# parallel compilation
# g++ -c -g parallel_master.cpp -o master.o -fopenmp
# normal compilation
 g++ -c -g master.cpp -o master.o
```

In a linux shell, enter the directory where HyGP source code was saved:

```
>> cd PROJECT_PATH
```

and type "make":

```
>> make
```

The makefile performs compilation and linking and generates the executable called "gp". If that does not happen, most of the times this is due to a missing library. Once "gp" has been generated, a HyGP experiment is launched by calling the script `experiment_new`:

```
>> ./experiment_new PATH_INPUT_FILE PATH_OUTPUT_DIRECTORY NO_RUNS
```

If the input file and/or the output directory is in the same directory of the script, then the two characters "./" have to be always used as in:

```
>> ./experiment_new ./input_file.txt ./output_dir 10
```

The script `experiment_new` collects the best individual generated in the experiment as well as some additional statistical data, so there is no need to run other scripts at the end of the experiment.

B.2.2 Compiling for parallel execution on a multiprocessor/multithread machine

This execution mode allocates a single evolution to every processor. As most of modern computers have a multiprocessor or multithread architecture, it is then recommended to substantially reduce the time required for an experiment, although for large building data sets and for high dimensional problems may require a large amount of RAM. OpenMP capability is required.

To generate the executable, first open the makefile and in the group of instructions following `master.o:` leave uncommented the line below "parallel compilation", as shown below:

```
master.o: master.cpp
# parallel compilation
g++ -c -g parallel_master.cpp -o master.o -fopenmp
# normal compilation
# g++ -c -g master.cpp -o master.o
```

In a linux shell, enter the directory where the source code was saved:

```
>> cd PROJECT_PATH
```

and type "make":

```
>> make
```

The makefile performs compilation and linking and generates the executable "gp". The experiment is then launched calling the executable "gp" and NOT the script `experiment_new`:

```
>> ./gp PATH_INPUT_FILE PATH_OUTPUT_DIRECTORY NO_RUNS p
```

Note that the "p" is an argument that tells the code to run in parallel mode. Replacing the "p" with an "s" the code would run sequentially (no difference with the execution mode described in the previous section). When the experiment is finished, the script `posteriori` (located in `PROJECT_PATH`) has to be launched to extract the best model of the HyGP experiment as well as statistical data about the independent evolutions:

```
>> ./posteriori PATH_OUTPUT_DIRECTORY NO_RUNS
```

B.2.3 Compiling for pseudo parallel execution on a cluster (SGE array job)

This implementation was written specifically to be used on a Sun Grid Engine cluster, as array jobs are used to manage the allocation of the single evolutions to the different nodes of the cluster, if necessary. If enough resources are available, evolutions are run in parallel; if not, they are run sequentially. For this reason this implementation is particularly suggested for expensive evolutions (high number of points in DoE, big populations (>400), long evolutions (more than 200 generations). To generate the executable, first open the makefile and in the group of instructions following `master.o:` leave uncommented the line below "normal compilation", as shown below:

```
master.o: master.cpp
# parallel compilation
# g++ -c -g parallel_master.cpp -o master.o -fopenmp
# normal compilation
 g++ -c -g master.cpp -o master.o
```

In a linux shell, enter the directory where the source code was saved:

```
>> cd PROJECT_PATH
```

and type "make":

```
>> make
```

The makefile performs compilation and linking and generates the executable "gp". The experiment is launched through submission of the array job script `experiment_sge` to an SGE cluster. Before launching the script, the HyGP input file location and the output directory path have to be declared in the script. So open `experiment_sge`, located in `PROJECT_PATH`, and modify the lines as `INPUT_STRING=` and `OUTPUT_STRING=` as follows:

```
INPUT_STRING= PATH_INPUT_FILE
OUTPUT_STRING= PATH_OUTPUT_DIRECTORY
```

Once updated, submit the script to the Sun Grid Engine using the `qsub` command:

```
>> qsub ./experiment_sge
```

When the HyGP experiments terminates, general statistical data regarding the evolutions as well as the best model of the experiment can be extracted by typing:

```
>> ./posteriori PATH_OUTPUT_DIRECTORY NO_RUNS
```

as done with the parallel implementation described in the previous section.

B.3 HyGP output

For each independent run HyGP produces exactly the same output files. These files are saved in directory called “run_M”, where M is the number of the run. All “run_M” directories are generated in the path `PATH_OUTPUT_DIRECTORY` declared by the user when he launches the HyGP experiment (mind that the directory is automatically generated by HyGP).

The following files are created in each directory “run_M”:

- `best_gp.txt`: it contains the expression of the best model at each generation, together with its main properties (no of generation, fitness value, RMSE error, R^2, number of hits). The file is updated at each generation.

- `data_gp.txt`: it contains statistical data about the whole population: minimum, average and maximum model error (Fit), fitness (F), individual size (S) and depth (D) at each generation (generation number specified in first column). The file is updated at each generation.

- `latest_archive.txt`: the mathematical expression and the properties of the individuals belonging to the archive/elite (or best set of models found at each generation) are stored here. The file is overwritten at each generation and the data format is the same as in `best_gp.txt`.

- `node_selection.txt`: it contains statistical data regarding node selection. In first column the total no of nodes selected during the evolution is reported, in second to fifth column the number of nodes selected per type (Binary , Unary, Var, and Const nodes) d In the remaining columns the number of nodes selected per depth is written. The file is written only at the end of the HyGP run.

- `objectives.txt`: it can be neglected.

- `points_gp.txt`: it stores the response of the best model evolved (column “tree”) and of the known output (column “g_obj”) for each point in the building data set. The elapsed time since the start of the evolution, the seed used and other parameters are recorded as well. The file is overwritten at each generation.

- `n_tree_evaluations.txt`: the number of tree evaluations per generation are stored here (first column generation number, second column number of tree evaluations so far). The information reported can be useful in determining the cost of the evolution (see Section 3.1.6, Chapter 3).

Besides the data related to each HyGP run, statistical data is gathered at the end of a HyGP experiment by the script `posteriori`. The data extracted by the script is organised in the following files:

- `archives.txt`: it contains the archives/elites of each evolution
- `archives_best.txt`: it contains the expressions of the best model generated by each evolution
- `best_tree.txt`: it contains the expression of the best model generated among all evolutions
- `depth_ave.txt`
- `fitness_min.txt`
- `fitness_var.txt`
- `F_min.txt`
- `F_var.txt`
- `nodes_stat.txt`
- `size_ave.txt`
- `time_stat.txt`

B.4 Practitioner rules to improve results

The following are some suggestions on how to improve the metamodels produced by HyGP:

1. increase mutation rate and/or mutation effect on individuals (from point to subtree mutation, for example)
2. increase population size
3. increase number of generations

4. reduce parsimony pressure (decrease parameters W_COMPLEXITY and W_SIZE)
5. use less “dangerous” or irregular primitives (first to exclude: division and reciprocal, then sin and cos that may cause noise)
6. increase number of random guesses
7. increase number of points (most effective is the DoE is balanced)

It is worth mentioning that in some cases the target values may have values so small as to result in non neglectable numerical errors in RMSE evaluation. In this cases different scaling techniques can be used. Multiplying the output by a large constant is nome cases substantially improved the quality of the metamodels evolved.

B.5 Post-processing

Once the best metamodel produced by a HyGP experiment has been identified, for example using the `posteriori` script, its properties on the training and validation data set can be computed executing the following Matlab scripts (in the given order):

1. `test_eval.m`: it evaluates the individuals on a validation data set provided by the user and writes the results in the text file `archives_best_TEST.txt`. It also shows the run that produced the metamodel performing best on the validation data set;
2. `list_selection.m`: it visualises the RMSE and R^2 distribution of the best individuals produced by each run of a GP experiment, on the training and validation data set, and compare the RMSE distribution using the Kruskal-Wallis test (see Section 3.2.2, Chapter 3);
3. `verifyGP_overlap.m`: it computes RMSE, R^2, max absolute and relative errors of a selected metamodel, both on building and validation data sets. It also plots its mathematical expression and the actual response versus the response generated by the metamodel, on both data sets. In case Matlab toolbox is enabled, it also simplifies the expression of the selected metamodel, returning it in latex form.

Appendix C

List of HyGP input settings

The input settings for the HyGP experiments described in Chapter 5 and in Chapter 6 are reported in the following sections.

C.1 Kotanchek function

HyGP input parameters for the symbolic regression of Kotanchek function through the *Omegalim* implementation are reported in Table C.1.

TABLE C.1: HyGP input parameters for the symbolic regression of Kotanchek function (*Omegalim* implementation)

number of independent runs	10
binary functions unary functions	add,sub,mult,sdiv square,cube,sin,cos,exp
Population size Max no. of generations reproduction rate crossover rate mutation rate tree generation method max. depth of randomly generated trees min. depth of randomly generated trees tree depth upper bound during evolution	200 50 0.2 0.4 0.4 ramped half and half (50% full, 50% grow method) 4 2 50
NFITCASES N_INEQUALITY0 MAX_N_PERIODS	40 0 1.5
a_2 (W_COMPLEXITY) a_3 (W_N_CORRECTIONS) a_4 (W_SIZE) a_5 (W_PEN_ORD0)	0.001 0.1 0.0001 0.0
no. of initial guesses for SQP optimiser	2
error threshold for termination	1.0E-12

C.2 Salustowicz function

HyGP input parameters for the symbolic regression of Salustowicz function through the *shift* implementation are reported in Table C.2.

TABLE C.2: HyGP input parameters for the symbolic regression of Salustowicz function (*shift* implementation)

number of independent runs	10
binary functions	add,sub,mult,sdiv
unary functions	square,cube,sin,cos,exp,neg,inv,nxp,shift
Population size	300
Max no. of generations	50
reproduction rate	0.2
crossover rate	0.4
mutation rate	0.4
tree generation method	ramped half and half (50% full, 50% grow method)
max. depth of randomly generated trees	4
min. depth of randomly generated trees	2
tree depth upper bound during evolution	50
NFITCASES	100
N_INEQUALITY0	0
a_2 (W_COMPLEXITY)	1.0e-6
a_3 (W_N_CORRECTIONS)	0.01
a_4 (W_SIZE)	1.0e-6
a_5 (W_PEN_ORD0)	0.0
no. of initial guesses for SQP optimiser	2
error threshold for termination	1.0E-12

C.3 RatPol2D function

HyGP input parameters for the symbolic regression of RatPol2D function through the *shift* implementation are reported in Table C.3.

TABLE C.3: HyGP input parameters for the symbolic regression of RatPol2D function (*shift* implementation)

number of independent runs	10
binary functions	add,sub,mult,sdiv
unary functions	square,cube,sin,cos,exp,inv,shift
Population size	200
Max no. of generations	50
reproduction rate	0.2
crossover rate	0.4
mutation rate	0.4
tree generation method	ramped half and half (50% full, 50% grow method)
max. depth of randomly generated trees	4
min. depth of randomly generated trees	2
tree depth upper bound during evolution	50
NFITCASES	40
N_INEQUALITY0	0
a_2 (W_COMPLEXITY)	0.0001
a_3 (W_N_CORRECTIONS)	0.1
a_4 (W_SIZE)	0.00001
a_5 (W_PEN_ORD0)	0.0
no. of initial guesses for SQP optimiser	2
error threshold for termination	1.0E-12

C.4 Hock function

HyGP input parameters for the symbolic regression of Hock function through the *Omegalim* implementation are reported in Table C.4.

TABLE C.4: HyGP input parameters for the symbolic regression of Hock function (*Omegalim* implementation)

number of independent runs	10
binary functions	add,sub,mult,sdiv
unary functions	square,cube,sin,cos,exp
Population size	200
Max no. of generations	50
reproduction rate	0.2
crossover rate	0.4
mutation rate	0.4
tree generation method	ramped half and half (50% full, 50% grow method)
max. depth of randomly generated trees	4
min. depth of randomly generated trees	2
tree depth upper bound during evolution	30
NFITCASES	20
N_INEQUALITY0	0
MAX_N_PERIODS	1.5
a_2 (W_COMPLEXITY)	0.01
a_3 (W_N_CORRECTIONS)	0.1
a_4 (W_SIZE)	0.001
a_5 (W_PEN_ORD0)	0.0
no. of initial guesses for SQP optimiser	2
error threshold for termination	1.0E-12

C.5 Branin-Hoo function

HyGP input parameters for the symbolic regression of Branin-Hoo function through the *10guesses* implementation are reported in Table C.5.

TABLE C.5: HyGP input parameters for the symbolic regression of Branin-Hoo function (*10guesses* implementation)

number of independent runs	10
binary functions	add,sub,mult,sdiv
unary functions	square,cube,sin,cos,exp
Population size	200
Max no. of generations	50
reproduction rate	0.2
crossover rate	0.4
mutation rate	0.4
tree generation method	ramped half and half (50% full, 50% grow method)
max. depth of randomly generated trees	4
min. depth of randomly generated trees	2
tree depth upper bound during evolution	50
NFITCASES	30
N_INEQUALITY0	0
a_2 (W_COMPLEXITY)	0.001
a_3 (W_N_CORRECTIONS)	0.1
a_4 (W_SIZE)	0.0001
a_5 (W_PEN_ORD0)	0.0
no. of initial guesses for SQP optimiser	10
error threshold for termination	1.0E-12

C.6 Rosenbrock function (PCE comparison)

The input parameters used for Rosenbrock function symbolic regression are listed in Table C.6.

TABLE C.6: GP settings for Rosenbrock function symbolic regression

number of independent runs	20
binary functions	add, sub, mult, sdiv
unary functions	shift, square, cube, sin, cos, exp
Population size	200
Max no. of generations	50
reproduction rate	0.2
crossover rate	0.4
mutation rate	0.4
tree generation method	ramped half and half (50% full, 50% grow method)
max. depth of randomly generated trees	4
min. depth of randomly generated trees	2
tree depth upper bound during evolution	30
a_2 (W_COMPLEXITY)	0.01
a_3 (W_N_CORRECTIONS)	0.1
a_4 (W_SIZE)	.001
a_5 (W_PEN_ORD0)	0.0
MAX_N_PERIODS	1.5
number of initial random guesses for SQP optimiser	2
error threshold for terminating the evolution	1.0E-7

C.7 Kotanchek function (PCE comparison)

The input parameters used for Kotanchek function symbolic regression are listed in Table C.7.

TABLE C.7: GP setttings for Kotanchek function inference

number of independent runs	12
binary functions	add,sub,mult,sdiv
unary functions	shift,square,cube,sin,cos,exp
Population size	200
Max no. of generations	50
reproduction rate	0.2
crossover rate	0.4
mutation rate	0.4
tree generation method	ramped half and half (50% full, 50% grow method)
max. depth of randomly generated trees	4
min. depth of randomly generated trees	2
tree depth upper bound during evolution	50
a_2 (W_COMPLEXITY)	0.001
a_3 (W_N_CORRECTIONS)	0.1
a_4 (W_SIZE)	.0001
a_5 (W_PEN_ORD0)	0.0
MAX_N_PERIODS	1.5
number of initial random guesses for SQP optimiser	3
error threshold for terminating the evolution	1.0E-12

C.8 10-bar truss optimisation

In Tables C.8 and C.9 the GP settings used for the symbolic regression of the axial forces and the mass are reported.

TABLE C.8: GP settings for 10-bar truss axial forces' models inference

number of independent runs	12
binary functions unary functions	add, sub, mult, sdiv, spow shift, square, cube, sin, cos, exp, nxp, inv, sinh, cosh, tanh
Population size Max no. of generations reproduction rate crossover rate mutation rate tree generation method max. depth of randomly generated trees min. depth of randomly generated trees tree depth upper bound during evolution	400 300 0.2 0.4 0.4 ramped half and half (50% full, 50% grow method) 4 2 40 (bar 1, 10), 30 (bar 2, . . . , 9)
MAX_N_PERIODS	2
a_2 (W_COMPLEXITY) a_3 (W_N_CORRECTIONS) a_4 (W_SIZE)	0.0001 0.1 0.0
max number of periods in variables' range in case sin or cos are selected no. of initial guesses for SQP optimiser	4.0 3 (bar 1), 4 (bar 2, . . . , 10)
error threshold for termination	1.0E-5

TABLE C.9: GP settings for 10-bar truss mass model inference

number of independent runs	12
binary functions unary functions	add, sub, mult, sdiv, spow shift, square, cube, inv
Population size Max no. of generations reproduction rate crossover rate mutation rate tree generation method max. depth of randomly generated trees min. depth of randomly generated trees tree depth upper bound during evolution	500 300 0.2 0.4 0.4 ramped half and half (50% full, 50% grow method) 4 2 40
MAX_N_PERIODS	2
a_2 (W_COMPLEXITY) a_3 (W_N_CORRECTIONS) a_4 (W_SIZE)	0.001 0.1 0.001
no. of initial guesses for SQP optimiser	4
error threshold for termination	1.0E-5

C.9 Hospital ward ventilation optimisation

HyGP input parameters for the symbolic regression of thermal comfort are reported in Table C.10. The settings for pathogen concentration are listed in Table C.11.

TABLE C.10: HyGP input parameters for thermal comfort symbolic regression

number of independent runs	16
binary functions	add, sub, mult, sdiv
unary functions	shift, square, cube, sin, cos, exp, nxp, inv, sinh, cosh, tanh
Population size	400
Max no. of generations	200
reproduction rate	0.2
crossover rate	0.4
mutation rate	0.4
tree generation method	ramped half and half (50% full, 50% grow method)
max. depth of randomly generated trees	4
min. depth of randomly generated trees	2
tree depth upper bound during evolution	50
NFITCASES	45
N_INEQUALITY0	32
a_2 (W_COMPLEXITY)	0.001
a_3 (W_N_CORRECTIONS)	0.1
a_4 (W_SIZE)	0.0001
a_5 (W_PEN_ORD0)	0.0001
no. of initial guesses for SQP optimiser	2
error threshold for termination	1.0E-12

TABLE C.11: HyGP input parameters for pathogen concentration symbolic regression

number of independent runs	12
binary functions	add, sub, mult
unary functions	shift, square, cube, exp, nxp, sin, cos, sinh, cosh, tanh
Population size	300
Max no. of generations	100
reproduction rate	0.2
crossover rate	0.4
mutation rate	0.4
tree generation method	ramped half and half (50% full, 50% grow method)
max. depth of randomly generated trees	4
min. depth of randomly generated trees	2
tree depth upper bound during evolution	50
NFITCASES	45
N_INEQUALITY0	32
a_2 (W_COMPLEXITY)	0.01
a_3 (W_N_CORRECTIONS)	0.1
a_4 (W_SIZE)	0.001
a_5 (W_PEN_ORD0)	0.0005
no. of initial guesses for SQP optimiser	2
error threshold for termination	1.0E-12

C.10 Chromate diffusion model

GP settings used in the experiments are reported in Table C.12 (Primer A), Table C.13 (Primer B) and Table C.14 (Primer C).

TABLE C.12: HyGP input parameters for the generation of chromate diffusion model, primer A

number of independent runs	10
binary functions unary functions	add, sub, mult, sdiv, spow shift, square, cube, exp, nxp, sin, cos, log, inv, cosh, sinh, tanh
Population size Max no. of generations reproduction rate crossover rate mutation rate tree generation method max. depth of randomly generated trees min. depth of randomly generated trees tree depth upper bound during evolution	200 100 0.2 0.4 0.4 ramped half and half (50% full, 50% grow method) 4 2 30
MAX_N_PERIODS	1.5
a_2 (W_COMPLEXITY) a_3 (W_N_CORRECTIONS) a_4 (W_SIZE)	0.001 0.1 0.02
no. of initial guesses for SQP optimiser	2
error threshold for termination	1.0E-5

TABLE C.13: HyGP input parameters for for the generation of chromate diffusion model, primer B

number of independent runs	10
binary functions unary functions	add, sub, mult, sdiv, spow shift, square, cube, exp, nxp, sin, cos, log, inv, cosh, sinh, tanh
Population size Max no. of generations reproduction rate crossover rate mutation rate tree generation method max. depth of randomly generated trees min. depth of randomly generated trees tree depth upper bound during evolution	200 100 0.2 0.4 0.4 ramped half and half (50% full, 50% grow method) 4 2 30
MAX_N_PERIODS	1.5
a_2 (W_COMPLEXITY) a_3 (W_N_CORRECTIONS) a_4 (W_SIZE)	0.00001 0.1 0.005
no. of initial guesses for SQP optimiser	2
error threshold for termination	1.0E-5

TABLE C.14: HyGP input parameters for the generation of chromate diffusion model, primer C

number of independent runs	10
binary functions unary functions	add, sub, mult, sdiv, spow shift, square, cube, exp, nxp, sin, cos, log, inv, cosh, sinh, tanh
Population size Max no. of generations reproduction rate crossover rate mutation rate tree generation method max. depth of randomly generated trees min. depth of randomly generated trees tree depth upper bound during evolution	200 100 0.2 0.4 0.4 ramped half and half (50% full, 50% grow method) 4 2 30
MAX_N_PERIODS	1.5
a_2 (W_COMPLEXITY) a_3 (W_N_CORRECTIONS) a_4 (W_SIZE)	0.0001 0.1 0.001
no. of initial guesses for SQP optimiser	2
error threshold for termination	1.0E-5

C.11 Jet pump model

HyGP settings for entrained flow rate symbolic regression are reported in Table C.15.

TABLE C.15: HyGP input parameters for supersonic jet pump entrained flow regression

number of independent runs	10
binary functions unary functions	add, sub, mult, sdiv, spow shift, square, cube, sin, cos, exp, nxp, inv, log, sinh, cosh, tanh
Population size Max no. of generations reproduction rate crossover rate mutation rate tree generation method max. depth of randomly generated trees min. depth of randomly generated trees tree depth upper bound during evolution	200 50 0.2 0.4 0.4 ramped half and half (50% full, 50% grow method) 4 2 30
MAX_N_PERIODS	1.5
a_2 (W_COMPLEXITY) a_3 (W_N_CORRECTIONS) a_4 (W_SIZE)	0.0001 0.1 0.00001
no. of initial guesses for SQP optimiser	2
error threshold for termination	1.0E-5

C.12 Bread baking oven design optimisation

HyGP settings for the regression of the temperature uniformity σ_T are reported in Table C.16.

TABLE C.16: GP settings for the regression of temperature uniformity σ_T in bread baking oven design optimisation

number of independent runs	12
binary functions unary functions	add, sub, mult, spow shift, square, cube, sin, cos, sinh, cosh, tanh, exp, nxp
Population size Max no. of generations reproduction rate crossover rate mutation rate tree generation method max. depth of randomly generated trees min. depth of randomly generated trees tree depth upper bound during evolution	300 200 0.2 0.4 0.4 ramped half and half (50% full, 50% grow method) 4 2 30
NFITCASES N_INEQUALITY0 MAX_N_PERIODS	29 120 1.5
a_2 (W_COMPLEXITY) a_3 (W_N_CORRECTIONS) a_4 (W_SIZE) a_5 (W_PEN_ORD0)	0.0001 0.1 0.00001 0.0005
no. of initial guesses for SQP optimiser	2
error threshold for termination	1.0E-5

C.13 Aerodynamic optimisation of NASA rotor 37 compressor rotor blade

The HyGP settings used for the generation of NASA rotor 37 efficiency and constraints metamodels are reported in Table C.17.

TABLE C.17: HyGP settings for NASA rotor 37 experiments

number of independent runs	10
binary functions	add, sub, mult, spow
unary functions	shift, square, cube, sin, cos, exp, nxp, log, sinh, cosh, tanh
Population size	400
Max no. of generations	200
reproduction rate	0.2
crossover rate	0.4
mutation rate	0.4
tree generation method	ramped half and half (50% full, 50% grow method)
max. depth of randomly generated trees	4
min. depth of randomly generated trees	2
tree depth upper bound during evolution	50
a_2 (W_COMPLEXITY)	0.0001
a_3 (W_N_CORRECTIONS)	0.1
a_4 (W_SIZE)	0.00001
a_5 (W_PEN_ORD0)	0.0001
max number of periods in variables' range in case sin or cos are selected	1.5
no. of initial guesses for SQP optimiser	4
error threshold for termination	1.0E-7

References

Unfinished business. *The Economist*, 390(8617), February 2009.

Alasca (Advanced Lattice Structures for Composite Airframes). Technical Report EU FP7 project, `http://cordis.europa.eu/projects/index.cfm?fuseaction=app.details&REF=97744`, 2010. [Online; last access October 2, 2013].

M. Affenzeller and S. Wagner. SASEGASA: A new generic parallel evolutionary algorithm for achieving highest quality results. *Journal of Heuristics*, 10:243–267, 2004.

M. Aichour and E. Lutton. Cooperative co-evolution inspired operators for classical GP schemes. In N. Krasnogor, G. Nicosia, M. Pavone, and D. Pelta, editors, *Proceedings of International Workshop on Nature Inspired Cooperative Strategies for Optimization (NICSO '07)*, volume 129 of *Studies in Computational Intelligence*, pages 169–178, Acireale, Italy, 2007. Springer.

HyperStudy Introduction. Altair Engineering, Inc, HyperWorks 10.0 edition, 2009.

L. Altenberg. Emergent phenomena in genetic programming. In A.V. Sebald and L.J. Fodel, editors, *Proceedings of the 3rd annual conference on evolutionary programming*, San Diego, California, USA, 1994. World Scientific, Singapore.

L. F. Alvarez. *Design optimization based on genetic programming*. PhD thesis, University of Bradford, Bradford, UK, 2000.

A. A. Ameri. NASA rotor 37 CFD code validation. Technical Report NASA/CR-2010-216235, The Ohio State University, USA, 2010.

P. J. Angeline and J. B. Pollack. Competitive environments evolve better solutions for complex tasks. In Stephanie Forrest, editor, *Proceedings of the 5th International Conference on Genetic Algorithms (GA93)*, pages 264–270, University of Illinois at Urbana-Champaign, USA, 1993. Morgan Kaufmann Publishers Inc.

U. Armani. HyGP: hybrid genetic programming implementation written in C++. *http://www.personal.leeds.ac.uk/~cnua/mypage.html*, 2011. [Online; created September 2011].

S. R. Arwade, M. Moradi, and A. Louhghalam. Variance decomposition and global sensitivity for structural systems. *Engineering structures*, 32:1–10, 2010.

A. F. Ashour, L. F. Alvarez, and V. V. Toropov. Empirical modelling of shear strength of RC deep beams by genetic programming. *Computers & Structures*, 81(5):331–338, 2003.

V. O. Balabanov and G. Venter. Response surface optimization with discrete variables. In *Proceedings of the 45th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics and Materials Conference*, Palm springs, CA, USA, 2004. AIAA 2004-1872.

W. Banzhaf. Genotype-phenotype-mapping and neutral variation - a case study in genetic programming. In Y. Davidor, H.-P. Schwefel, and R. M anner, editors, *Proceedings of the Third Conference on Parallel Problem Solving from Nature*, pages 322–332, London, UK, 1994. Springer-Verlag.

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

W. Banzhaf, F. Francone, and P. Nordin. The effect of extensive use of the mutation operator on generalization in genetic programming using sparse data sets. In H.-M. Voigt, W. Ebeling, I. Rechenberg, and H.-P. Schwefel, editors, *Parallel Problem Solving from Nature - PPSN IV*, volume 1141 of *Lecture Notes in Computer Science*, pages 300–309. Springer Berlin / Heidelberg, 1996.

W. Banzhaf, F.D. Francone, R.E. Keller, and P. Nordin. *Genetic programming: an introduction: on the automatic evolution of computer programs and its applications*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1998.

H. J. C. Barbosa and H. S. Bernardino. Genetic programming in civil, structural and environmental engineering. *Computational technology reviews*, 4:115–145, 2011.

B. Barney. Tutorials in High Performance Computing (Lawrence Livermore National Laboratory). 2011. [Online; accessed March 9, 2011].

S. J. Bates, J. Sienz, and V. V. Toropov. Formulation of the optimal latin hypercube design of experiments using a permutation genetic algorithm. In *Proceedings of the*

45th AIAA/ASME/ASCE/AHS/ASC structures, structural dynamics & materials conference, Palm Springs, California, USA, 2004.

R. Bellman. *Adaptive control processes: a guided tour*. Princeton University Press, Princeton, N.J., 1961.

H.-G. Beyer and H.-P. Schwefel. Evolution strategies. a comprehensive introduction. *Natural Computing*, 1:3–52, 2002.

C. M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, 1995.

S. Bleuler, M. Brack, L. Thiele, and E. Zitzler. Multiobjective genetic programming: reducing bloat using SPEA2. In *Proceedings of the 2001 Congress on Evolutionary Computation (CEC2001)*, pages 536–543, Seoul, Korea, 2001. IEEE Press.

T. Blickle. Evolving compact solutions in genetic programming: A case study. In H.-M. Voigt, W. Ebeling, I. Rechenberg, and H.-P. Schwefel, editors, *Parallel Problem Solving From Nature IV. Proceedings of the International Conference on Evolutionary Computation*, volume 1141 of *LNCS*, pages 564–573, Berlin, Germany, 1996. Springer-Verlag.

T. Blickle and L. Thiele. Genetic programming and redundancy. In J. Hopf, editor, *Genetic Algorithms within the Framework of Evolutionary Computation (Workshop at KI-94, Saarbrücken)*, pages 33–38. Max-Planck-Institut für Informatik (MPI-I-94-241), 1994. URL `http://www.tik.ee.ethz.ch/~tec/publications/bt94/GPandRedundancy.ps.gz`.

T. Blickle and L. Thiele. A comparison of selection schemes used in evolutionary algorithms. *Evolutionary Computation*, 4(4):361–394, 1997.

M.H.B. Bonte, A.H. van den Boogaard, and J. Huétink. Solving optimisation problems in metal forming using finite element simulation and metamodelling techniques. In D. Büche and N. Hofmann, editors, *Proceedings of the Automatic Process Optimization in Materials Technology (APOMAT) Conference*, pages 242–251, Morschach, Switzerland, 2005.

M. Brameier and W. Banzhaf. *Linear Genetic Programming*. Number XVI in Genetic and Evolutionary Computation. Springer, 2007.

M. Brameier, W. Kantschik, P. Dittrich, and W. Banzhaf. SYSGP – a C++ library of different GP variants. Technical Report of the Collaborative Research Centre 531 *Computational Intelligence* CI–48/98, University of Dortmund, 1998.

T. Buchsbaum. Toward a winning GP strategy for continuous nonlinear dynamical system identification. In D. Srinivasan and L. Wang, editors, *IEEE Congress on Evolutionary Computation*, pages 1269–1275. IEEE press, 2007.

L. Bull. Dedication: Dr. Lawrence J. Fogel (1928-2007). *Evolutionary Intelligence*, 1:1–1, 2008.

B. Chapman, G. Jost, and R. van der Pas. *Using OpenMP: Portable Shared Memory Parallel Programming*. The MIT Press, Cambridge, MA, USA, 2007.

K. Chellapilla. Evolving computer programs without subtree crossover. *IEEE Transactions on Evolutionary Computation*, 1(3), 1997.

D. Chetwynd, K. Worden, and G. Manson. An application of interval-valued neural networks to a regression problem. *Proceedings of the Royal Society A*, 462:3097–3114, 2006.

K. K. Choi, B. Youn, and R.-J. Yang. Moving least squares method for reliability-based design optimization. In G. Cheng, Y. Gu, S. Liu, and Y. Wang, editors, *Proceedings of the Fourth World Congress of Structural and Multidisciplinary Optimization*, Dalian, China, 2001.

P. Collet and M. Schoenauer. GUIDE: unifying evolutionary engines through a graphical user interface. In P. Liardet, P. Collet, C. Fonlupt, E. Lutton, and M. Schoenauer, editors, *Artificial Evolution*, volume 2936 of *Lecture Notes in Computer Science*, pages 203–215. Springer Berlin / Heidelberg, 2004.

P. Collet, E. Lutton, F. Raynal, and M. Schoenauer. Polar IFS+parisian genetic programming=efficient IFS inverse problem solving. *Genetic Programming and Evolvable Machines*, 1(4):339–361, 2000.

N. L. Cramer. A representation for the adaptive generation of simple sequential programs. In J. J. Grefenstette, editor, *Proceedings of the first International Conference on Genetic Algorithms and their applications*, pages 183–187, Carnegie-Mellon University, Pittsburgh, PA, USA, 1985.

E. D. De Jong and J. B. Pollack. Multi-objective methods for tree size control. *Genetic Programming and Evolvable Machines*, 4:211–233, 2003.

K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.

J. Dunham, G. Meauzk, V. Couaillier, C. Hirsch, F. Leboeuf, G. K. Serovy, F. Bassi, R. A. Deliney, L. A. Povinelli, W. J. Calvert, J. J. Salva, and W. F. 0'Brien. CFD validation for propulsion system components. Technical Report AGARD AR-355, North Atlantic Treaty Organization, 1998.

M. C. Duta and M. B. Giles. The use of automatic differentiation for adjoint CFD codes. In *Proceedings of the European Conference on Computational Fluid Dynamics (ECCOMAS CFD)*, Egmond aan Zee, The Netherlands, 2006.

A. E. Eiben and M. Schoenauer. Evolutionary computing. *Information Processing Letters*, 82:1–6, 2002.

M.S. Eldred, C. G. Webster, and P. G. Constantine. Evaluation of non-intrusive approaches for Wiener-Askey generalized polynomial chaos. *American Institute of Aeronautics and Astronautics*, (1892):1–22, 2008.

I. Elishakoff, R. T. Haftka, and J. Fang. Structural design under bounded uncertainty - optimization with anti-optimization. *Computers & Structures*, 53(6):1401–1405, 1994.

J. Eves, V.V. Toropov, H. M. Thompson, N. Kapur, J. Fan, D. Copley, and A. Mincher. Design optimization of supersonic jet pumps using high fidelity flow analysis. *Structural and Multidisciplinary Optimization*, 2011.

C. Ferreira. Gene expression programming: a new adaptive algorithm for solving problems. *Complex Systems*, 13(2):87–129, 2001.

C. Ferreira. Function finding and the creation of numerical constants in gene expression programming. In *Proceedings of the 7th online world conference on Soft Computing in Industrial Applications (WSC7)*, September 2002.

I. Flood. Current and future trends in the use of artificial neural networks in engineering. *Computational technology reviews*, 4:93–114, 2011.

C. M. Focken. *Dimensional Methods and Their Applications*. Edward Arnold & Co., London, UK, 1st edition, 1953.

D. B. Fogel. An introduction to simulated evolutionary optimization. *IEEE Transactions on Neural Networks*, 5(1):3 –14, 1994.

L. J. Fogel. *On the organization of intellect*. PhD thesis, UCLA, USA, 1964.

C. M. Fonseca and P. J. Fleming. An overview of evolutionary algorithms in multiobjective optimization. *Evolutionary Computation*, 3(1):1–16, 1995.

J. H. Friedman. Multivariate adaptive regression splines. *The Annals of Statistics*, 19(1): 1–141, 1991.

S. A. Furman, F. H. Scholes, A. E. Hughes, and D. Lau. Chromate leaching from inhibited primers - part II: Modelling of leaching. *Progress in Organic Coatings*, 56:33–38, 2006.

C. Gagné and M. Parizeau. Open BEAGLE: A new C++ evolutionary computation framework. In W. B. Langdon, E. Cantú-Paz, K. Mathias, R. Roy, D. Davis, R. Poli, K. Balakrishnan, V. Honavar, G. Rudolph, J. Wegener, L. Bull, M. A. Potter, A. C. Schultz, J. F. Miller, E. Burke, and N. Jonoska, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2002)*, New York, 2002. Morgan Kaufmann Publishers.

J. Garcke, M. Hegland, and O. Nielsen. Parallelisation of sparse grids for large scale data analysis. In *Proceedings of the 2003 International Conference on Computational Science: PartIII*, ICCS'03, pages 683–692, Berlin, Heidelberg, 2003. Springer-Verlag.

N. Garshina and C. Vladislavleva. On development of a complexity measure for symbolic regression via genetic programming. Technical Report Modeling Report for Dow Benelux B.V. Eindhoven, Technische Universiteit, Eindhoven, The Netherlands, 2004.

S. Gelly, O. Teytaud, N. Bredeche, and M. Schoenauer. Universal consistency and bloat in GP. *Revue d'Intelligence Artificielle*, 20(6):805–827, 2006.

G. Georgiou and J. E. Cooper. Virtual prototyping and MDO of aircraft structures. In *DiPaRTLoads and Aeroelastics Workshop (http://www.stirling-dynamics.co.uk/images/stories/dipart2011/)*, Bristol, December 2011.

M. Giacobini, M. Tomassini, and L. Vanneschi. Limiting the number of fitness cases in genetic programming using statistics. In J. Guervós, P. Adamidis, H.-G. Beyer, H.-P.l Schwefel, and J.-L. Fernández-Villacañas, editors, *Parallel Problem Solving from Nature*

- *PPSN VII*, volume 2439 of *Lecture Notes in Computer Science*, pages 371–380. Springer Berlin / Heidelberg, 2002.

A. Grama, A. Gupta, G. Karypis, and V. Kumar. *Introduction to parallel computing*. Pearson Education, Ltd., third edition, 2008.

G. J. Gray, D. J. Murray-Smith, Y. Li, and K. C. Sharman. Nonlinear model structure identification using genetic programming. In *Proceedings of the 1996 Genetic Programming Conference (GP-96)*, Stanford University, CA, USA, 1996.

G.J. Gray, T. Weinbrenner, D.J. Murray-Smith, Yun Li, and K.C. Sharman. Issues in nonlinear model structure identification using genetic programming. In *Genetic Algorithms in Engineering Systems: Innovations and Applications, Conference Publication No. 446*, pages 308–313, 1997.

P. Gustafson. Detecting and avoiding OpenMP race conditions in C++. 2011. URL `http://developers.sun.com/solaris/articles/cpp_race.html`. [Online; accessed 11 March 2011].

Maryam Amir Haeri, Mohammad Mehdi Ebadzadeh, and Gianluigi Folino. Statistical genetic programming: the role of diversity. In *Proceedings of the 17th Online Conference on Soft Computing in Industrial Applications Anywhere on Earth (WCS17)*, 2012.

R. T. Haftka and Z. Gürdal. *Elements of Structural Optimization*. Kluwer Academic Publishers, third edition, 1993.

S. Harding and W. Banzhaf. Fast genetic programming on GPUs. In M. Ebner, M. O'Neill, A. Ekárt, L. Vanneschi, and A. I. Esparcia-Alcázar, editors, *Proceedings of the 10th European Conference on Genetic Programming*, volume 4445 of *Lecture Notes in Computer Science*, Valencia, Spain, 11-13 April 2007. Springer.

F. Harewood, R. Thornton, and P. Sharp. Step change in design: Exploring sixty stent design variations overnight. In *Proceedings of the biennial Altair CAE Technology Conference*, UK, 2007.

S. Harmeling, G. Dornhege, D. Tax, F. Meinecke, and K.-R. Müller. From outliers to prototypes: Ordering data. *Neurocomputing*, 69:1608–1618, 2006.

J. C. Helton and F. J. Davis. Latin hypercube sampling and the propagation of uncertainty in analyses of complex systems. *Reliability engineering and system safety*, 81:23–69, 2003.

J. E. Herencia and R.T. Haftka. Structural optimization with limited number of element properties. *Structural and Multidisciplinary Optimization*, 41(5):817–820, 2010.

W. Hock and K. Schittkowski. Test examples for nonlinear programming codes. *Lecture notes in economics and mathematical systems*, 187, 1981. URL `http://www.klaus-schittkowski.de`.

J. P. Hoeflinger. *Extending OpenMP* to clusters*. White paper Intel Corporation, 2006.

M. Hofwing, N. Strömberg, and M. Tapankov. Optimal polynomial regression models by using a genetic algorithm. In Y. Tsompanakis and B.H.V. Topping, editors, *Proceedings of the Second International Conference on Soft Computing Technology in Civil, Structural and Environmental Engineering*, Chania, Greece, 2011. Civil-Comp Press.

J.H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, Michigan, USA, 1975.

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

Myles Hollander and Douglas A. Wolfe. *Nonparametric Statistical Methods, 2nd Edition*. Wiley-Interscience, 1999.

M. Hollick and H. Kuhlmann. Genetic programming in C/C++. CSE99/CIS899 final report, Computer and Information Science department, University of Pennsylvania, Philadelphia, USA, 1995. GP code downloadable from http://www.cis.upenn.edu/ hollick/genetic/paper2.html (accessed October 2008).

G. S. Hornby. ALPS: the age-layered population structure for reducing the problem of premature convergence. In *Proceedings of the 8th annual conference on Genetic and evolutionary computation (GECCO '06)*, pages 815–822, New York, NY, USA, 2006. ACM.

H. Iba and M. Terao. Controlling effective introns for multi-agent learning by genetic programming. In L. D. Whitley, D. E. Goldberg, E. Cantú-Paz, L. Spector, I. C. Parmee, and H.-G. Beyer, editors, *Proceedings of the genetic and and evolutionary computation conference (GECCO '00)*, pages 419–426, Las Vegas, Nevada, USA, 2000.

R. Jin, W. Chen, and T.W. Simpson. Comparative studies of metamodelling techniques under multiple modelling criteria. *Structural and Multidisciplinary Optimization*, 23: 1–13, 2001.

B. E. Johanson and R. Poli. GP-Music: An interactive genetic programming system for music generation with automated fitness raters. Technical Report CSRP-98-13, University of Birmingham, School of Computer Science, 1998.

T. Kalganova and J. Miller. Evolving more efficient digital circuits by allowing circuit layout evolution and multi-objective fitness. In A. Stoica, J. Lohn, and D. Keymeulen, editors, *Proceedings of the First NASA/DoD Workshop on Evolvable Hardware (EH'99)*, pages 54–63, Pasadena, CA, USA, 1999. IEEE Computer Society.

W. Kantschik and W. Banzhaf. Linear-tree GP and its comparison with other GP structures. In Julian F. Miller, M. Tomassini, P. Lanzi, C. Ryan, A. G. B. Tettamanzi, and W. B. Langdon, editors, *Genetic Programming, Proceedings of EuroGP'2001*, volume 2038 of *LNCS*, pages 302–312, Lake Como, Italy, 18-20 April 2001a. Springer-Verlag.

W. Kantschik and W. Banzhaf. Linear-tree GP and its comparison with other GP structures. In J. F. Miller, M. Tomassini, P. L. Lanzi, C. Ryan, A. G. B. Tettamanzi, and W. B. Langdon, editors, *Proceedings of EuroGP'2001*, volume 2038 of *LNCS*, pages 302–312, Lake Como, Italy, 2001b. Springer-Verlag.

M. Keijzer. Improving symbolic regression with interval arithmetic and linear scaling. In C. Ryan, T. Soule, M. Keijzer, E. Tsang, R. Poli, and E. Costa, editors, *Proceedings of EuroGP 2003*, volume 2610 of *LNCS*, pages 70–82. Springer-Verlag, 2003.

M. Keijzer and V. Babovic. Dimensionally aware genetic programming. In W. Banzhaf, J. Daida, A.E. Eiben, M.H. Garzon, V. Honavar, M. Jakiela, and R.E. Smith, editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 2, pages 1069–1076. Morgan Kaufmann, 1999.

M. Keijzer, M. Baptist, V. Babovic, and J. R. Uthurburu. Determining equations for vegetation induced resistance using genetic programming. In *Proceedings of the 2005 conference on genetic and evolutionary computation (GECCO '05)*, pages 1999–2006, New York, NY, USA, 2005. ACM.

E. Kermani, Y. Jafarian, and M. H. Baziar. New predictive models for the v_{max}/a_{max} ratio of strong ground motions using genetic programming. *International Journal of Civil Engineering*, 7(4), 2009.

M. A. I. Khan, C. J. Noakes, and V. V. Toropov. Development of a numerical optimization approach to design ventilation for both infection control and comfort. *To be submitted to Journal of Building Simulation*, 2011a.

M. A. I. Khan, C. J. Noakes, and V. V. Toropov. Optimization of ventilation system design and operation in hospital wards. In *Proceedings of the Twelfth International Conference on Indoor Air Quality and Climate (Indoor Air 2011)*, Austin, Texas, USA, 2011b.

Z. Khatir, J. Paton, H. Thompson, N. Kapur, V.V. Toropov, M. Lawes, and D. Kirk. Computational fluid dynamics (cfd) investigation of air flow and temperature distribution in a small scale bread-baking oven. *Applied Energy*, In Press, Corrected Proof, 2011a.

Z. Khatir, H. Thompson, N. Kapur, V.V. Toropov, J. Paton, and M. Lawes. The application of computational fluid dynamics (cfd) and oven design optimisation in the british bread-baking industry. In *Proceedings of the Eighth International Conference on CFD in Oil & Gas, Metallurgical and Process Industries (SINTEF/NTNU)*, Trondheim, Norway, 2011b.

J. Knowles and D. Corne. The pareto archived evolution strategy : a new baseline algorithm for pareto multiobjective optimisation. In *Proceedings of the 1999 Congress on Evolutionary Computation*, pages 98–105, Piscataway, NJ, 1999. IEEE Press.

N.A. Kondratenko and V.P. Sherstyuk. Spectroscopic characteristics of Cr(VI) oxyanions in water solutions. *Theoretical and Experimental Chemistry*, 22:656–662, 1986.

A. Kordon and C.-T. Lue. Symbolic regression modeling of blown film process effects. In *Proceedings of the 2004 IEEE Congress on Evolutionary Computation*, pages 561–568, Portland, Oregon, 2004. IEEE Press.

M. F. Korns. Accuracy in symbolic regression. In R. Riolo, E. Vladislavleva, and J. H. Moore, editors, *Genetic Programming Theory and Practice IX*, Genetic and Evolutionary Computation. Springer New York, 2011.

J. R. Koza. *Genetic programming: on the programming of computers by means of natural selection*. MIT press, Cambridge, MA, USA, 6th edition, 1992.

I. Kroo. Aeronautical applications of evolutionary design. In H. Deconinck, J. Periaux, and K. Giannakoglou, editors, *VKI Lecture Series on Optimization Methods & Tools for Multicriteria/Multidisciplinary Design*. Von Karman Institute for Fluid Dynamics, 2004.

U. Kutschera. A comparative analysis of the Darwin-Wallace papers and the development of the concept of natural selection. *Theory in Biosciences*, 122:343–359, 2003.

L. Lamberti and C. Pappalettere. A fast big bang-big crunch optimization algorithm for weight minimization of truss structures. In Y. Tsompanakis and B. H. V. Topping, editors, *Proceedings of the second international conference on soft computing technology in civil, structural and environmental engineering (CSC 2011)*, Chania, Crete, Greece, 2011.

W. Langdon and R. Poli. Fitness causes bloat: Mutation. In W. Banzhaf, R. Poli, M. Schoenauer, and T. Fogarty, editors, *Genetic Programming*, volume 1391 of *Lecture Notes in Computer Science*, pages 37–48. Springer Berlin / Heidelberg, 1998a.

W. B. Langdon. The evolution of size in variable length representations. In *Proceedings of the 1998 IEEE International Conference on Evolutionary Computation*, pages 633–638. IEEE Press, 1998.

W. B. Langdon. Size fair and homologous tree crossovers for tree genetic programming. *Genetic Programming and Evolvable Machines*, 1(1/2), 2000.

W. B. Langdon and R. Poli. Genetic programming bloat with dynamic fitness. In W. Banzhaf, R. Poli, M. Schoenauer, and T. C. Fogarty, editors, *Proceedings of the First European Workshop on Genetic Programming (EuroGP '98)*, volume 1391 of *LNCS*, Paris, 1998b. Springer-Verlag.

W. B. Langdon, T. Soule, R. Poli, and J. A. Foster. The evolution of size and shape. In L. Spector, W. B. Langdon, U-M O'Reilly, and P. J. Angeline, editors, *Advances in genetic programming III*, chapter 8. MIT Press, Cambridge, MA, USA, 1999.

M. Laumanns, L. Thiele, K. Deb, and E. Zitzler. Archiving with guaranteed convergence and diversity in multi-objective optimization. In *Genetic and Evolutionary Computation Conference (GECCO 2002)*, pages 439–447, New York, NY, USA, 2002. Morgan Kaufmann Publishers.

T. L. Lew, A. B. Spencer, F. Scarpa, K. Worden, A. Rutherford, and F. Hemez. Identification of response surface models using genetic programming. *Mechanical Systems and Signal Processing*, 20:1819–1831, 2006.

R. J. Lloyd. Gaussian Process Structure Search. `https://github.com/jamesrobertlloyd/gp-structure-search`, 2012. [Online; accessed September 1, 2013].

E. L. Loweth, G. N. de Boer, and V. V. Toropov. Practical recommendations on the use of moving least squares metamodel building. In B. H. V. Topping and Y. Tsompanakis, editors, *Proceedings of the thirteenth international conference on civil, structural and environmental engineering computing (CSC2011)*, Chania, Greece, 2011. Civil-Comp Press.

S. Luke. ECJ 20. A Java-based Evolutionary Computation Research System. `http://cs.gmu.edu/~eclab/projects/ecj/`, 2010. [Online; accessed April 5, 2011].

S. Luke and L. Panait. Fighting bloat with nonparametric parsimony pressure. In J. Guervós, P. Adamidis, H.-G. Beyer, H.-P. Schwefel, and J.-L. Fernández-Villacañas, editors, *Parallel Problem Solving from Nature - PPSN VII*, volume 2439 of *Lecture Notes in Computer Science*, pages 411–421. Springer Berlin / Heidelberg, 2002a.

S. Luke and L. Panait. Lexicographic parsimony pressure. In W. B. Langdon, E. Cantú-Paz, K. Mathias, R. Roy, D. Davis, R. Poli, K. Balakrishnan, V. Honavar, G. Rudolph, J. Wegener, L. Bull, M. A. Potter, A. C. Schultz, J. F. Miller, E. Burke, and N. Jonoska, editors, *Proceedings of the GECCO 2002 Genetic and Evolutionary Computation Conference*, pages 829–836, New York, 2002b. Morgan Kaufmann Publishers.

S. Luke and L. Spector. Evolving teamwork and coordination with genetic programming. In John R. Koza, David E. Goldberg, David B. Fogel, and Rick L. Riolo, editors, *Proceedings of the First Annual Conference on Genetic Programming*, pages 150–156, Stanford University, CA, USA, 1996. MIT Press.

S. Luke and L. Spector. A comparison of crossover and mutation in genetic programming. In J. R. Koza, K. Deb, M. Dorigo, D. B. Fogel, M. Garzon, H. Iba, and R. L. Riolo, editors, *Proceedings of the Second Annual Conference on Genetic Programming*, Stanford University, CA, USA, 1997. Morgan Kaufmann.

S. Luke and L. Spector. A revised comparison of crossover and mutation in genetic programming. In J. R. Koza, Wolfgang Banzhaf, K. Chellapilla, K. Deb, M. Dorigo, D. B.

Fogel, M. H. Garzon, D. E. Goldberg, H. Iba, and R. Riolo, editors, *Proceedings of the Third Annual Conference on Genetic Programming*, University of Wisconsin, Madison, Wisconsin, USA, 1998. Morgan Kaufmann.

C. D. MacLean, E. A. Wollesen, and B. Worzel. Listening to data: Tuning a genetic programming system. In U.-M. O'Reilly, T. Yu, R. Riolo, and B. Worzel, editors, *Genetic Programming Theory and Practice II*, pages 245–262, New York, NY, USA, 2005. Springer Science+Business Media, Inc.

K. Madsen, H. B. Nielsen, and J. Sondergaard. Robust subroutines for non-linear optimization. Technical report IMM-REP-2002-02, Technical University of Denmark, 2002.

R.T. Marler and J.S. Arora. Survey of multi-objective optimization methods for engineering. *Structural and Multidisciplinary Optimization*, 26:369–395, 2004.

W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of mathematical biophysics*, 5:115–133, 1943.

H. Meuer, E. Strohmaier, J. Dongarra, and H. Simon. TOP500 Project. *http://www.top500.org*, 1993. [Online; accessed November 28, 2010].

Z. Michalewicz. *Genetic Algorithm + Data Structures = Evolution Programs*. Springer-Verlag, New York, USA, 3rd edition, 1996.

J. F. Miller and P. Thomson. Cartesian genetic programming. In *Proceedings of the Third European Conference on Genetic Programming (EuroGP 2000)*, volume 1802, pages 121–132. Springer-Verlag, 2000.

K. Mohammadi and S. J. Seyyed Mahdavi. On improving training time of neural networks in mixed signal circuit fault diagnosis applications. *Microelectronics Reliability*, 48:781–793, 2008.

D. J. Montana. Strongly typed genetic programming. *Evolutionary Computation*, 3(2): 199–230, 1995.

D. C. Montgomery, E. A. Peck, and G. G. Vining. *Introduction to Linear Regression Analysis (Wiley Series in Probability and Statistics)*. Wiley & Sons, Inc., 2006.

D. R. Munroe. Genetic programming: the ratio of crossover to mutation as a function of time. *Research Letters in the Information and Mathematical Sciences*, 6:83–96, 2004.

I. T. Nabney. Efficient training of RBF networks for classification. *International Journal of Neural Systems*, 14(3):1–8, 2003.

S. Niemann, B. Kolesnikov, H. Lohse-Busch, C. Hühne, D. Liu, V. V. Toropov, and O. M. Querin. The use of topology optimisation in the conceptual design of a next generation lattice composite fuselage structure. In *Proceedings of the 3rd Aircraft structural design conference*, Delft, Netherlands, October 2012.

P. Nordin and W. Banzhaf. Complexity compression and evolution. In L. Eshelman, editor, *Proceedings of the 6th International Conference on Genetic Algorithms (ICGA-95)*, pages 310–317, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc.

P. Nordin, F. D. Francone, and W. Banzhaf. Explicitly defined introns and destructive crossover in genetic programming. In K. Jr. Kinnear. and P. Angeline, editors, *Advances in genetic programming II*. MIT Press, Cambridge, MA, USA, 1996.

P. Nordin, W. Banzhaf, and F. D. Francone. Efficient evolution of machine code for CISC architectures using instruction blocks and homologous crossover. In L. Spector, W. B. Langdon, U-M O'Reilly, and P. J. Angeline, editors, *Advances in genetic programming III*, chapter 12. MIT Press, Cambridge, MA, USA, 1999.

H.O. Nyongesa, S. Kent, and R. O'Keefe. Genetic programming for anti-air missile proximity fuze delay-time algorithms. *Aerospace and Electronic Systems Magazine, IEEE*, 16 (1):41–45, 2001.

Y. S. Ong, P. B. Nair, and A. J. Keane. Evolutionary optimization of computationally expensive problems via surrogate modeling. *AIAA Journal*, 41(4):687–696, 2003.

M.J.L. Orr. Introduction to radial basis function networks. `http://www.anc.ed.ac.uk/rbf/intro/intro.html`, 1996. [Online; accessed June 14, 2010].

A. Oyama and S. Obayashi. Transonic axial-flow blade shape optimization using evolutionary algorithm and three-dimensional Navier-Stokes solver. In *Proceedings of the 9th AIAA/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, Atlanta, Georgia, 2002.

C. S. Perone. Pyevolve. `http://pyevolve.sourceforge.net/`, 2009. [Online; accessed April 4, 2011].

S. G. Pierce, K. Worden, and G. Manson. A novel information-gap technique to assess reliability of neural network-based damage detection. *Journal of Sound and Vibration*, 293:96–111, 2006.

S. G. Pierce, K. Worden, and A. Bezazi. Uncertainty analysis of a neural network used for fatigue lifetime prediction. *Mechanical Systems and Signal Processing*, 22:1395–1411, 2008.

R. Poli and William B. Langdon. On the search properties of different crossover operators in genetic programming. In J. R. Koza, W. Banzhaf, K. Chellapilla, K. Deb, M. Dorigo, D. B. Fogel, M. H. Garzon, D. E. Goldberg, H. Iba, and R. Riolo, editors, *Proceedings of the Third Annual Conference of Genetic Programming 1998*, Madison, Wisconsin, USA, 1998. Morgan Kaufmann.

R. Poli, W. B. Langdon, and N. F. McPhee. *A field guide to genetic programming*. Published via http://lulu.com and freely available at http://www.gp-field-guide.org.uk, 2008. [Online; accessed February 18, 2009].

A. Polynkin and V. V. Toropov. Mid-range metamodel assembly building based on linear regression for large scale optimization problems. *Structural and Multidisciplinary Optimization*, 45:515–527, 2011.

A. Polynkin, V. V. Toropov, and S. Shahpar. Adaptive and parallel capabilities in the multipoint approximation method. In *Proceedings of the 12th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, Victoria, British Columbia, Canada, 2008.

S. Prata. *C++ Primer Plus*. Sams Publishing, Indianapolis, Indiana, USA, fifth edition, 2005.

T. Prosek and D. Thierry. A model for the release of chromate from organic coatings. *Progress in Organic Coatings*, 49:209–217, 2004.

B. Punch and D. Zongker. lil-gp genetic programming system. *http://garage.cse.msu.edu/software/lil-gp/*, 1998. [Online; accessed April 4, 2011].

A. Quarteroni, R. Sacco, and F. Saleri. *Matematica numerica*. Springer-Verlag, Milan, Italy, 2nd edition, 2000.

M. Ramu, V. Prabhu Raja, P. R. Thyla, and M. Gunaseelan. Design optimization of complex structures using metamodels. *Jordan Journal of Mechanical and Industrial Engineering*, 4(5), 2010.

C. E. Rasmussen. Documentation for GPML Matlab Code version 3.2. *http://www.gaussianprocess.org/gpml/code/matlab/doc/*, 2006. [Online; accessed August 15, 2013].

C. E. Rasmussen and H. Nickish. Gaussian processes for machine learning (gpml) toolbox. *Journal of Machine Learning Research*, 11:3011–3015, 2010.

C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning*. The MIT Press, 2006.

T. C. S. Rendall and C.B. Allen. Multidimensional aircraft surface pressure interpolation using radial basis functions. In *I MECH E Part G Journal of Aerospace Engineering*, volume 222, pages 483–495, 2008.

M. D. Richards, D. Whitley, J. Ross Beveridge, T. Mytkowicz, D. Nguyen, and D. Rome. Evolving cooperative strategies for UAV teams. In *Proceedings of the 2005 conference on Genetic and Evolutionary computation (GECCO '05)*, pages 1721–1728, New York, NY, USA, 2005. ACM.

J. L. Jr. Rogers and W. J. II LaMarsh. Reducing neural network training time with parallel processing. Technical Report NASA Technical Memorandum 110154, NASA, Langley Research Center, Hampton, Virginia, USA, 1995.

F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958.

C. Ryan, J. J. Collins, and M. O. Neill. Grammatical evolution: evolving programs for an arbitrary language. In W. Banzhaf, R. Poli, M. Schoenauer, and T. C. Fogarty, editors, *Proceedings of the First European Workshop on Genetic Programming (EuroGP98)*, volume Lecture Notes in Computer Science 1391, pages 83–95. Springer - Verlag, 1998.

P. Sætrom and M. L. Hetland. Multiobjective evolution of temporal rules. In *Proceedings of the 8th Scandinavian Conference on Artificial Intelligence (SCAI 2003)*, Bergen, Norway, 2003. IOS Press.

A. Samad and K. Y. Kim. Shape optimization of an axial compressor blade by multi-objective genetic algorithm. *Proceedings of the Institution of Mechanical Engineers, Part A: Journal of Power and Energy*, 222(6):599–611, 2008.

M. Schmidt and H. Lipson. Distilling free-form natural laws from experimental data. *Science*, 324(5923):81–85, 2009a.

M. Schmidt and H. Lipson. Cornell Creative Machines Lab/ Eureqa webpage. `http://creativemachines.cornell.edu/eureqa`, 2009b. [Online; accessed September 18, 2013].

M. Schmidt and H. Lipson. Symbolic regression of implicit equations. In R. Riolo et al., editor, *Genetic Programming Theory and Practice VII*. Springer Science, 2010.

M. Schoenauer, M. Sebag, F. Jouve, B. Lamy, and H. Maitournam. Evolutionary identification of macro-mechanical models. In P. J. Angeline and K. E. Kinnear Jr., editors, *Advances in genetic programming II*, chapter 23, pages 467–488. MIT Press, Cambridge, MA, USA, 1996.

F. H. Scholes, S. A. Furman, A. E. Hughes, T. Nikpour, N. Wright, P. R. Curtis, C. M. Macrae, S. Intem, and A. J. Hill. Chromate leaching from inhibited primers - part I: Characterisation of leaching. *Progress in Organic Coatings*, 56:23–32, 2006.

M. Sebag, M. Schoenauer, and H. Maitournam. Parametric and non-parametric identification of macro-mechanical models. In D. Quagliarella, J. Periaux, C. Poloni, and G. Winter, editors, *Genetic Algorithms and Evolution Strategies in Engineering and Computer Sciences*, pages 327–340. John Wiley & Sons, 1997.

S. Shahpar. SOFT: A new design and optimisation tool for turbomachinery. In K. Giannakoglou, D. Tsahalis, J. Périaux, K. Papailiou, and T. Fogarty, editors, *Evolutionary methods for design, optimisation and control*. CIMNE, Barcelona, 2002.

S. Shahpar. SOPHY: An integrated CFD based automatic design optimisation system. In *Proceedings of the 17th Symposium on Air Breathing Engines*, Munich, Germany, 2005.

S. Shahpar and B.L. Lapworth. PADRAM: Parametric design and rapid meshing system for turbomachinery optimisation. In *Proceedings of the ASME Turbo Expo Conference*, number Paper GT-2003-38698, Atlanta, Georgia, USA, 2003.

S. Shahpar, A. Polyinkin, and V.V. Toropov. Large scale optimization of transonic axial compressor rotor blades. In *Proceedings of the 49th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics and Materials Conference*, Schaumburg, Illinois, USA, 2008. Conference Proceeding series. AIAA.

D. Shaw, J. Miles, and A. Gray. Genetic programming within civil engineering. In *Proceedings of the 6th international conference on Adaptive Computing in Design and Manufacture (ACDM 2004)*. AIAA, 2004.

E. L. Silva, P. J. G. Lisboa, and A. G. Carmona. Regression with radial basis function artificial neural networks using qlp decomposition to prune hidden nodes with different functional form. In *Proceedings of the 8th WSEAS International Conference on Neural Networks*, Vancouver, British Columbia, Canada, 2007.

S. Silva. GPLAB - a genetic programming toolbox for MATLAB. `http://gplab.sourceforge.net/index.html`, 2003. [Online; accessed June 13, 2011].

T. W. Simpson, J. D. Peplinski, P. N. Koch, and J. K. Allen. Metamodels for computer-based engineering design: survey and recommendations. *Engineering with computers*, 17:129–150, 2001.

K. Sims. Karl Sims webpage. `http://www.karlsims.com`, 1987. [Online; accessed August 15, 2011].

K. Sims. Interactive evolution of equations for procedural models. *The Visual Computer*, 9(8):466–476, 1993.

A. P. Singh, V. Mani, and R. Ganguli. Genetic programming metamodel for rotating beams. *Computer Modeling in Engineering and Sciences*, 21(2):133–148, 2007.

A. Singleton. Andy's Genetic Programming - GPQuick and Doodle Garden. `http://www.assembla.com/spaces/andysgp/wiki`, 1993. [Online; accessed August 16, 2011].

G. Smits and M. Kotanchek. Pareto-front exploitation in symbolic regression. In U. O'Reilly, T. Yu, R. L. Riolo, and B. Worzel, editors, *Genetic Programming Theory and Practice II*, chapter 17, pages 283–299. Springer, 2004.

G. Smits, A. Kordon, K. Vladislavleva, E. Jordaan, and M. Kotanchek. Variable selection in industrial datasets using pareto genetic programming. In T. Yu, R. L. Riolo, and

B. Worzel, editors, *Genetic Programming Theory and Practice III*, chapter 6, pages 79–92. Springer, 2005.

E. Snelson, C. E. Rasmussen, and Z. Ghahramani. Warped gaussian processes. *Advances in Neural Information Processing Systems*, 16:337–344, 2004.

I. M. Sobol. Sensitivity estimates for nonlinear mathematical models. *Mathematical Modelling and Computational Experiment*, 1(4):407–414, 1993.

T. Soule and J. A. Foster. Effects of code growth and parsimony pressure on populations in genetic programming. *Evolutionary computation*, 6(4):293–309, 1998a.

T. Soule and J.A. Foster. Removal bias: a new cause of code growth in tree based evolutionary programming. In *Proceedings of the 1998 IEEE International Conference on Evolutionary Computation*, pages 781 –786. IEEE Press, 1998b.

T. Soule, J. A. Foster, and J. Dickinson. Code growth in genetic programming. In *Proceedings of the First Annual Conference on Genetic Programming (GECCO '96)*, pages 215–223, Cambridge, MA, USA, 1996. MIT Press.

M. Srinivas and L.M. Patnaik. Adaptive probabilities of crossover and mutation in genetic algorithms. *IEEE Transactions on Systems, Man and Cybernetics*, 24(4):656–667, 1994.

M. Stolpe. To bee or not to bee - comments on discrete optimum design of truss structures using artificial bee colony algorithm. In *Structural and Multidisciplinary Optimization*, volume 44, pages 707–711. 2011.

B. Sudret. Global sensitivity analysis using polynomial chaos expansions. *Reliability Engineering & System Safety*, 93(7), 2008.

A. Syberfeldt, A. Ng, R. I. John, and P. Moore. Multi-objective evolutionary simulation-optimisation of a real-world manufacturing problem. *Robotics and Computer-Integrated Manufacturing*, 25:926–931, 2009.

A. Teller and M. Veloso. Pado: A new learning architecture for object recognition. In *Symbolic Visual Learning*, pages 81–116, Oxford, UK, 1996. Oxford University Press.

K. Thornton. Simple job array how to. 2010. [Online; accessed March 26, 2011].

A. Topchy and W.F. Punch. Faster genetic programming based on local gradient search of numeric leaf values. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, pages 155–162, San Francisco, California, USA, 2001. Morgan Kaufmann.

V. V. Toropov, U. Schramm, A. Sahai, R. D. Jones, and T. Zeguer. Design optimization and stochastic analysis based on the moving least squares method. In J. Herskovits, S. Mazorche, and A. Canelas, editors, *Proceedings of the Sixth World Congress of Structural and Multidisciplinary Optimization*, Rio de Janeiro, Brasil, 2005.

G. Upton and I. Cook. *Understanding statistics*. Oxford University Press, 1996.

T. Van Belle and D. H. Ackley. Uniform subtree mutation. In J. A. Foster, E. Lutton, J. Miller, C. Ryan, and A. G. B. Tettamanzi, editors, *Proceedings of the 5th European Conference on Genetic Programming (EuroGP 2002)*, volume 2278 of *LNCS*, Kinsale, Ireland, 2002. Springer-Verlag.

V. V. Vasiliev. Anisogrid lattice structures. *Survey of Development and applications, Composite structures*, (54):361–370, 2001.

V. V. Vasiliev. Anisogrid composite lattice structures. *Development and aerospace applications, Composite structures*, (94):1117–1127, 2012.

S. J. Vaughan-Nichols. New trends revive supercomputing industry. *Computer*, 37:10–13, 2004.

F.A.C. Viana and R.T. Haftka. Cross validation can estimate how well prediction variance correlates with error. *AIAA Journal*, 47(9):2266–2270, 2009.

E. Vladislavleva. *Model-based Problem Solving through Symbolic Regression via Pareto Genetic Programming*. PhD thesis, Tilburg University, Tilburg, the Netherlands, 2008.

E. Vladislavleva, G. Smits, and D. Den Hertog. On the importance of data balancing for symbolic regression. *IEEE Transanctions on Evolutionary Computation*, 14(2):252–277, 2010.

S. Wagner and M. Affenzeller. HeuristicLab. A paradigm-independent and extensible environment for heuristic optimization. `http://dev.heuristiclab.com/trac/hl/core`, 2002. [Online; accessed August 16, 2011].

A. R. Wallace. On the tendency of varieties to depart indefinitely from the original type. *Proceedings of the Linnean Society London*, 3:53–62, 1858.

P. A. Whigham. Grammatically-based genetic programming. In J.P. Rosca, editor, *Proceedings of the workshop on Genetic Programming: from theory to real-world applications*, pages 33–41, 1995.

D. Whitley, M. Richards, R. Beveridge, and A. da Motta Salles Barreto. Alternative evolutionary algorithms for evolving programs: evolution strategies and steady state GP. In *Proceedings of the 8th annual conference on genetic and evolutionary computation (GECCO 2006)*, volume 1, pages 919–926, Seattle, Washington, USA, 2006. ACM Press, NY, USA.

B. Widrow and M. A. Lehr. 30 years of adaptive neural networks: perceptron, madaline, and backpropagation. *Proceddings of the IEEE*, 78(9):1415–1442, 1990.

S. Winkler, H. Efendic, L. Del Re, M. Affenzeller, and S. Wagner. Online modelling based on genetic programming. *International Journal of Intelligent Systems Technologies and Applications*, 2(2/3):255–270, 2007.

M. S. Withall, C. J. Hinde, and R. G. Stone. An improved representation for evolving programs. *Genetic Programming and Evolvable Machines*, 10(1):37–70, 2009.

L. Xia. Storage and release of soluble hexavalent chromium from chromate conversion coatings. *Journal of The Electrochemical Society*, 147:2556–2562, 2000.

H. Xie, M. Zhang, and P. Andreae. Population clustering in genetic programming. In P. Collet, M. Tomassini, M. Ebner, S. Gustafson, and A. Ekárt, editors, *Genetic Programming - Lecture Notes in Computer Science*, volume 3905, pages 190–201. Springer-Verlag Berlin Heidelberg, 2006.

Z. Yin, A. Brabazon, C. O'Sullivan, and M. O'Neill. Genetic programming for dynamic environments. In *Proceedings of the International Multiconference on Computer Science and Information Technology*, pages 437–446, 2007.

T. Zeguer and S. Bates. Signpost the future: Simultaneous robust and design optimization of a knee bolster. Technical Report Altair Product Design technical report, Altair Engineering, 2011.

B.-T. Zhang and H. Mühlenbein. Balancing accuracy and parsimony in genetic programming. *Evolutionary Computation*, 3(1), 1995.

J. Ziegler and W. Banzhaf. Decreasing the number of evaluations in evolutionary algorithms by using a meta-model of the fitness function. In C. Ryan, T. Soule, M. Keijzer, E. Tsang, R. Poli, and E. Costa, editors, *Genetic Programming*, volume 2610 of *Lecture Notes in Computer Science*, pages 141–161. Springer Berlin / Heidelberg, 2003.

E. Zitzler and L. Thiele. Multiobjective evolutionary algorithms: A comparative case study and the strength Pareto approach. *IEEE Transactions on Evolutionary Computation*, 3(4): 257–271, 1999.

E. Zitzler, M. Laumanns, and L. Thiele. SPEA2: Improving the strength Pareto evolutionary algorithm. Technical report 103, Computer engineering and networks laboratory (TIK), ETH Zurich, Switzerland, 2001.

D. Zongker, B. Punch, and B. Rand. *lil-gp 1.01 User's manual*. Michigan State University, 1996.