THE UNIVERSITY OF SHEFFIELD

DOCTORAL THESIS

Nonlinear Dynamic System Identification and Model Predictive Control Using Genetic Programming

---

Author:
Tiantian DOU

Supervisor:
Dr. Peter ROCKETT

A thesis submitted in fulfillment of the requirements
for the degree of Doctor of Philosophy
in the

RISE Group
Department of Electronic and Electrical Engineering

September 29, 2019

List of Publications

Conference Paper

Dou, T. and Rockett, P.I. (2017). “Semantic-based Local Search in Multiobjective Genetic Programming”. In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion.* GECCO ’17. Berlin, Germany: ACM, pp. 225–226. ISBN: 978-1-4503-4939-0. DOI: 10.1145/3067695.3076015. URL: http://doi.acm.org/10.1145/3067695.3076015.

Journal Paper

Dou, T. and Rockett, P.I. (2018). “Comparison of semantic-based local search methods for multiobjective genetic programming”. In: *Genetic Programming and Evolvable Machines.*, In press. DOI: 10.1007/s10710-018-9325-4.

Under Review

Dou, T., Kaszubowski Lopes, Y., Rockett, P.I. (2018). "GPML: An XML-based Standard for the Interchange of Genetic Programming Trees". Technical Report. Submitted to *Genetic Programming & Evolvable Machines*.

Dou, T., Kaszubowski Lopes, Y., Rockett, P.I., Hathway, E.A. and Saber, E. (2019). “Model Predictive Control of Buildings Using Genetic Programming Dynamic Models”. Submitted to *Applied Soft Computing*.

THE UNIVERSITY OF SHEFFIELD

Abstract

Electronic and Electrical Engineering

Department of Electronic and Electrical Engineering

Doctor of Philosophy

Nonlinear Dynamic System Identification and Model Predictive Control Using Genetic Programming

by Tiantian DOU

During the last century, a lot of developments have been made in research of complex nonlinear process control. As a powerful control methodology, model predictive control (MPC) has been extensively applied to chemical industrial applications. Core to MPC is a predictive model of the dynamics of the system being controlled. Most practical systems exhibit complex nonlinear dynamics, which imposes big challenges in system modelling. Being able to automatically evolve both model structure and numeric parameters, Genetic Programming (GP) shows great potential in identifying nonlinear dynamic systems. This thesis is devoted to GP based system identification and model-based control of nonlinear systems.

To improve the generalization ability of GP models, a series of experiments that use semantic-based local search within a multiobjective GP framework are reported. The influence of various ways of selecting target subtrees for local search as well as different methods for performing that search were investigated; a comparison with the Random Desired Operator (RDO) of Pawlak et al. was made by statistical hypothesis testing. Compared with the corresponding baseline GP algorithms, models produced by a standard steady state or generational GP followed by a carefully-designed single-objective GP implementing semantic-based local search are statistically more accurate and with smaller (or equal) tree size, compared with the RDO-based GP algorithms.

Considering the practical application, how to correctly and efficiently apply an evolved GP model to other larger systems is a critical research concern. Currently, the replication of

GP models is normally done by repeating other's work given the necessary algorithm parameters. However, due to the empirical and stochastic nature of GP, it is difficult to completely reproduce research findings. An XML-based standard file format, named Genetic Programming Markup Language (GPML), is proposed for the interchange of GP trees. A formal definition of this standard and details of implementation are described. GPML provides convenience and modularity for further applications based on GP models.

The large-scale adoption of MPC in buildings is not economically viable due to the time and cost involved in designing and adjusting predictive models by expert control engineers. A GP-based control framework is proposed for automatically evolving dynamic nonlinear models for the MPC of buildings. An open-loop system identification was conducted using the data generated by a building simulator, and the obtained GP model was then employed to construct the predictive model for the MPC. The experimental result shows GP is able to produce models that allow the MPC of building to achieve the desired temperature band in a single zone space.

Acknowledgements

First and foremost, I would like to express my sincere gratitude to my supervisor, Peter Rockett, who gave me the opportunity to pursue my passion in Genetic Programming and provided me with strong support through my studies. His patience and encouragement during my PhD made this journey possible.

Very special thanks to my parents who offered me unconditional love, support, and understanding during the period of my study. I am very grateful for their sacrifices and trust that encourage me in achieving my goals. I would also like to sincerely thank my boyfriend for selflessly providing me with emotional support.

I would like to thank all of my research fellows in the RISE Group, for their help, kindness, interesting discussions and scientific cooperation for many years. Particularly, I would like to thank Abigail Hathway for providing professional advice, Yuri Kaszubowski Lopes who contributed a lot to our collaborative work making possible conclusions presented here, and Esmail Saber for all the interesting conversations and very useful comments on my scientific research.

This work was partly supported by the UK Engineering and Physical Sciences Research Council (EPSRC) grant EP/N022351/1.

Contents

List of Publications **iii**

Abstract **v**

Acknowledgements **vii**

1 Introduction **1**

1.1 Motivation 1

1.2 Contributions 5

1.3 Thesis Structure 5

2 Background: Genetic Programming **7**

2.1 Research Issue of Model Complexity Control in System Identification Using GP 9

2.1.1 Model Complexity Measurements 10

2.1.2 Model Accuracy Measurements 11

2.1.3 Two Important Issues Affecting Generalization Ablity of GP Models . 12

Bloat in GP and Associated Approaches Used for Tackling This Issue 12

Overfitting in GP and Associated Approaches Used for Tackling This Issue 14

2.1.4 Multi-Objective Optimization in GP 16

2.2 Details of GP 18

2.2.1 GP Evolution Process 18

2.2.2 Pareto Based Multi-Objective Optimization and Model Selection . . . 22

2.3 Other Comparable System Identification Techniques 22

2.3.1 Artificial Neural Networks for System Identification and Predictive Control 23

2.3.2 Volterra-Series Models for System Identification and Predictive Control 26
2.3.3 Hammerstein Model for System Identification and Predictive Control 28
2.3.4 Gaussian Process Models for System Identification and Predictive Control . . . 30
2.3.5 Summary . . . 32

3 Symbolic Regression System Identification Using Genetic Programming 35
3.1 Author's Contribution . . . 35
3.2 Introduction . . . 36
3.3 Related Work . . . 37
3.3.1 Semantically-Aware Methods in GP . . . 37
3.3.2 Local Search in GP . . . 39
3.4 Experimental Methodology . . . 41
3.4.1 Evolutionary Framework . . . 41
3.4.2 Local Search Methods . . . 43
3.4.3 Subtree Selection . . . 44
3.4.4 Algorithm Comparisons . . . 44
3.4.5 Subtree Generation and Replacement . . . 45
3.4.6 Test Functions . . . 47
3.4.7 Statistical Testing . . . 47
3.5 Results & Discussion . . . 48
3.5.1 Comparison of Generational and Steady-state Global Strategies Without Local Search . . . 49
3.5.2 Influence of the Global Search Strategy on the Efficacy of a Given Local Search Method . . . 55
3.5.3 Comparing RDO in Generational and Steady-state Global Strategies . 55
3.5.4 The Role of the Generational and Steady-state Strategies for Local Search . . . 56
3.5.5 Influence of the Number of Cycles of Local Search . . . 57
3.5.6 Influence of Local Search on Expected Tree Sizes . . . 61
3.5.7 Performance of Random Subtree Generation as a Local Search Operator . . . 62

3.5.8 The Performance of RDO as a Local Search Operator 62
3.5.9 RDO Compared to Global GP + GP Local Search 63
3.5.10 Computational Complexity Resulting from Different Local Search Strategies 63
3.6 Discussion and Future Work 64
3.7 Conclusions 66

4 GPML **69**
4.1 Author's Contribution 69
4.2 Introduction 69
4.3 Extensible Markup Language (XML) 71
4.4 GPML Specification 73
4.5 Implementation 78
4.5.1 Writing GPML 78
4.5.2 Reading GPML 79
4.5.3 Validating GPML 79
4.6 Conclusions 80

5 Model Predictive Control of Nonlinear Dynamic System Using GP **81**
5.1 Author's Contribution 81
5.2 Introduction 82
5.3 Genetic Programming for Dynamic System Identification 85
5.3.1 Identification of the Eaton-Rawlings Reactor 87
5.4 Building Control Methodology 89
5.4.1 Building Simulator – EnergyPlus 90
5.4.2 Test Building Description 92
5.4.3 Data Acquisition 92
5.4.4 Input Selection 94
5.4.5 Genetic Programming for Building Identification 95
5.4.6 MPC Test Framework 98
5.5 Results 99
5.5.1 Model Performance 99
5.5.2 MPC Performance 101

5.6 Discussion and Future Work 102
5.7 Conclusions 106

6 Conclusions **107**

7 Future Work **111**

7.1 Local Search in GP 111
7.2 Building MPC Based on GP Models 111

A The Best Model Presented in GPML Form **113**

B GPML Schema **119**

References **127**

List of Figures

2.1 Simple example GP tree. 19
2.2 Example of subtree crossover. 21
2.3 Example of subtree mutation. 21
2.4 A general model of a neuron of ANN. 24
2.5 A simple artificial neural network. 24
2.6 Hammerstein model structure. 28

3.1 Relationship between successful replacement rate with normalized subtree depth over five benchmark functions. 58
3.2 Relationship between MSE reduction with normalized subtree depth over five benchmark functions. 59
3.3 Relationship between test MSE and the number of local search cycles of SSGP-SSGP-3 over five benchmark functions; the number following 'LSCycle' denotes the number of local search cycles in the local search process. . . 60

4.1 Simple example GP tree. 75

5.1 Example GP tree representing a simple dynamic system. 86
5.2 Test residual comparison between the NARMAX model and the best GP model 90
5.3 Overview of the process employed in this work for MPC (contributed by Dr. Yuri Kaszubowski Lopes). 91
5.4 SketchUp representation of the simulated building (contributed by Dr. Esmail Saber). 93
5.5 A PRBS-7 sequence (a) and two different examples of APRBS-7 cycles (b-c) generated over a minimum-to–maximum amplitude range of 0.0 to 0.11 kg/s. 94

5.6 Residuals of the selected GP model over the test dataset. Each plot shows the residuals for a given number of steps ahead – for example, “OSA” = 1-step head, “2SA”= 2-steps ahead, etc. The units of the ordinate axes are Celsius. . 100

5.7 Typical results of MPC controlling a single-zone room for the (test) month of February. The upper plot is the zone temperature, and the lower plot the mass flow rate (MFR) controlled variable. The rectangular upper plot represents the temperature schedule. 103

List of Tables

2.1 Commonly used model accuracy measurements 12

3.1 Evolutionary parameters used in this work 43
3.2 Test functions 48
3.3 Summary of experimental protocols used: Generational global search 51
3.4 Summary of experimental protocols used: Steady-state global search 52
3.5 Ranking of the mean squared test errors (MSEs) by algorithm; algorithms listed in the same column display no statistical difference. Conversely, a statistically significant difference is detected between algorithms in different columns. The gray-shaded cells denote that the algorithms shown to their immediate left column have no statistical difference with the GenGP-RDO-1 algorithm in column 10. The rightmost column shows the actual mean rank values. 53
3.6 Overall ranking of node counts by algorithm; algorithms listed in the same column display no statistical difference. Conversely, a statistically significant difference is detected between algorithms in different columns. The gray-shaded cells denote that algorithms to their immediate left show no statistical difference to the algorithms in that column. The rightmost column shows the actual mean rank values. 54
3.7 CPU runtimes for one cycle of local search on the French curve function . . . 64

5.1 Evolutionary parameters used for the Eaton-Rawlings reactor system identification 88
5.2 Variables used in the system identification model 95
5.3 Evolutionary parameters used in this work 97
5.4 MPC optimisation parameters used in this work. 99

List of Abbreviations

GP	**G**enetic **P**rogramming
MOGP	**M**ulti**O**bjective **G**enetic **P**rogramming
MPC	**M**odel **P**redictive **C**ontrol
NMPC	**N**onlinear **M**odel **P**redictive **C**ontrol
GPML	**G**enetic **P**rogramming **M**arkup **L**anguage
RDO	**R**andom **D**esired **O**perator
AR	**A**utoreg**R**essive
NAR	**N**onlinear **A**utoreg**R**essive
ARX	**A**utoreg**R**essive with e**X**ogenous input
NARX	**N**onlinear **A**utoreg**R**essive with e**X**ogenous input
ARMA	**A**utoreg**R**essive **M**oving **A**verage
ARMAX	**A**utoreg**R**essive **M**oving **A**verage with e**X**ogenous input
NARMAX	**N**onlinear **A**utoreg**R**essive **M**oving **A**verage with e**X**ogenous input
ANN	**A**rtificial **N**eural **N**etwork
NEAT	**N**euro-**E**volution of **A**ugmenting **T**opologies
GA	**G**enetic **A**lgorithm
MDL	**M**inimum **D**escription **L**ength
VC	**V**apnik-**C**hervonenki
MSE	**M**ean **S**quare **E**rror
RMSE	**R**oot **M**ean **S**quare **E**rror
MAE	**M**ean **A**bsolute **E**rror
DWA	**D**ynamic **W**eighted **A**ggregation
VEGA	**V**ector **E**valuated **G**enetic **A**lgorithm
MOGA	**M**ulti**O**bjective **G**enetic **A**lgorithm
NSGA	**N**on-dominated **S**orting **G**enetic **A**lgorithm
NPGA	**N**iched **P**areto **G**enetic **A**lgorithm

EA	**E**volutionary **A**lgorithm
SPEA	**S**trength **P**areto **E**volutionary **A**lgorithm
NSGA-II	**N**on-dominated **S**orting **G**enetic **A**lgorithm-II
PAES	**P**areto **A**rchived **E**volution **S**trategy
RNN	**R**ecurrent **N**eural **N**etwork
CTRNN	**C**ontinuous **T**ime **R**ecurrent **N**eural **N**etwork
VFXLMS	**V**olterra **F**iltered-**X** **L**east **M**ean **S**quare
SISO	**S**ingle **I**nput **S**ingle **O**utput
PDE	**P**artial **D**ifferential **E**quation
DPS	**D**istributed **P**arameter **S**ystem
MMMPC	**M**in-**M**ax **M**odel **P**redictive **C**ontrol
PCE	**P**olynomial **C**haos **E**xpansions
FIR	**F**inite **I**mpulse **R**esponse
IIR	**I**nfinite **I**mpulse **R**esponse
SVM	**S**upport **V**ector **M**achine
AGX	**A**pproximately **G**eometric semantic **crossover**
GSGP	**G**eometric **S**emantic **G**enetic **P**rogramming
SSGP	**S**teady-**S**tate **G**enetic **P**rogramming
GenGP	**Gen**erational **G**enetic **P**rogramming
XML	e**X**tensible **M**arkup **L**anguage
DOM	**D**ocument **O**bject **M**odel
OSA	**O**ne **S**tep **A**head
PRBS	**P**seudo-**R**andom **B**inary **S**equence
APRBS	**A**mplitude modulated **P**seudo-**R**andom **B**inary **S**equence
NLopt	**N**on**L**inear time **opt**imization library
SID	**S**ystem **I**dentification
IDF	**I**nput **D**ata **F**ile
FMI	**F**unctional **M**ock-up **I**nterface
CIBSE	**C**hartered **I**nstitute of **B**uilding **S**ervices **E**ngineers
COBYLA	**C**onstrained **O**ptimization **BY** **L**inear **A**pproximations
MLSL	**M**ulti-**L**evel **S**ingle-**L**inkage

Chapter 1

Introduction

1.1 Motivation

Briefly, a dynamical system refers to a system whose states change over time. Developing mathematical models for dynamical systems by utilizing collected data is known as system identification. It presents the relationships between all the inputs and outputs by determining an explanatory model. Constructing models based on available data is the first and crucial step for further controller design. Practically, system identification requires quick and efficient conversion of measurement data into a model that can capture the underlying dynamics of the systems or processes of interest. In general, dynamical systems can be categorized into linear and nonlinear models. For identification of linear models, the most popular methods are autoregressive moving average with exogenous input (ARMAX) models and its subsets, which are autoregressive (AR), autoregressive moving average (ARMA), and autoregressive with exogenous input (ARX) models (Nelles, 2001). However, most practical systems in the real world are nonlinear. Therefore, linear techniques are no longer sufficient to model the nonlinearities of complex systems in the real world. In this thesis, only research on nonlinear dynamic systems is considered. The discussion of techniques and applications to linear systems is out of scope. In nonlinear system modelling, many statistical models (i.e., nonlinear ARMAX, nonlinear ARX, nonlinear ARMA) have been adopted for approximating nonlinear systems. One drawback of these statistical methods is that appropriate model structures need to be identified in advance based on prior information, which is a difficult task in practice. Being able to evolve both model structure and numerical coefficients simultaneously, Genetic Programming (GP) has been receiving a considerable amount of attention for nonlinear system studies, both as a system identification tool and further as a model-based controller.

In system identification, the trade-off between model accuracy and model complexity is important. Based on a finite number of training samples, it is possible to produce undesired models which try to 'memorize' the information extracted from a particular training dataset rather than learn the features from it. Those models are excessively complex due to the inclusion of too many unnecessary input variables or parameters. In the literature, the overly complex models are often referred to as overfitted models. Consequently, they exhibit high model accuracy on a specific training dataset but bad generalization results on new, unseen data samples. Additionally, there is a commonly observed phenomenon in GP that the size of solutions grows dramatically with increasing generations without any improvement in model fitness. Models with an unnecessarily large size impose a heavy computation burden and are of little practical use. To deal with those issues, various model complexity control methods are applied, such as the regularization framework, which includes a penalty term in the objective function to restrict the growth of tree size. Such an integrated objective function needs proper weights on different elements to tune the balance between model accuracy and complexity. However, the determination of these weights is a difficult problem in practice. Pareto-based multi-objective optimization scheme optimizes multiple objectives separately and simultaneously, which avoids the difficulty of determining weights for different objectives. This advantage makes it a promising tool for dealing with complex system identification since most real-world processes have multiple and conflicting objectives to be optimized.

GP has shown great potential in empirical modelling of complex processes in the real world (Poli, Langdon, and McPhee, 2008). Nonetheless, conventional GP modifies trees at the syntactic level. A slight change in syntax can dramatically change the fitness of a program, which has a negative effect on GP search efficiency. Methods that take the semantics into account have been witnessed to show great potential in improving the search efficiency (Beadle, 2009; Jackson, 2010; Beadle and Johnson, 2009). Broadly, memetic algorithms which enhance population-based global search with a heuristic local search have attained distinguished performance (Neri, Cotta, and Moscato, 2012). The integration of semantic-based local search into GP has been demonstrated to be able to produce more accurate models by improving the search power (Iba, Garis, and Sato, 1994; Topchy and Punch, 2001). Though the hybridization of local search in GP boosts the model accuracy in many applications (Beadle, 2009; Jackson, 2010; Beadle and Johnson, 2009), it does not come for

free. The computational effort increases inevitably due to the additional exploitation process of local search. That is the main reason that, in practice, few applications use such a hybrid method. The trade-off between model complexity and accuracy caused by local search remains a critical research issue in GP. Typically, most local search methods in GP concentrate on fine-tuning node functionality. Approaches that change the tree morphologies by local search are comparatively little explored. In this thesis, a semantic-aware local search method that optimizes GP trees by modifying program morphologies is proposed. In addition, a size restriction strategy in local search was designed to prevent the growth of the GP parent tree for the consideration of practical use. The motivation to tree size restriction is due to the fact that the rapid growth of GP individuals is a feature of many semantically aware methods (Vanneschi, Castelli, and Silva, 2014).

When a GP model is built and used for further applications, accurate model replication is an essential requirement. A wide range of researches in evolutionary computing is, perforce, both empirical and stochastic, which imposes a particular difficulty in model replication. Replicability is critical in scientific research since it allows the research findings to be verified independently by others. Otherwise, the research irreproducibility may provide a hotbed for the rapid spread of erroneous results and thus hinder the effective progress of research. Traditionally, a universal way to reproduce GP models is to clarify algorithm parameters, such as population size, crossover and mutation rates, etc. However, the given information is often incomplete and inadequate to generate an accurate replication of the experiments being published. GP evolution is a stochastic process, which is influenced by many detailed settings, like seed values, the design of the random number generator, and so on. Theoretically, the stochastic characteristics can be averaged out by a large number of repetitions of experiments, but the generality of this claim is far from clear. Therefore, for achieving correct replication, the result needs to be carefully addressed and a standardized format for the interchange of GP trees is required. A standard interchange format in GP can provide many benefits. A clear representation of GP trees instead of a trivial description of how models are generated would speed research pace in this field. It allows a direct comparison of different research findings in the research community, which avoids the difficulties of reproducing others' results, often with inadequate information. Besides, it would offer a major convenience for large systems to use evolved GP models just like a 'plug-in' component.

Model predictive control (MPC) (Camacho and Bordons, 2004) is a potent control methodology targetted to systems in which a significant delay exists between the current inputs and any visible response. The core of MPC is a trained model which approximates the features of the system being studied. Given a prediction horizon extending many discrete time steps into the future, a model predictive controller calculates the optimal control strategy over a certain control horizon by optimizing some objective function. Specifically, at each step, the future input sequence is optimized, and only the first value of the sequence adopted by the controller. The entire process is repeated at the next time step. Even though MPC has been frequently used in chemical industrial control engineering, few pieces of research have been done on building applications (Rockett and Hathway, 2017). Compared to traditional rule-based methods, MPC in buildings can provide noticeable energy savings perhaps up to 25% (Rockett and Hathway, 2017), with less CO_2 emissions and improved internal environmental conditions. Nevertheless, the generation and calibration of the predictive model, central to MPC, can consume 70% of total costs for MPC implementation (Henze, 2013). The conventional models applied in MPC are hand-tuned by highly skilled control experts through trial and error. Such high-cost approaches are not sufficient to achieve an economical MPC of buildings (Rockett and Hathway, 2017). Under this circumstance, recently developed machine learning-based techniques that produce models based on data collected from real systems appears as a promising tool in building environment control. It is widely acknowledged that buildings exhibit nonlinear characteristics, which imposes difficulties in deciding model structures for traditional methods. Additionally, due to building deterioration, internal alterations or external changes, the characteristics of buildings vary over time. This also increases the difficulty of generating appropriate models for buildings. Taking advantage of being able to automatically optimize both model structure and appropriate numeric coefficients during evolution, GP shows great potential in modelling nonlinear dynamics of building systems.

In this thesis, we intend to investigate the ability of GP to identify nonlinear dynamic systems and assess the control performance of GP models for buildings under a MPC scheme. The rest of this chapter is organized as follows: a short introduction and motivating description of this work is described first. Afterward, contributions made in this thesis are outlined in the next section. Last, the structure of the thesis is set out.

1.2 Contributions

The main intention of this thesis is to investigate the performance of GP in nonlinear dynamic system identificaton and MPC applications. As a start, a brief introduction to GP is given. Some key research issues, such as model complexity control and model selection, encountered in GP for solving system identification problems are reviewed.

Detailed research on the performance of GP algorithms when supplemented by semantically-aware local search methods is discussed. In particular, this thesis extends consideration of the effectiveness of local search to a multiobjective objective optimization framework since balancing two critical but explicitly conflicting objectives, namely goodness-of-fit and model complexity, is a key requirement in the empirical modeling of data (Cherkassky and Mulier, 2007; Le et al., 2016). Accordingly, a comprehensive comparison between various combinatory GP algorithms including the Random Desired Operator (RDO) approach is presented.

The establishment of a standardised format for the interchange of GP trees would be of great benefit in improving research efficiency and preventing the unnecessary time waste for researchers to reproduce others' experiments with limited often incomplete information. This thesis describes a standardised format for the interchange of GP trees, named Genetic Programming Markup Language (GPML), based on XML, to allow evolved GP solutions be used just like a 'plug-in' component in larger systems. The instructions for GPML reading, and validation are explained in detail.

In a control scenario, a GP approach was employed to construct the predictive model for MPC. The experimental results shows that GP is able to automate this control process using an open-loop excitation experiment. The resulting MPC simulation is able to maintain the internal temperature of a single-zone test building to within ± 1 C of the desired setpoint. In this work, one member of our research group, Yuri Kaszubowski Lopes, contributed to the data generation of a simulated building for the following open-loop system identification and provided the MPC results.

1.3 Thesis Structure

This thesis is partitioned into seven chapters, which are organized as follows:

1. Chapter 1 sets the stage for the thesis. The motivation for work done so far is described. Additionally, the main contributions of the thesis are summarized.

2. Chapter 2 presents the main issues concerned in the research of GP-based system identification and gives an informative introduction to GP. Several comparable system identification techiniques are also discussed.

3. Chapter 3 introduces a hybridation algorithm that uses semantic-based local search within a multiobjective genetic programming (MOGP) framework for improving the search power of GP in a semantic level.

4. Chapter 4 proposes an XML-based standardised format, namely GPML, for the interchange of GP trees. Details of how to read, write and validation a GPML file are also described.

5. Chapter 5 demonstrates a novel approach to obtaining dynamic nonlinear models using GP for the MPC of buildings.

6. Chapter 6 is devoted to concluding the main contributions of the thesis.

7. Chapter 7 provides suggestions on future work and extensions.

Chapter 2

Background: Genetic Programming

The modelling of real-world processes is a challenging task due to the difficulties of finding both the model structures and appropriate parameters of a model at the same time.

There is no one algorithm can be utilized as a universal optimal solution, because different systems have various complex behavior. In addition, the choice of modelling technique also depends on the identification purpose (e.g. For prediction, a desired model is expected to have small prediction error, usually with no other particular requirement on the model structure and parameters. If a model is built for control, the approximation is expected to be able to capture the true dynamics of the system studied, and of appropriate orders, since a high-order model may increase difficulties in the process of control design (Đukić and Sarić, 2012).). In general, methods used for empirical modelling can be classified in two groups: phenomenological or behavioral (Metenidis, Witczak, and Korbicz, 2004).

Developing a phenomenological model is a process of theoretical derivations, which requires prior knowledge of the systems under investigation. In the literature of system identification, the phenomenological models are interchangeably referred to as white-box models (Nelles, 2001). If the rationale governing a system has already been well understood then it is more likely to select proper model structures and yield good results for specific problems. However, in practice, the principles of dynamical processes are not always adequately understood in advance. In this situation, the selection of proper models becomes a critical problem because it has direct effects on the accuracy of system identification. To deal with this issue, behavioral modelling is commonly employed.

A behavioral model is derived by approximating the relationships of inputs and outputs from experimental data without a need of prior knowledge of the system mechanics. In empirical modelling, models developed solely based on measurement data are also called

black-box models (Nelles, 2001). Traditional statistical regression methods can be used for developing behavioral models. However, they can only be used under certain constraints. Specifically, traditional statistical regression techniques approximate the dynamics of systems by linear or nonlinear models with pre-specified structures, which are often hard to determine. Additionally, when traditional statistical regression techniques are applied, the residuals are often assumed to have a normal distribution. In recent decades, another family of algorithms has attracted an increasing number of researchers. These algorithms, called modern heuristic techniques, draw inspiration from strategies and phenomena existing in the natural world. Among them, the widely studied techniques include artificial neural networks (ANNs), and evolutionary algorithms. ANNs are inspired by biological neural networks (Haykin, 1994), and have been successfully used in various structural engineering problems (Guzelbey, Cevik, and Gögüş, 2006; Guzelbey, Cevik, and Erklig, 2006; Gandomi and Alavi, 2011; Guven, 2011; Pala, 2006). Despite ANNs exhibiting their potential in a wide range of engineering applications, they have a drawback that no explicit mathematical equations are developed to describe the relationships between inputs and outputs of systems. Besides, the structure of ANNs, such as the number of hidden layers and transfer functions, needs to be pre-defined. Stanley and Miikkulainen (Stanley and Miikkulainen, 2002) presented the Neuro-Evolution of Augmenting Topologies (NEAT) method which can tune both model structure and numerical parameters of neural network models by a GA during the learning process. Overall, complex engineering system identification calls for more robust computing techniques.

Being able to automatically optimize both structures and numerical coefficients of a model simultaneously, GP remains a highly-researched technique in empirical modelling. Inspired by the principle of Darwinian natural selection, GP produces solutions by emulating the biological evolution of living organisms (Koza, 1992). The idea of gradually improving candidate models guided by a learning procedure was first discussed by Friedberg (Friedberg, 1958). Genetic algorithms (GA) were then developed by John Holland (Holland, 1992), who established a theoretical foundation for computational evolution. Holland's GA schema theory simulated the biological evolution by introducing the processes of crossover, recombination, mutation to find better solutions to problems. However, it was criticized for having difficulty in predicting the behaviour of a GA over multiple generations (Poli, 2000). Cramer (Cramer, 1985) used a GA to develop a sequential programming language so as to

evolve tree-structured simple computer functions from low-level computational primitives. A breakthrough in GP was made through the work of Koza (Koza, 1992) on symbolic regression. The difference between GA and GP is the representation of solutions. Candidates in GA are classically encoded into bit strings (sequences of 1's and 0's), which are often used for parameter optimization. Real numbers and other objects are also accepted by modern computers to form a GA chromosome. GP solutions, on the other hand, are suitable for evolving both model structure and coefficients, benefitting from the hierarchical, tree-structured computer programs with variable length. The structures and corresponding parameters of a tree change during the search process, which makes it a promising alternative for developing models from an input-output dataset. As an extension of GA, most of the genetic operators in GP, such as crossover and mutation, can be implemented with little change.

2.1 Research Issue of Model Complexity Control in System Identification Using GP

Although GP has had a variety of applications in solving nonlinear dynamic system identification and model-based control problems (Poli, Langdon, and McPhee, 2008), there are some research issues, such as how to improve efficiency of GP, how to prevent premature convergence, and how to improve the generalization ability of the trained solutions (Dabhi and Chaudhary, 2015), that still needs to be carefully dealt with when GP is applied in practice. In order to address these problems, a lot of effort has been made by researchers (Dabhi and Chaudhary, 2015). Since this thesis concentrates on nonlinear dynamic system identification and model-based control using GP, the problem of model complexity control is a key question that we are concerned about.

Practically, a desired model is expected to be accurate and compact. Especially in real-world control applications, model reduction is often conducted for approximating complex high-order models by producing adequate low-order models to facilitate both computationally efficiency and controller design (Besselink et al., 2013). If a trained model is too simple, it would not be able to approximate a system to a reasonable degree of accuracy. On the other hand, if a learned model is overly complex and captures not only the features of the process of interest but also the noise contained in a limited training set, high generalization error may be obtained when the model is evaluated on other new, unseen datasets. Further, an efficient

controller is less likely to be achieved based on an over-parameterized model, which is also called an overfitted model. Thus, there is an obvious trade-off between model complexity and model accuracy. In this section, the research issue about model complexity control is discussed and its related techniques are reviewed.

2.1.1 Model Complexity Measurements

Methods used for measuring model complexity can be generally classified into three types: structural complexity measurement, functional complexity measurement, and statistical machine learning complexity measurement (Le et al., 2016).

Early criteria used to estimate the complexity of GP trees are simply by observing the model structure. To be specific, the structural complexity of a GP tree can be measured by the number of nodes in a tree, the number of levels (layers) in a tree and the total number of nodes in all subtrees of a GP tree. These techniques were proven to be useful in controlling tree growth (Le et al., 2016), but they have also been criticized for being deceptive. A GP tree with a large size sometimes can be simplified to a small tree with fewer nodes. Considering two GP trees: $cos((4.1+0.9-3.0)x)$ and $cos(10x)$, the former one appears twice as complex as the latter from the perspective of node count. Thus, the evolution process would favour the second tree for breeding the next generation. However, from the perspective of semantics, the second function is more oscillatory, and therefore of higher probability to exhibit excessive variability, which tends to the generation of overfitted models. Additionally, utilizing the node count as the measurement assumes each node make an equal contribution to model complexity. Without considering the associated functions of nodes, the complexity introduced by an exponential node, though embedding an infinite power series, is the same with a unary minus node. This deficiency has motivated people to consider other complexity measures. Rissanen (Rissanen, 1978) proposed the Minimum Description Length (MDL) to measure the Kolmogorov complexity of a model. The above examples can be simplified to $cos(2x)$ and $cos(10x)$ respectively, and model complexity of them is considered the same based on the MDL measurement. Consequently, the former model exhibiting better functional smoothness would be preferred during the evolution process (Ni and Rockett, 2015).

Functional complexity measurements were initially proposed for preventing the generation of overfitted models since some researchers suspect that there is an association between the evolution of overfitted models with the functional complexity of GP trees. This type

of measurement estimates the complexity of a model by analyzing its output over possible input space (Le et al., 2016). To take a simple example, theoretically, any GP tree can be approximated by a Chebyshev polynomial. Vladislavleva et al. (Vladislavleva, Smits, and Hertog, 2009) used the order of the Chebyshev polynomial as the model complexity, named "order of nonlinearity", of a GP model. Overfitted models exhibiting large oscillation feature are usually approximated by high-order Chebyshev polynomials (Stinstra, Rennen, and Teeuwen, 2008). This type of complexity measurement provides a new perspective on calculating model complexity, however, it is criticized for lacking reliable theoretical basis (Le et al., 2016).

The research on employing statistical learning theory for measuring model complexity is still at a preliminary stage. To take an instance, Montana et al. (Montaña et al., 2011) utilized a new criterion called Vapnik-Chervonenki (VC) dimension (Vapnik, 1998) to measure the model complexity of GP trees. The experimental result shows the VC-based algorithm exhibited promising performance on model complexity control during evolution. The study of injecting statistical learning techniques into the measurement of model complexity shed light on a new trend for further research on model selection algorithms in GP. However, one notable limitation of this type of measurement is that it requires complex computation (Le et al., 2016).

Therefore, counting the number of nodes is the simplest and most common complexity measure used to date in the GP literature to represent the complexity of a model, and this measurement is adopted in this thesis.

2.1.2 Model Accuracy Measurements

Model accuracy is also termed as "goodness of fit" which indicates how well a model fits a set of observations. In system identification, there are several commonly used model accuracy measurements, such as the mean square error (MSE), root mean square error (RMSE), mean absolute error (MAE) and R squared (R^2). The formulas for these criteria are described in Table 2.1. In empirical modelling, goodness-of-fit is normally MSE. Thus, in this thesis, we use MSE to evaluate model accuracy of GP trees.

TABLE 2.1: Commonly used model accuracy measurements

Criteria	Formulas
Mean square error (MSE)	$= \frac{1}{N}\sum_{i=1}^{N}(x_i - \hat{x}_i)^2$
Root mean square error (RMSE)	$= \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \hat{x}_i)^2}$
Mean absolute error (MAE)	$= \frac{1}{N}\sum_{i=1}^{N}\lvert x_i - \hat{x}_i\rvert$
R squared (R^2)	$= 1 - \frac{\sum_{i=1}^{N}(x_i - \hat{x}_i)^2}{\sum_{i=1}^{N}(x_i - \bar{x}_i)^2}$

2.1.3 Two Important Issues Affecting Generalization Ablity of GP Models

In machine learning problems, the goal is to produce a unique solution based on a finite number of training data (Nelles, 2001). Thus, any approximating function is inevitably inaccurate due to the limited training information. In GP, the generalization ability of models is affected by two critical research issues: bloat and overfitting (Dabhi and Chaudhary, 2015).

Bloat in GP and Associated Approaches Used for Tackling This Issue

In practice, it is often noted that the average size (number of nodes) of GP trees grows very rapidly after a certain number of generations, without any corresponding increase in fitness. This phenomenon of trees growing at a rapid pace with no improved fitness is known as bloat. Researchers have been puzzled by the origin of bloat for over a decade. There is no consensus on why it occurs during the evolution process. Five theories have been proposed to explain the reasons for the bloat phenomenon, which are: Replication Accuracy Theory (Mcphee and Miller, 1995), Removal Bias Theory (Soule and Foster, 1998), Modification Point Depth Theory (Luke, 2003), Program Search Space Theory (Langdon and Poli, 1998) and Crossover Bias Theory (Poli, Langdon, and Dignum, 2007).

Bloat hinders the search efficiency of GP in practical applications because the redundancy of a tree produces a model of 'complex' form and inevitably raises the burden of computation. A program with bloat means it is more complex than it needs to be. Bloat impairs the model comprehensibility as the programs are too complex to uncover the nature of systems intuitively. Application of bloat control gives several obvious benefits, such as encouraging parsimony, lessening exorbitant computational resources, and promoting the search efficiency.

Different approaches have been applied for controlling bloat in GP. In 1992, Koza (Koza, 1992) proposed to use code editing or an expression simplification method to control the size of programs during evolution by removing redundant code. However, this strategy was criticized for leading to premature convergence (Haynes, 1998). Simplifying trees by specifying the size or depth limits of produced child solutions was also studied by the GP community (Koza, 1992). This approach applies a validation test to child models to verify if the size of them exceeds the depth limit once they are created. If a new offspring is bigger than any of the pre-specified limits, it is ignored and the best of the selected parent trees return. Crawford-Marks and Spector (Crawford-Marks and Spector, 2002) designed a new genetic operator, named size-fair crossover, to combat the bloat. The size-fair crossover operator exchanges subtrees from their parent solutions under a certain restriction to control tree growth. They suggested that parsimonious trees can be generated by using the size-fair genetic operator with no extra computation. Poli (Poli, 2003) described an selection mechanism, called the Tarpeian technique, to address the bloat problem. In this work, a fixed portion of the population is assigned low fitness values if their tree size is larger than the average. GP programs with low fitness values have lower probabilities of being selected as parents to breed child solutions. The Tarpeian technique minimizes the number of evaluations required and does not have to specify the size of the potential solutions in advance. However, when the portion is set large, it becomes excessively aggressive by rejecting large programs regardless of how fit they are. A commonly used approach to dealing with bloat during the evolution is parsimony pressure (Poli and McPhee, 2008), which is the simplest method to address bloat problems. This technique penalizes the fitness of a solution according to its complexity. A new fitness estimation formulation of program k is defined

$$f_p(k) = f(k) + c * \ell(k), \tag{2.1}$$

where $f(k)$ denotes the original fitness evaluation of a solution, $\ell(k)$ is the size of the solution, and c is the parsimony coefficient. Obviously, the parsimony coefficient controls the trade-off between model accuracy and complexity. Though difficult, the determination of the parsimony coefficient is crucial as it directly influences the intensity of bloat control. Poli and McPhee (Poli and McPhee, 2008) introduced a co-variant parsimony pressure strategy, according to which the parsimony coefficient is assigned dynamically during evolution. The

experimental result over symbolic regression problems shows the co-variant parsimony pressure approach achieves complete control over the tree bloat. Dignum and Poli (Dignum and Poli, 2008) showed the potential of using an operator equalization approach to control bloat. In this approach, the search process is guided towards finding smaller or larger programs. It controls the distribution of tree sizes during an evolutionary run by probabilistically accepting every generated offspring candidate based on its size. They concluded that the operator equalization technique is able to control the solution length distribution effectively by saving efforts on exploring larger model spaces.

Overfitting in GP and Associated Approaches Used for Tackling This Issue

A model which is developed based on finite training data carries the risk of over-adaption. In machine learning, the phenomenon of generating an over-parameterized model that can fit a specific training dataset very well but fails to fit unseen data is known as overfitting. The essence of overfitting is that the approximating functions are not only learning the true dynamics of systems under study, but also unknowingly extracting extra residual variations (i.e. the noise), and taking them as the true features of the system by mistake. The overfitted models tend to 'memorize' a specific training dataset (including noise) rather learn the characteristics it contained. Therefore, the overfitted solutions exhibit low approximation errors on the training dataset but high generalization errors on an unseen dataset.

A wide range of methods has been used by GP researchers to overcome the problem of overfitting. Kotanchek et al. (Kotanchek, Smits, and Vladislavleva, 2008) prevented the occurrence of overfitting in the evolution by using interval arithmetic. Similarly, Stinstra et al. (Stinstra, Rennen, and Teeuwen, 2008) applied interval arithmetic in Pareto simulated annealing based symbolic regression to ensure the development of robust models. A commonly used method for avoiding the selection of overfitted models is partitioning the dataset. An available dataset can be divided into two disjoint data sets, training data, and test data. The training data are used for developing models during evolution, and the test dataset is employed to evaluate the generalization performance of the evolved solutions (Zavoianu, 2010). In this way, the overfitted programs can be easily identified by producing smaller approximation errors on the training data but large errors on the test data. N-fold cross-validation is also a widely used technique for preventing the selection of overfitted models (Zavoianu,

2010). In N-fold cross-validation approach, the finite data are divided into N disjoint subsets. Models are trained N times and each time N-1 subsets used for training, the remaining subset used for the test. However, this N-fold cross-validation is criticized as unsuitable for GP, since GP may produce a different model in each training process (Dabhi and Chaudhary, 2015). An alternative that partitions a given finite dataset into three parts, namely training, validation, and test, has been studied by many researchers (Gagné et al., 2006; Zavoianu, 2010; Schmidt and Lipson, 2009). The training dataset is used for evolving models. The validation part is used for selecting the best model for a specific problem and can also be used for identifying overfitted models among the final population obtained. The test dataset is used for evaluating the generalization ability of the obtained solutions over unseen data. This approach prevents overfitting by detecting the overfitted models. The selected model over the validation dataset can further be compared with other models in an unbiased way over the test dataset which is not used in any part of the training and validation step. However, one concern in this approach is that in practice, the amount of available data can be too small to be partitioned into three parts. Methods aimed at reducing the complexity of models during the evolution have also drawn an increasing attention. Nikolaev and Iba (Nikolaev and Iba, 2001) presented a regularizated fitness function to discover parsimonious, accurate, and predictive solutions. In training process, MDL-based model complexity measurement was adapted for inducing tree-like polynomials and used for smoothing the fitness landscape by discarding complex models. Vladislavleva et al. (Vladislavleva, Smits, and Hertog, 2009) utilized a novel expressional complexity measure, the order of nonlinearity, to navigate the evolution process to develop compact models with smoother response surfaces. They drew a conclusion that models produced by GP which used the order of nonlinearity as a secondary optimization objective exhibit better extrapolative performance than those generated taking a size-related measurement as a secondary objective. This work was criticized by Ni and Rockett (Ni and Rockett, 2015), however, because the model selection issue in GP is simply switched to another model selection problem on the set of polynomial fits which they solve with an arbitrary threshold. The original model selection problem has not been solved in essence. Costelloe and Ryan (Costelloe and Ryan, 2009) improved the generalization ability of GP models by combining "no same mate" selection with linear scaling. Kotanchek et al. (Kotanchek, Smits, and Vladislavleva, 2008) proposed an ensemble of diverse homogeneous models to evolve more accurate models. Additionally, instead of splitting data into training,

validation and test dataset, this method can utilize all available data for model development without risk of overfitting.

2.1.4 Multi-Objective Optimization in GP

Another effective method to address overfitting problem in GP is using a multi-objective optimization scheme. Real-world problems are difficult to solve in that they are inherently characterized by multiple objectives which are often in conflict with each other. It is usually hard to find an optimal solution that can meet all criteria mathematically at the same time. Multi-objective optimization can produce a set of acceptable trade-off optimal candidates, among which practitioners can select the best solution for their own specific problems.

Traditional approaches transfer multiple objective optimization problems into single objective optimization problems by either developing a weighted function which aggregates all the objectives, or taking one objective as the optimization task while others as constraints. The former strategy is called the weighted sum method assigning a weight coefficient to each objective and then combining them into a weighted sum function. However, the determination of weight coefficients is difficult since it needs prior knowledge to decide the relative importance of each objective. To deal with this issue, Jin et al. (Jin, Olhofer, and Sendhoff, 2001) proposed Dynamic Weighted Aggregation (DWA) that changes the weights of the objectives incrementally in the evolution process. In system identification, a typical solution to model complexity control is regularization (Cherkassky and Mulier, 2007). Regularization adds a penalization term to the objective function to be optimized. The penalization term is used for evaluating model complexity. Typically, the objective to be optimized for the regularization principle is described as

$$R_{pen}(\omega) = R_{emp}(\omega) + \lambda \phi[f(\mathbf{x}, \omega)], \tag{2.2}$$

$$R_{emp} = \frac{1}{n} \sum_{i=1}^{n} (y_i - f(\mathbf{x}_i))^2, \tag{2.3}$$

where y_i is the sample measurement, and $f(\mathbf{x}_i)$ is the estimate of the approximating functions. Thus $R_{emp}(\omega)$ stands for the empirical risk for a specific learning problem, which intends to enforce closeness of the approximating models to the available data. $\lambda \phi[f(\mathbf{x}, \omega)]$ is a penalty term, which enforces smoothness. The regularization parameter λ controls the strength of

the penalty term and adapts the tradeoff between model accuracy and complexity. The setting of λ is often problematic and time consuming. Analytical arguments and data resampling are powerful tools that can be used to determine the optimal value of λ. For details, readers can refer to (Cherkassky and Mulier, 2007). The regularization framework has been frequently applied in empirical modelling tasks. It presents a formal mechanism to regulate model complexity during the training process.

Apart from the weighted sum algorithm, Coello (Coello Coello, 1999) introduced ε constraints, which focuses on solving one objective optimization problem while taking the other objectives as constraints restricted in some acceptable range. Those classical methods have limits that demand the user has prior knowledge about the underlying system, and the single solution evolved is highly sensitive to the weight vector used in the scalarization process.

Unlike aggregation-based single objective optimization, Pareto-based methods solve the optimization problem without coupling objectives. They optimize multiple objectives simultaneously so as to maintain individual features of each objective. A non-Pareto based technique named Vector Evaluated Genetic Algorithm (VEGA) (Schaffer, 1985) was proposed. It divides the whole population into several equal parts. Each sub-part evolves to develop the fittest solution for one objective. A clear drawback of VEGA is that the evolved models can only perform well in terms of one objective and have a bias towards some regions.

Pareto-based techniques prevent premature convergence and overfitting by sorting and assigning ranks to GP trees, which serves as the guide to the evolution process (Coello Coello, 1999; Ngatchou, Zarei, and El-Sharkawi, 2005). Multiple Objective Genetic Algorithm (MOGA) (Fonseca and Fleming, 1993) evaluates the performance of solutions from the number of other candidates it dominates. Non-dominated Sorting Genetic Algorithm (NSGA) (Srinivas and Deb, 1994) combines the non-dominated sorting in GAs with a niche and speciation method to search for Pareto optimal solutions. The Niched Pareto Genetic Algorithm (NPGA) (Horn, Nafpliotis, and Goldberg, 1994) adjusts the selection pressure and controls the convergence speed by adding Pareto dominance to the tournament selection scheme.

Strength Pareto Evolutionary Algorithm (SPEA) (Zitzler and Thiele, 1999) combines several features of previous multiobjective EAs in a unique manner. This approach stores nondominated solutions externally in a second, continuously updated archive population. The fitness of a solution is estimated from the number of external nondominated points that

dominate it. A clustering procedure is applied for reducing the size of the nondominated set without destroying its characteristics.

A faster version of NSGA, named Non-dominated Sorting Genetic Algorithm-II (NSGA-II), was proposed by Deb et al. (Deb et al., 2000). It adopts an elitism strategy to prevent the loss of the best previously found solutions. Experiments show the NSGA-II can produce an improved spread of solutions and convergence near the true Pareto-optimal front.

Pareto Archived Evolution Strategy (PAES) (Knowles and Corne, 2000) employs local search but using a reference archive of previously found solutions in order to identify the approximate dominance ranking of the current and candidate solution vectors.

Being able to simultaneously deal with a set of objectives, evolutionary algorithms are particularly suitable for solving multi-objective optimization problems. Additionally, evolutionary algorithms have minimal requirements regarding the problem formulation. The fact that Pareto-based multi-objective algorithms finally approximate a set of Pareto optimal solutions provides flexibility for researchers to select the best model according to different problems. All those variants of multi-objective optimization algorithm have performed well in various applications (Sharma and Virk, 2014). Due to this advantage, in this thesis, we adopted the basic Pareto-based multi-objective algorithm and integrated it in GP for solving nonlinear system identification and control problems.

2.2 Details of GP

2.2.1 GP Evolution Process

Inspired by biological evolution processes, evolutionary algorithms (EA) solve problems by applying the theory of natural selection to an assembly of candidate solutions with the expectation of evolving better models. GA are one class of EAs. Basically, a GA consists of a reproductive strategy for generating offspring with better fitness using the principal genetic operators of crossover and mutation. GP is a subset or an extension of GA. The essential principles of GA and GP are similar although solutions in GP are expressed as programs with hierarchical tree structures, which consist of pre-specified functional and terminal nodes. This flexible tree structure provides a dynamic and variable representation. A typical example of

such a GP tree is shown in Figure 4.1, and its functional expression is given in

$$y(k) = (-x_1) + (0.2 * (-x_2)). \tag{2.4}$$

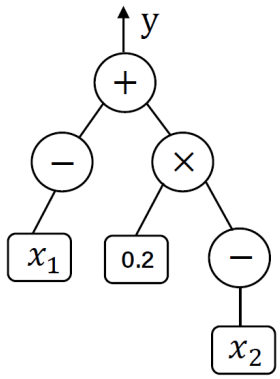

FIGURE 2.1: Simple example GP tree.

Generally, evolutionary algorithms can be classified into two different types: steady-state and generational. In steady-state evolution, one (or two) offspring are produced at each step and appended to the population; the population size is then reduced down to its original size by removing the weakest one (or two) individuals. (In fact, the term 'steady-state' is a misnomer – quasi-steady state would be more accurate.) In generational algorithms, on the other hand, a whole new child population is produced by repeated selection from a parent population before the child population is swapped to become the parent population and the process repeated. The following algorithm describes the evolution process of a typical steady-state GP.

- Step 1: Population initialization

  In GP, candidates in the initial population are randomly generated. The process of creating random trees can be implemented in different ways (Poli, Langdon, and McPhee, 2008), but two simple methods (the 'full' and 'grow' strategies (Poli, Langdon, and McPhee, 2008)) and are extensively used. In the 'full' method, the initial trees are created up to a pre-defined maximum depth; the depth of a GP tree is the minimum number of layers that need to be crossed to reach the deepest terminal node from a tree's root node. Trees are generated by randomly selecting nodes from the function set until all the leaves reach the maximum tree depth. In the 'grow' method, trees are

created with more diverse structures with some probability of terminating tree growth before reaching the depth limit.

In order to initialize the population of trees with a variety of shapes and sizes, a *ramped half-and-half* method was proposed by Koza (Koza, 1992). This initialization strategy generates half of population based on the 'full' method and the remaining trees based on the 'grow' method. In the present work, we have used the *ramped half-and-half* method for population initialization.

- Step 2: Fitness evaluation

  The performance of each tree is evaluated with a fitness function, which is used for estimating how well a solution performs on the given problem. Details about the fitness estimation are described in the following section 2.2.2. Then the population is sorted according to fitness value. Solutions with higher ranks are more likely to be selected as parent trees to breed child candidates in the evolution process.

- Step 3: Offspring generation

  At each iteration, two GP trees are selected as parents. Two main genetic operations, crossover and mutation, are then applied to produce new offspring solutions. Specifically, the crossover operator randomly selects a crossover point in each parent tree. The child trees are then generated by crossing over and splicing together the two trees at the selected crossover points between two parent trees, as illustrated in Figure 2.2. The mutation operation modifies a GP tree by randomly selecting a mutation point in a tree, and then replacing it with a new, randomly-generated subtree, as illustrated in Figure 2.3. After crossover and mutation, the fitness value of each newly generated child tree is evaluated. The population is then re-ranked after appending the offspring solutions and the two least-fit individuals deleted to return the population to its original size.

- Step 4: Process termination

  The above procedures are iterated from step 2 to step 3 until user-specified termination conditions are met.

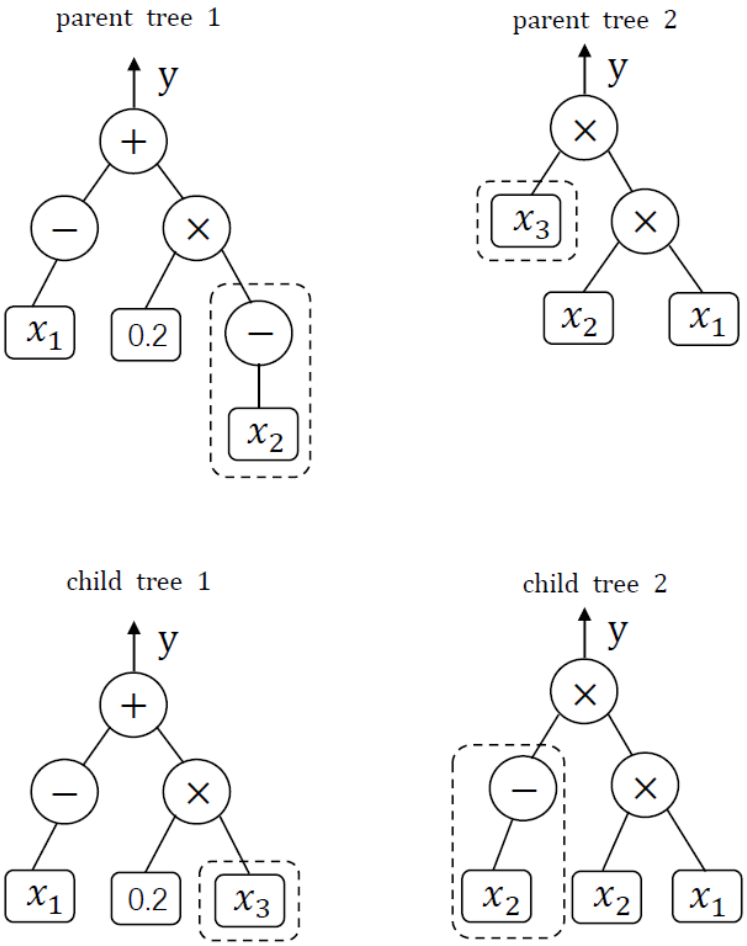

FIGURE 2.2: Example of subtree crossover.

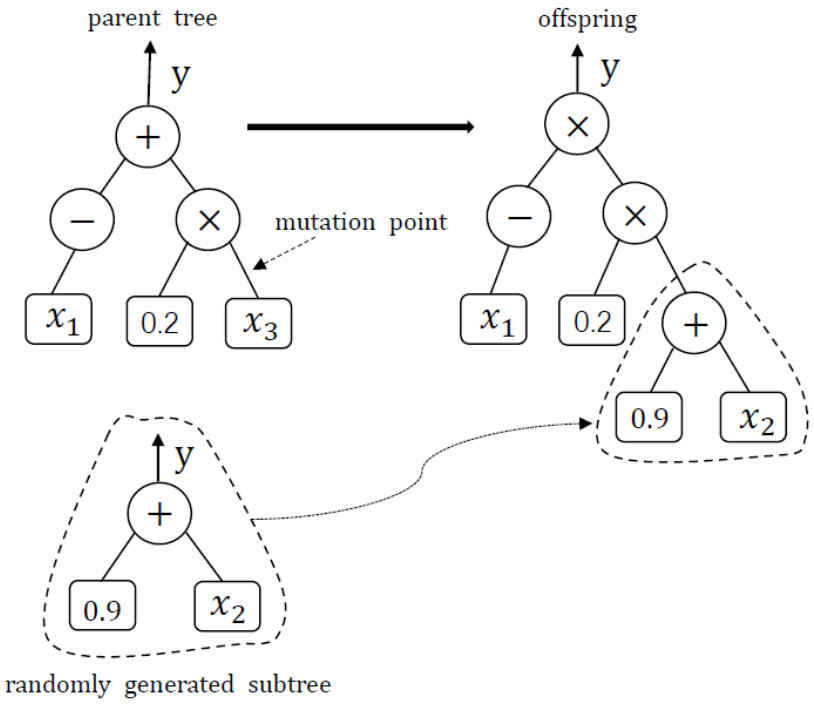

FIGURE 2.3: Example of subtree mutation.

2.2.2 Pareto Based Multi-Objective Optimization and Model Selection

One of the fundamental issues of GP is bloat – the inexorable growth in tree size with no accompanying improvement in fitness. The use of multiobjective optimization in GP, however, can reduce the effects of bloat by providing a selective pressure that favours smaller models – so-called parsimony pressure. In order to generate compact and accurate models, we have used the twin fitness measures of tree size (number of tree nodes) and MSE over the dataset as two non-commensurable objectives. The Pareto dominance based ranking scheme (Goldberg, 1989) was used to rank the individuals in the population. A vector of objectives $\mathbf{a} = (a_1, ..., a_p)$ is said to dominate $\mathbf{b} = (b_1, ..., b_p)$ if and only if $\mathbf{a}$ is partially less than $\mathbf{b}$, i.e., $\mathbf{a} \prec \mathbf{b}$ $\forall i : a_i \leq b_i \wedge \exists i \in 1, ..., p : a_i < b_i$; in our case the fitness vectors are the 2-vectors with node count and MSE as elements.

During the evolution process, trees with higher ranks have bigger probabilities of being selected as parent trees to produce child candidates, and at the end of the run, the population comprises a set of individuals which trades-off compactness against the goodness of fit (small MSE) to the training data.

After the evolution, the final population obtained spans the spectrum of small individuals with large MSE values (under fitted models) through to large models with small MSE values (overfitted models). The model with the smallest MSE over the validation set was selected as the final solution.

2.3 Other Comparable System Identification Techniques

Except for GP, there are other frequently used techniques used for solving dynamic system identification problems. Due to the fact that most real-world processes are nonlinear and complex, the discussion about techniques for linear system identification is beyond the scope of this thesis, and only nonlinear dynamical algorithms are reviewed here. Traditional statistical nonlinear models have a range of applications in empirical modelling tasks (Nelles, 2001). In the discrete time domain, the most widely used methods are nonlinear ARMAX (NARMAX), nonlinear AR (NAR), and nonlinear ARX (NARX). The Volterra-series models, block-structured models, such as Hammerstein models, have also been widely researched for determining mathematical descriptions of nonlinear systems (Nelles, 2001). From a statistical view, Gaussian processes exhibit great potential in empirical modelling and

have been extensively researched in system identification and control applications (Kocijan et al., 2004; Grancharova, Kocijan, and Johansen, 2007; Cao, Lai, and Alam, 2017). In recent decades, another family of algorithms has attracted an increasing number of researchers (Nelles, 2001). These algorithms are called modern heuristic techniques, among which the most widely studied techniques include ANNs, and evolutionary computational algorithms.

The methods described above are frequently applied in regression, classification, control and prediction problems (Nelles, 2001). Particularly, there is an increasing number of attempts that use data-driven models to improve the design of controllers. For control purposes, though the relationship of the controller performance and the model performance are not strictly associated, the former is still closely linked with the latter (the well-performing controller certainly results from an appropriate model). Thus, in model-based control applications, model accuracy is a key research issue.

In this section, several contemporary comparable models (ANNs, Volterra-series models, Hammerstein models, and Gaussian process models) and their applications in the field of nonlinear system identification and control are described. In this thesis, we have not compared the performance of GP with those models. The reason why other comparable techniques are discussed is to illustrate that apart from GP, ANNs, Volterra-series models, Hammerstein models, and Gaussian process models have also recently been frequently applied to nonlinear system identification and control applications (Fu et al., 2013; Yuzgec, Becerikli, and Turker, 2008; Sentoni et al., 1996; Kocijan et al., 2004; Cao, Lai, and Alam, 2017).

2.3.1 Artificial Neural Networks for System Identification and Predictive Control

An artificial network (or simply a neural network) is a network that connects simple processing elements referred to as neurons. Artificial neurons are elementary units in an artificial neural network. Generally, a neuron is a function that maps multiple inputs to a scalar. Figure 2.4 illustrates a very simple form of a neuron. It receives one or more inputs and sums them to produce an output.

In Figure 2.4, the example neuron has $m+1$ inputs. u_1 to u_m are the inputs. b is a bias input. w_1 to w_m are the weight parameters. v is the sum of all the inputs multiplied by their

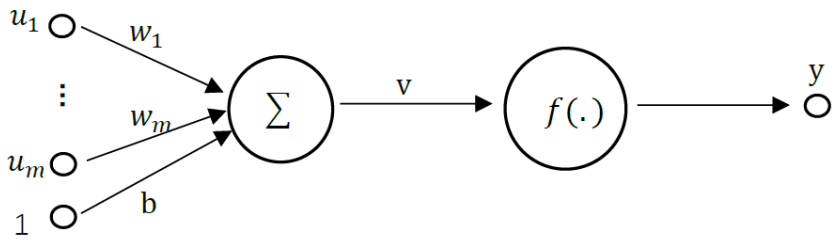

FIGURE 2.4: A general model of a neuron of ANN.

weights. $f(.)$ is typically a nonlinear transfer function and y is the output of the neuron. Accordingly, the output of a neuron is calculated by

$$y = f(v) = f(\sum_{i=1}^{m} u_i * w_i + b) \tag{2.5}$$

An ANN is based on a collection of connected neurons. The basic structure of an ANN is illustrated in Figure 2.5.

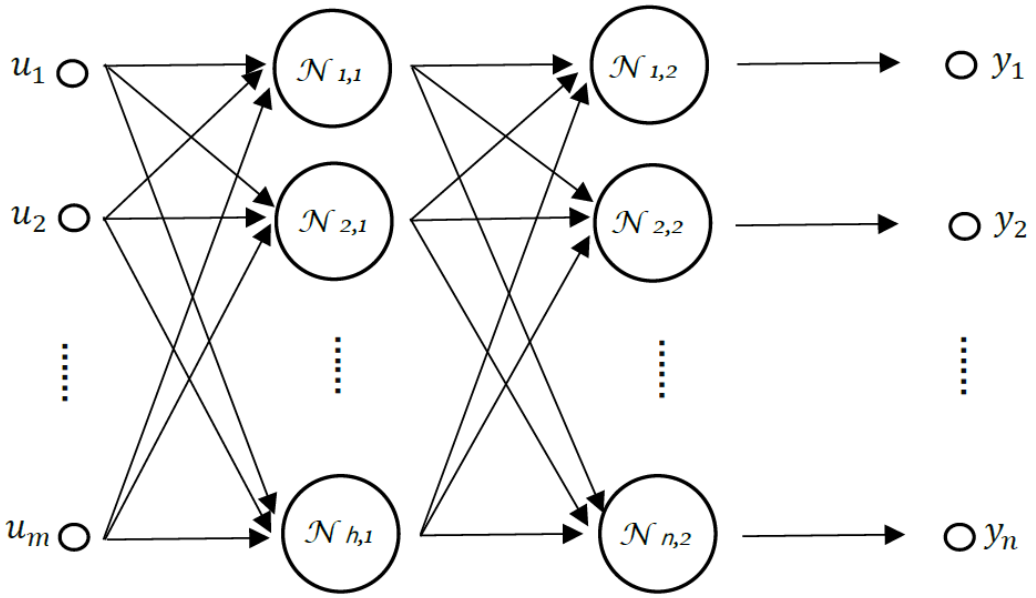

FIGURE 2.5: A simple artificial neural network.

In ANN, the layer used for receiving signals from the environment is named the input layer. The layer that produces outputs is called the output layer. All the intermediate layers located between the input layer and the output layer are called hidden layers. The considered example in Figure 2.5 has one hidden layer. Figure 2.5 shows a two-layer ANN. Since there are no calculations done in the input layer, it is not taken into account when the number of layers is calculated.

Learning is the process of adapting the connection weights in an ANN to produce the desired output vector in response to a stimulus vector presented to the input buffer (Tsoukalas and Uhrig, 1996). Typically, a gradient-based algorithm is used for the optimization process, since it is suitable for local search.

In general, neural networks used for nonlinear system identification can be classified into two types, feedforward and recurrent. A clear limitation of feedforward neural network models is that the map from inputs to outputs is static, which arouses a big challenge in modelling complex dynamical systems (Fairbank et al., 2014; Phan and Hagan, 2013).

Recent works (Pan and Wang, 2012; Seyab and Cao, 2008; Fu et al., 2013; Yuzgec, Becerikli, and Turker, 2008; Shen et al., 2014; Lu and Tsai, 2008; Hosen, Hussain, and Mjalli, 2011; Tai and Ahn, 2012; Na et al., 2012; Xiong and Zhang, 2005; Han and Qiao, 2011; Chen, 2011; Han and Qiao, 2014) show that recurrent neural networks (RNNs) are suitable for nonlinear dynamical system identification problems as they are able to conduct long-range predictions, even in the presence of measurement noise. Based on two RNNs, an echo state network and a simplified dual network (Pan and Wang, 2012) were developed for MPC of an unknown nonlinear dynamical systems. The results show the RNN-based nonlinear MPC scheme is effective and potentially suitable for real-time MPC implementation. A continuous time recurrent neural network (CTRNN) (Seyab and Cao, 2008) was proposed for nonlinear MPC, and improved model accuracy for the nonlinear process was obtained using the new method. Fu et al. (Fu et al., 2013) utilized two dynamic multilayer neural networks with different timescales to solve the adaptive nonlinear identification and trajectory tracking tasks. The results demonstrate the effectiveness of the proposed identification and control algorithms.

Yuzgec et al. (Yuzgec, Becerikli, and Turker, 2008) proposed a dynamic neural-network-based MPC structure for a baker's yeast drying process. The RNNs used in (Pan and Wang, 2012; Seyab and Cao, 2008; Fu et al., 2013; Yuzgec, Becerikli, and Turker, 2008) and some others (Shen et al., 2014; Lu and Tsai, 2008; Hosen, Hussain, and Mjalli, 2011) show promising performance both on nonlinear dynamic system identification problems and control applications. However, the research on RNNs for nonlinear system modelling still faces difficulties (Tai and Ahn, 2012; Na et al., 2012). The modelling efficiency is sensitive to the structures and parameters of RNNs. Thus, they are often fixed after initialization, which may cause troubles in the nonlinear MPC (Xiong and Zhang, 2005). To deal with this problem,

recent works (Han and Qiao, 2011; Chen, 2011; Han and Qiao, 2014) pointed out that a self-organizing scheme can be used for improving the model effectiveness.

Despite ANNs have been successfully employed for approximating many real-world systems, the shortcoming of not being able to produce an explicit mathematical equation which uncovers the relationships between inputs and outputs of a system impairs their application flexibility. In addition, some critical settings of the structure of an ANN (i.e., the number of hidden layers and transfer functions) require to be specified by knowledgeable experts in advance. Therefore, more robust approximation methods are desired for identifying complex engineering processes.

2.3.2 Volterra-Series Models for System Identification and Predictive Control

In nonlinear system identification, classical models are developed based on polynomials for approximating the nonlinear mapping between inputs and outputs. Volterra series are a direct generalisation of the linear convolution integral (Billings, 2013). It represents the nonlinear behaviours of systems without output feedback (Nelles, 2001). A typical Volterra-series model for a discrete-time causal system is shown

$$y(n) = h_0 + \sum_{i=1}^{P} (H_i x)(n) \tag{2.6}$$

$$(H_i x)(n) = \sum_{\tau_1=0}^{M} \dots \sum_{\tau_i=0}^{M} h_i(\tau_1, \dots, \tau_i) \prod_{j=1}^{i} x(n-\tau_j) \tag{2.7}$$

where $h_i(\tau_1, \dots, \tau_i)$ are discrete-time Volterra kernels and h_0 is a constant term (Orcioni, 2014). $x(n), y(n) \in \mathbb{R}$, are the system input and output, respectively. The input to a Volterra-series system at all other times can be used to calculated the output. $P \in \mathbb{N} \cup \{+\infty\}$, $M \in \mathbb{Z} \cup \{+\infty\}$. In function 2.7, the summations begin with 0 meaning a causal system is considering here.

Volterra-series models have been widely used for nonlinear system identification and control applications(Gruber et al., 2011; Gruber, Bordons, and Oliva, 2012; Gruber et al., 2013; Li, Qi, and Yu, 2009; Kumar and Budman, 2014). Tan and Jiang (Tan and Jiang, 2001) proposed a Volterra filtered-X least mean square (VFXLMS) algorithm to control feedforward active noise. The experimental results indicate the VFXLMS algorithm is of great potential

in active noise control where the noise to be controlled exhibits nonlinearities. Since a wide range of nonlinear behaviours can be described by these models, Maner et al. (Maner et al., 1996) introduced a control methodology based on second-order Volterra-series models. They demonstrated that improved control performance was obtained over linear model predictive controllers in both single input single output (SISO) isothermal reactor, and a multivariable large reactor control experiment. A current-dependent model based on Volterra-series was developed by Mastromauro et al. (Mastromauro, Liserre, and Dell'Aquila, 2008) for investigating the effects of nonlinear inductance on the performance of current controllers. The mitigation capability of the proposed controllers was justified by the Volterra-series models.

To solve the boundary control of nonlinear parabolic partial differential equations (PDEs), Vazquez and Krstic (Vazquez and Krstic, 2008a; Vazquez and Krstic, 2008b) proposed a stabilizing control framework which applied spatial Volterra series to a stable linear PDE transformation and a feedback law. Their work illustrated that the inverse of the transformation can be represented by explicit construction and is at least locally feasible. The thermal features of a greenhouse exhibit strong nonlinearities. Gruber et al. (Gruber et al., 2011) presented a nonlinear model predictive control (NMPC) scheme based on a second-order Volterra-series model for greenhouse temperature control. The proposed NMPC was shown to be able to regulate the greenhouse temperature in simulation, and an adequate control performance was also observed in a real greenhouse test. For safety and performance reasons, Gruber et al. (Gruber, Bordons, and Oliva, 2012) designed a nonlinear model predictive controller for regulating the oxygen flow rate in a polymer electrolyte membrane or proton exchange membrane fuel cells. The NMPC was based on a second-order Volterra-series model. Compared with linear MPC and a built-in controller, the proposed NMPC provided better control performance without high computational requirements. Li et al. (Li, Qi, and Yu, 2009) constructed a spatiotemporal Volterra-series model with a set of associated spatiotemporal kernels for representing unknown nonlinear distributed parameter systems (DPS). In the simulation, the Volterra-based models generated were simple and of low dimensionality, which justified its effectiveness and feasibility for further applications, such as control and prediction of the DPS. A nonlinear Min–Max model predictive controller (MMMPC) based on a second-order Volterra-series model (Gruber et al., 2013) was proposed for achieving an optimal control sequence by optimizing user-defined objective functions. The effectiveness of the proposed MMMPC was demonstrated in a continuous stirred tank reactor experiment.

Kumar and Budman (Kumar and Budman, 2014) developed a robust nonlinear model predictive controller based on Volterra series. The uncertainty in the Volterra series coefficients is expressed by polynomial chaos expansions (PCE). In a pH-neutralization process simulation, the proposed PCE-based NMPC provided better control performance with lower computational cost compared with another robust controller and a non-robust controller.

The Volterra-series models are universal approximators in nonlinear system identification. The model structure requires no prior assumption. Due to its linear-in-parameters nature, model parameters can be identified using the least squares method. Therefore, Volterra series is a straightforward tool to research nonlinear system identification problems. However, some challenging research issues still exist. The core to Volterra-series modelling in nonlinear system identification is the identification of its kernel functions. Typically, a large number of parameters in its kernel functions needs to be identified to describe a nonlinear system adequately so estimation of the elevated number of kernel parameters impairs the efficiency of Volterra-series kernel function identification. Additionally, the convergence of Volterra series is still an open question, which imposes a big challenge in Volterra-based modelling (Cheng et al., 2017).

2.3.3 Hammerstein Model for System Identification and Predictive Control

A typical Hammerstein model consists of a static nonlinear model followed by a dynamic linear block, as illustrated in Figure 2.6.

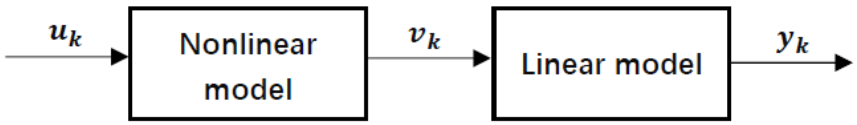

FIGURE 2.6: Hammerstein model structure.

In the Hammerstein model structure in Figure 2.6, $\mathbf{u}_k$ is the nonlinear block input, $\mathbf{v}_k$ is linear block input and $\mathbf{y}_k$ is linear block output. In the most typical case, the linear dynamic part is identified with classical statistical models, (i.e. finite impulse response (FIR) or infinite impulse response (IIR) models) and the nonlinear part uses polynomials with low order ($n <= 3$) (Jeraj and Mathews, 2006; Voros, 1999; Sbeity et al., 2012; Yu et al., 2008; Jeraj, Mathews, and Dubow, 2002; Costa, Lagrange, and Arliaud, 2003; Tingqi, Changlu, and Wenjiang, 2000; Ralston and Zoubir, 1995; Gotmare, Patidar, and George, 2015). In this

kind of cascade model, the approximation process is flexible and simple to use. However, in this approach, an accurate approximation of complex nonlinearities is more likely to be obtained with high polynomial order (Ławryńczuk, 2013). This increases the difficulty of identification and oscillatory interpolation. Neural networks (Ławryńczuk, 2014) have also been employed to implement the static part of a Hammerstein model, but the training process of determining appropriate structure and parameters is not easy. A static nonlinear model approximated with a support vector machine (SVM) (Ławryńczuk, 2013) helped to build a computationally efficient nonlinear MPC algorithm. Unfortunately, the model was criticized for lack of spareness since it consists of a large number of support vectors and parameters. In addition to those memoryless techniques, dynamic polynomial nonlinear models have also been analyzed for approximating the nonlinear part of Hammerstein models. For instance, truncated Volterra-series models have been widely used for nonlinear system representations and can be employed for the nonlinear part in a Hammerstein model (Mete, Ozer, and Zorlu, 2016; Ozer, Zorlu, and Mete, 2016).

Being able to capture both linear and nonlinear features, Hammerstein models have been widely applied for approximating a large number of real-world processes, such as a binary distillation column (Fruzzetti, Palazoğlu, and McDonald, 1997), a pH neutralization process (Du, Song, and Li, 2009), a continuous stirred tank reactor (Menold, Allgöwer, and Pearson, 1997), a solid oxide fuel cell (Huo et al., 2008), a stretch reflex process (Jalaleddini and Kearney, 2013), a turntable servo system (Zhang, Wang, and Li, 2016), and so on.

Besides, in terms of control, various MPC schemes have been proposed for nonlinear dynamic processes modelled by Hammerstein models. The problem of how to deal with the static nonlinearity for optimizing control laws is the main research topic. Sentoni et al. (Sentoni et al., 1996) applied a Hammerstein model to an MPC controller directly with the control law designed by solving a complex non-convex nonlinear optimization problem. Better control performance was observed by taking the nonlinearity into account through the polytopic description (Bloemen, Boom, and Verbruggen, 2000; Bloemen, Boom, and Verbruggen, 2001). Recently, the most commonly used control strategy for Hammerstein models is the nonlinearity inversion-based MPC algorithm which consists of a linear controller followed by the inversion of the input nonlinearity (Zhang, Chin, and Ławryńczuk, 2018; Fruzzetti, Palazoğlu, and McDonald, 1997; Patwardhan, Lakshminarayanan, and Shah, 1998; Chan and Bao, 2007). All of the inversion-based algorithms assume that the nonlinear process is

ivertable. However, in practice, the nonlinearity of systems is not always invertible or well-conditioned. In this circumstance, inversion may cause numerical problems for MPC using Hammerstein models. Therefore, more robust system identification methods are required to form a reliable basis for further MPC.

2.3.4 Gaussian Process Models for System Identification and Predictive Control

When the nonlinear dynamic process under research is seen from a statistical perspective, Gaussian process models are commonly used techniques for system modelling. A Gaussian process model is a probabilistic, non-parametric black-box model.

Unlike other approaches, it does not approximate systems by optimizing the parameters of the pre-specified functions but rather learn the underlying dynamics among available measurements. The output of a Gaussian process model is a normal distribution represented by mean and covariance functions. Additionally, prediction uncertainties are also expressed by Gaussian process models. All the information provided by Gaussian process models can be efficiently utilized for stochastic MPC design.

Mathematically, a Gaussian process is a set of random variables which have a joint multivariate Gaussian distribution (Kocijan et al., 2004). Given the input $\mathbf{x}$ and output $\mathbf{y}$, and their relationship $\mathbf{y} = f(\mathbf{x})$, we have $y^1, y^2, ..., y^n \sim \mathcal{N}(0, \sum)$, where $\sum_{pq} = Cov(y_p, y_q) = C(\mathbf{x}_p, \mathbf{x}_q)$ calculates the covariance between output points corresponding to input points $\mathbf{x}_p$ and $\mathbf{x}_q$. The mean $\mu(\mathbf{x})$ is usually set to zero. The covariance function $C(\mathbf{x}_p, \mathbf{x}_q)$ evaluates the covariance between pairs of random variables. It is also referred to as the kernel. Only those functions that produce a positive definite covariance matrix can be used as kernels. A common choice is the squared exponential function (Kocijan et al., 2004)

$$C(\mathbf{x}_p, \mathbf{x}_q) = v_1 \exp[-\frac{1}{2}\sum_{d=1}^{D} w_d (x_p^d - x_q^d)^2] + v_0 \tag{2.8}$$

where $w_1, ...w_D, v_0, v_1$ are the hyperparameters of the covariance functions and D the input dimension. The w parameters are calculated based on the available measurement data. For more kernel options, readers can refer to (Rasmussen, 1997).

Given a training dataset which contains a set of N D-dimensional input vectors $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N]$ and its corresponding outputs $\mathbf{y} = [y^1, y^2, ..., y^N]^T$, the distribution of the output

y^* of a new D-dimensional input vector $\mathbf{x}^*$ can be obtained. Similarly, the distribution of the new output is also shown to have a Gaussian distribution with mean and variance (Kocijan et al., 2004) estimated by

$$\mu(\mathbf{x}^*) = \mathbf{k}(\mathbf{x}^*)^T \mathbf{K}^{-1} \mathbf{y} \tag{2.9}$$

$$\sigma^2(\mathbf{x}^*) = k(\mathbf{x}^*) - \mathbf{k}(\mathbf{x}^*)^T \mathbf{K}^{-1} \mathbf{k}(\mathbf{x}^*) + v_0 \tag{2.10}$$

where $\mathbf{k}(\mathbf{x}^*) = [C(\mathbf{x}^1, \mathbf{x}^*), ..., C(\mathbf{x}^N, \mathbf{x}^*)]^T$ is the vector of covariances between the test and training cases. $\mathbf{K}$ is the covariance matrix based on training set $\mathbf{X}$, and the $k(\mathbf{x}^*) = C(\mathbf{x}^*, \mathbf{x}^*)$ is the covariance of the new input itself.

When Gaussian process models are applied to nonlinear dynamic system identification problems, delayed input and output signals serve as regressors. For more details of Gaussian process models, one can refer to the book (Kocijan, 2016b).

A Gaussian process based MPC scheme for nonlinear dynamic systems was first introduced by Kocijan et al. (Kocijan et al., 2004). The results show that Gaussian process models offer an attractive possibility for MPC that generates a controller with a higher level of robustness due to information contained in the model. Latter, Grancharova et al. (Grancharova, Kocijan, and Johansen, 2007) proposed an approximate multi-parametric nonlinear programming approach for explicit solution of stochastic NMPC problems based on Gaussian process models. The algorithm develops an orthogonal search tree structure of the state space partition and consists in constructing a feasible piece-wise linear approximation to the optimal control sequence. Based on a Gaussian process regression framework, Klenske et al. introduced a locally periodic covariance function to shape the hypothesis space (Klenske et al., 2016). The forecasts calculated from the Gaussian process model are then fed into the MPC framework to correct the external errors. Recently, Cao et al. (Cao, Lai, and Alam, 2016b) proposed an MPC methodology based on the probabilistic Gaussian Process models and demonstrated its effectiveness on a simulated linear time-varying system. A Gaussian process model based MPC was used for designing appropriate quadrotor controllers (Cao, Lai, and Alam, 2016a). Two Gaussian process models based approaches (Cao, Lai, and Alam, 2017) were then applied to two trajectory tracking problems. Results show that both of them produced effective control.

When Gaussian models are used for system identification, some disadvantages also need to be considered: during the training process, calculating the determinant of a large matrix seems to be a big problem with Gaussian process. Additionally, the approximation accuracy of Gaussian process models relies on the training data and the selection of covariance function with hyperparameters, which is problem dependent (Kocijan, 2016a).

2.3.5 Summary

For solving nonlinear system identification and MPC problems, some competing techniques to GP, such as ANNs, Hammerstein models and Gaussian process models, are widely adopted for approximating real-world systems. Each of them has their own advantages and disadvantages.

ANNs are the most frequently used pattern-recognition approach. They are able to learn and model non-linear and complex relationships between system inputs and outputs. When ANNs are used for system identification, they do not impose any restrictions on input and residual distributions. Benefitting from those advantages, ANNs have been succesfully used for dealing with a range of materials and structural engineering problems (Guzelbey, Cevik, and Gögüş, 2006; Guzelbey, Cevik, and Erklig, 2006; Gandomi and Alavi, 2011; Guven, 2011; Pala, 2006). On the other hand, it is also important that we are aware of the disadvantages of ANNs. ANNs do not usually produce a definite function to describe the explicit relationships between inputs and outputs. They are mostly utilized as a part of a computer program, thus it is hard to further assess the behavior on many engineering problems. The structure of ANNs (i.e., the number of hidden layers, transfer functions) needs to be predefined, but there is no universal rule for determining the structure of ANNs. Appropriate model structures for complex systems are usually decided by experts through trial and error, which is laborious and time-consuming. Stanley and Miikkulainen (Stanley and Miikkulainen, 2002) pointed out that this difficulty can be alleviate by using NEAT method.

Volterra series is a direct generalisation of the linear convolution integral (Billings, 2013). Practically, Volterra-based models can be obtained straightforwardly from the measurement data without prior consideration of model structure. Being able to describe nonlinearities of various processes, Volterra-series models are extensively researched in a range of nonlinear system identification problems. However, the identification of Volterra kernel function is

difficult due to the evaluation of a large number of kernel parameters. In addition, the convergence of Volterra series remains a challenging problem. There is no universal technique that can accurately specify the interval of convergence of Volterra series representation, which deteriorates the efficiency of empirical modelling based on Volterra-series models.

Hammerstein models are formed by the cascade of linear and nonlinear models. Being able to represent both linear and nonlinear characteristics, Hammerstein models have been widely used for modelling practical processes, such as heat exchangers, electric drives, thermal microsystem, sticky control valves, magneto dampers (Mete, Ozer, and Zorlu, 2016). The linear part of a Hammerstein model is usually identified by classical statistical models. However, the determination of the nonlinear part is not easy. For instance, in the most typical case, polynomials are used to capture the nonlinearity of a system, but an accurate nonlinear approximation often requires a complex high polynomial order. Other commonly used techniques, such as ANNs, SVM and Volterra-series models, have their own issues (like the selection of model structure, the determination of model parameters) that need to be carefully addressed in modelling complex engineering systems.

Gaussian process models approximate systems from a statistical perspective. As an empirical modelling tool, Gaussian process models have an advantage that only the covariance function and the regressors of the model need to be selected. Moreover, Gaussian process models allow the incorporation of prior knowledge, which is helpful in improving the model accuracy. Besides, there are fewer parameters required to be optimized in Gaussian process models compared with other other black-box modelling techniques. Nevertheless, some disadvantages also need to be addressed when using Gaussian process models for identifying complex systems. The computational complexity of optimization grows with amount of data and number of regressors. In addition, the performance of Gaussian process models is sensitive to the training data and the covariance functions selected.

Consequently, complex engineering system identification calls for robust computing techniques. GP has been receiving a considerable amount of attention, both as an identification tool and as a controller. There are several main supporting reasons for this trend. First, benefitting from the fact that a function set in GP can include various linear and nonlinear functional nodes, GP has the ability to capture both linear and nonlinear features of systems from data. Second, GP does not make any prior assumptions about the distribution of measured data, which makes GP flexible and simple to use. Third, GP is able to automatically

optimize both model structure and appropriate parameters of a model at the same time. It produces explicit approximating functions without assuming a prior form of the existing relationships between inputs and outputs. This implies that the model structure is to a large extent determined by the data. Additionally, incorporating a multi-objective optimization scheme, GP is able to solve complex system identification problems with multiple and often conflicting criteria optimized separately and simultaneously. Therefore, in this thesis, GP is selected and the performance of GP on system identification and control applications is investigated.

Chapter 3

Symbolic Regression System Identification Using Genetic Programming

A series of experiments that use semantic-based local search within a MOGP framework are reported in this section. We compare various ways of selecting target subtrees for local search as well as different methods for performing that search; we have also made comparison with the RDO of Pawlak et al. using statistical hypothesis testing. We find that a standard steady state or generational GP followed by a carefully-designed single-objective GP implementing semantic-based local search produces models that are more accurate and with statistically smaller (or equal) tree size than those generated by the corresponding baseline GP algorithms. The depth fair selection strategy of Ito et al. is found to perform best compared with other subtree selection methods in the model refinement.

3.1 Author's Contribution

1. A set of semantic-based local search algorithm within a multiobjective GP framework were proposed. The performance of these semantic-aware combinatorial methods was evaluated and compared over a series of benchmark regression functions.

2. Considering the issue of model complexity control, the proposed local search methods were designed with size restriction to prevent tree growth.

3. Unlike the common local search way to optimize models by fine-tuning model parameters or node functions, this work improved model accuracy by modifying tree morphologies, which are comparatively little explored in the literature.

4. The proposed semantically-aware local search is proven to be able to evolve GP trees with statistically better generalization performance and smaller tree sizes than a variant of GP which was proposed recently and exhibited powerful search ability using RDO as genetic operator (Pawlak, Wieloch, and Krawiec, 2015).

5. By analyzing the influence of various local search methods, Ito's depth fair selection method outperforms others. Ito's depth fair selection method optimize GP trees by more frequently tuning subtrees closer to the root node.

6. In addition, the performance of GP using RDO as a genetic operator and local search operator in steady state and generational multi-objective GP was studied and analyzed.

3.2 Introduction

It is well established that GP exhibits good performance on the empirical modelling of complex systems (Poli, Langdon, and McPhee, 2008). Nonetheless, traditional GP still has the limitation that since it acts at the syntactic level, a small syntactic modification can produce a dramatic change in program fitness, which can harm search efficiency.

To address these issues, the integration of local search into GP has attracted significant attention (Iba, Garis, and Sato, 1994; Topchy and Punch, 2001). At a wider level, the hybridization of population-based global search with heuristic local search – often termed a memetic algorithm (Moscato, 1989) – has achieved notable successes (Neri, Cotta, and Moscato, 2012) although remains comparatively little used in GP. Typically, existing GP memetic algorithms may include hill climbing local search over the coefficients or the model structures of the GP solutions.

In GP, semantics usually refers to the vector of output values a program produces over the training data (Vanneschi, Castelli, and Silva, 2014) and has been the subject of much recent research. Experimental results to date suggest that awareness of semantics is a great help both in maintaining population diversity and improving search power. Among these approaches, semantic-based local search methods (Ffrancon and Schoenauer, 2015; Vanneschi, Castelli,

and Silva, 2014) have exhibited promising performance. Based on semantics, Pawlak et al. (Pawlak, Wieloch, and Krawiec, 2015) have implemented a novel genetic operator RDO, which decomposes the search task into a series of subtasks. By backpropagating the desired semantics to individual subtrees, the fitness of a solution can be improved by seeking to replace a selected subtree with a subtree better matching the desired semantics. Since replacement subtrees are selected from a (pre-computed) library, it is unclear whether RDO is a local search method or a (global) crossover method as claimed in (Pawlak, Wieloch, and Krawiec, 2015); clarifying this question is a part of the motivation for the present work.

Local search inevitably adds to the computational burden and runtime of GP, which seemingly makes many practitioners wary of local search-based approaches. For this reason, we have restricted the present work to local search – where employed – for tuning the solutions at the end of a conventional GP run. We believe this approach thus closely fits with the conventional memetic algorithm formulation (Moscato, 1989) (although some may argue that in a memetic algorithm, the local search would be embedded within the global search). The principal contribution of this section is an investigation of the performance of GP approaches when supplemented by semantically-aware local search methods. In particular, this work extends consideration of the effectiveness of local search to a MOGP framework since this explicitly trades off goodness-of-fit against model complexity, a key requirement in the empirical modelling of data (Cherkassky and Mulier, 2007; Le et al., 2016). For the reasons stated in the preceding paragraph, we also make comparison with the RDO approach. We specifically constrain the scope of this work to local search methods that modify the morphologies of the GP trees rather than approaches described in Section 3.3 that fine-tune node functionality. Local search methods that change the tree morphologies are comparatively little explored.

3.3 Related Work

3.3.1 Semantically-Aware Methods in GP

The study of semantics in GP has been an active topic since the term was first proposed by McPhee et al. (McPhee, Ohs, and Hutchison, 2008). As an evolutionary method, GP faces the issue that the ‘shapes’ of initial trees can be rapidly lost within a few generations. Traditionally, a diverse initial population, which plays an important role in a successful GP run, has

usually been obtained with the ramped half-and-half method (Koza, 1992) – diversity here has to be interpreted at the syntactic level. Beadle (Beadle, 2009) proposed a semantically-driven initialization algorithm to produce a diverse initial population at the phenotypic level. Compared to the ramped half-and-half method, increased semantic diversity seems to have a positive effect on GP search efficiency. Jackson (Jackson, 2010), however, pointed out that it is not sufficient to ensure semantic diversity only in the initialization stage since a lack of semantic diversity diminishes the exploratory power of GP over the whole run. This author concluded that measures to promote syntactic diversity produced few gains, but those designed to produce semantic diversity generated a noticeable performance improvement. Based on the evaluation of behavioral changes caused by structural modification as a result of mutation, Beadle and Johnson (Beadle and Johnson, 2009) proposed a semantically-driven mutation operator to prevent the creation of new offspring with equivalent performance to that of their parents.

Locality in GP (Galván-López, O'Neill, and Brabazon, 2009) measures the effect of a genotypical change on the phenotype, which is a crucial prerequisite to prevent evolutionary algorithms from behaving as pure random search. Uy et al. (Uy et al., 2010) compared the roles of syntactic and semantic localities of crossover in GP, and pointed out that improving syntactic locality reduced tree size and produced a slight improvement in model generalization. In contrast, improving semantic locality was more effective in reducing tree size and improving model generalization. These authors also proposed a number of semantic-based crossover and mutation operators.

Krawiec et al. (Krawiec and Lichocki, 2009) proposed the approximately geometric crossover, which combined a geometric crossover operator with semantic backpropagation. The semantics were used for guiding the crossover operation during evolution; these operators were further generalized in (Pawlak, Wieloch, and Krawiec, 2015). The recently-proposed RDO and approximately geometric semantic crossover (AGX) operator use semantic backpropagation to identify intermediate subtasks during the evolution process, and then solve these using an exhaustive search method. When compared with other semantic-aware operators and standard genetic operators, RDO and AGX were shown to exhibit improved performance on a series of symbolic regression and boolean benchmark problems. Though generating promising performance, a major weak point of these algorithms is that the child solutions are typically larger than their parents, which may lead to unacceptably slow fitness

evaluations after a few generations. In this thesis, we adopt the semantic backpropagation strategy of (Pawlak, Wieloch, and Krawiec, 2015) for producing the desired output vectors of subtrees to serve as a basis for selecting better-performing replacements.

Geometric Semantic Genetic Programming (GSGP) has aroused the interest of an increasing number of researchers. Moraglio et al. (Moraglio, Krawiec, and Johnson, 2012) introduced a novel set of semantically-aware genetic operators to search the underlying semantic space directly. GSGP, however, has a major shortcoming in that the size of the individuals grows exponentially during the evolution, which makes it impractical for complex, real-life applications. Vanneschi et al. (Vanneschi et al., 2013) overcome this limitation by introducing new, efficient geometric semantic operators. Castelli et al. (Castelli, Silva, and Vanneschi, 2015) proposed geometric semantic genetic operators that enabled them to solve complex, real-world problems efficiently. Moreover, Ruberto et al. (Ruberto et al., 2014) presented a new GP framework by introducing two concepts: optimally aligned, or optimally coplanar, individuals, which outperformed the standard GSGP. Nevertheless, Ruberto et al. omitted problems for which they were unable to find aligned or coplanar individuals and the generalization to unseen data was not clear. Gonçalves et al. (Gonçalves et al., 2016) addressed these questions by using a geometric semantic hill climber to explore the search space.

This work provided a new insight into the relationship between program syntax and semantics, and allows for the principled, formal design of semantic operators for various problems.

3.3.2 Local Search in GP

The combination of local search and evolutionary global search (Giraud-Carrier, 2002; Gruau and Whitley, 1993) has been widely studied and shown to be a powerful strategy for improving search efficiency although this is less commonly employed in GP. Local hill climbing has been integrated into GP either for tuning numerical coefficients (Archetti et al., 2006; Iba, Garis, and Sato, 1994; Krawiec, 2001; Topchy and Punch, 2001; Z-Flores et al., 2014; Zhang and Smart, 2004), or fine-tuning the model structure (Azad and Ryan, 2014; Ffrancon and Schoenauer, 2015; Harries and Smith, 1997; Krasnogor, 2004; La Cava et al., 2015; Majeed, 2007; Wang et al., 2011; Zhang, Gao, and Lou, 2007). Interleaved with global search, the parameters of solutions in each generation have been optimized via: relabeling

(Iba, Garis, and Sato, 1994), genetic local search (Krawiec, 2001), gradient descent (Topchy and Punch, 2001; Zhang and Smart, 2004), and linear scaling (Archetti et al., 2006; Azad and Ryan, 2014).

Many hill climbing local search methods have been embedded in standard GP for model structure optimization. Harries and Smith (Harries and Smith, 1997) proposed a non-evolution based GP with several genetic operators to evolve solutions in a hill climbing manner. Later, a co-evolving memetic algorithm (Krasnogor, 2004) was introduced to produce solutions for the comparison of protein structures by integrating co-evolving local searches with GP. Wang et al. (Wang et al., 2011) optimized decision trees using a splitting operator to divide the whole sample space into subspaces, and then conducted a hill-climbing tuning process. Zhang et al. (Zhang, Gao, and Lou, 2007) introduced the new crossover operator, called looseness control crossover, to find good building blocks by continually crossing over selected parents in a hill climbing manner. Looseness values assigned to each link between adjacent nodes prevent disruption of good building blocks in subsequent operations.

As the traditional crossover operator has often been criticized for being less powerful in forming good offspring solutions, Majeed (Majeed, 2007) proposed a semantic context-aware crossover operator for breeding better child solutions with high fitness gain. This operator identified the best possible crossover point in each selected subtree by examining all possible contexts in which a subtree can be grafted, finally selecting the site where the highest fitness is attained.

Azad and Ryan (Azad and Ryan, 2014) proposed a method to tune the internal nodes of trees one-at-a-time by trying all possible nodes with the same arity, and retaining the modification if a change of node improved the fitness. Although the method demonstrated performance improvements, this is an extreme form of local hill climbing that is unable to modify the 'shape' of a tree.

Since it is only able to explore syntactic space, canonical GP is deficient at determining the (implicit) parameters of a particular program. In order to address this deficiency, Z-Flores et al. (Z-Flores et al., 2014) developed a Lamarckian memetic GP incorporating a local search strategy to optimize parameters embedded in the nodes of the GP trees. These authors concluded that incorporating local search improves convergence and performance while reducing code growth. As with the work in (Azad and Ryan, 2014), the approach of Z-Flores et al. does not use local search to modify the functional form ('shape') of the tree

although whether this approach is effective due to also modifying selective pressures within the population is possible but as yet unexplored. The work in (Z-Flores et al., 2014) was extended in (Juárez-Smith and Trujillo, 2016) by hybridization with the NEAT method.

For combining the exploration ability of semantic GP and the exploitation ability of local search, Castelli et al. (Castelli, Trujillo, and Vanneschi, 2015) integrated semantic mutation operators (Moraglio, Krawiec, and Johnson, 2012) with a local search method for solving a problem in energy consumption forecasting. This case study resulted in good model accuracy with a speeded-up search process. In order to accelerate convergence, Castelli et al. (Castelli et al., 2015) proposed a hybrid algorithm combining GSGP and the above method. The results show this hybrid method allows the search to converge quickly while also exhibiting a noteworthy ability to limit overfitting.

Inspired by the RDO algorithm, Ffrancon and Schoenauer (Ffrancon and Schoenauer, 2015) proposed a local tree improvement operator within a standard local search framework to find the best possible semantic match between all subtrees in a parent tree and all programs in a pre-constructed library. This semantic-aware method performed well on several boolean benchmark problems.

La Cava et al. (La Cava et al., 2015) claimed that the performance of stack-based GP can be improved by embedding local search using epigenetic instructions to specify active and silent genes. In contrast to tree-based GP, stack-based GP is “syntax-free" and syntactic validity is guaranteed no matter how the epigenetic instructions change.

Very recently, Trujillo et al. (Trujillo et al., 2017) have argued that local search is necessary to allow GP to reach its full potential; these authors also note that local search seems comparatively little utilized by the GP community.

3.4 Experimental Methodology

3.4.1 Evolutionary Framework

In the context of empirical modelling using GP, Le et al. (Le et al., 2016) have recently reviewed the use of complexity measures, and point out the critical importance of trading off goodness-of-fit to the training data against model complexity; see also (Nelles, 2001, Chap 7). To explicitly address this trade-off here, we have used a global multiobjective GP formulation in this work with conventional tree-based individuals where the single population

was sorted according to Pareto dominance. We have employed both the sorting approach and selection method of Fonseca and Fleming (Fonseca and Fleming, 1998). We have employed both generational and steady-state evolutionary strategies for 'global' search followed – optionally – by local search over the final populations; we make detailed comparisons below.

Experimental details of the basic evolutionary algorithm are shown in Table 3.1. This, we believe, is a fairly standard configuration except we have used the analytic quotient operator (Ni, Drieberg, and Rockett, 2013) instead of protected division to avoid near-singularities in the solutions. We have employed the straightforward complexity measure of tree node count in our multiobjective formulation since this gives a direct measure of the computational burden of evaluating a tree. The imposition of evolutionary pressure to reduce node count is also an effective way of controlling tree bloat.

Using the normal definition of semantics as the indexed output vector of tree responses over the training data, the semantics of each node within the tree were estimated recursively and stored when it was evaluated for the first time. The calculation of the desired semantics starts from the root node and propagates along all paths to all leaves. Since the desired output of the root node of a tree is known, the desired semantics of each child node in the tree can be calculated assuming that its siblings have the correct structure. If the backpropagation process yields multiple possible values, one is chosen arbitrarily; if the value is undefined, it is ignored in the subsequent calculations of semantic distances between subtrees. See Pawlak et al. (Pawlak, Wieloch, and Krawiec, 2015) for further details.

We have considered the basic evolutionary GP algorithm followed by one of a number of different local search methods; the aim in each case was to reduce the Euclidean distance between the subtree's actual and desired outputs. We investigated a number of strategies for selecting subtrees for replacement that we detail below. Local search has been restricted to the final population in order to keep the computation times within practical limits[1]. In addition, we also include results from the basic GP without local search as a baseline case.

Since it is a prominent example of semantic-based search, we have also included the RDO operator (Pawlak, Wieloch, and Krawiec, 2015) as a comparator. This method uses a library of semantically-unique programs, and when a subtree in a parent is selected during the evolutionary process, a new offspring is generated by replacing the selected subtree with

[1] Additionally, we have observed that, apart from significantly increasing the computation time, applying local search to every generation is ineffective because the conventional evolutionary operators of crossover and mutation are so highly disruptive.

TABLE 3.1: Evolutionary parameters used in this work

Parameter	Value
Population size	100
Initialization method	Ramped half-and-half; maximum tree depth = 6
Number of evolutionary generations	222
Function set	$+$, $-$, $\times$, Analytic quotient (Ni, Drieberg, and Rockett, 2013)
Terminal set	Input variables; constants in 0.1, 0.2..., 0.9
Conventional GP Elitism	Top 10 solutions survive
Conventional GP Operators	Point crossover + point mutation (tree depth <= 4)
RDO-based GP Elitism	None
RDO-based GP Search Operator	Static library (maximum tree depth = 4)
Subtree Selection Method	Equal node probability OR Equal depth probability OR Ito depth-fair selection (Ito, Iba, and Sato, 1998)

the library program exhibiting the closest match to the subtree's desired semantics. (This strategy has the disadvantage that growth in the overall size of the parent tree is not explicitly constrained.) We have used only a static library of trees up to a predefined size limit, precomputed before the evolutionary process commences since this has been shown to yield superior performance to the alternative of a dynamic library (Pawlak, Wieloch, and Krawiec, 2015). Further, we have used more modest library sizes compared to the 100,000 used by Pawlak et al. because we are concerned with the practical application of the method, and therefore its runtime; even with a reduced library size of 1,000, the runtime of the RDO-based method was typically 30 times longer than that of the baseline GP approach. Static libraries were generated with a maximum tree depth of 4, and an initial library size of 1,000 that was then reduced by removing semantic duplicates; typically 5% of the library individuals were removed at this stage. Within RDO, we have explored a range of subtree selection approaches – see Section 3.4.3 for full details. The algorithm settings are shown in Table 3.1.

3.4.2 Local Search Methods

We have applied one of a number of local search approaches to the final population obtained from the baseline GP algorithm. These comprise two key elements: i) the method for selecting a target subtree upon which local search acts, and ii) the method for generating a (potentially) better subtree. Local search was applied to every individual in the final population generated by the baseline GP algorithm. Note that we have not selected a final, single model for evaluation until *after* local search was applied to the whole population. See

Section 3.4.5 below for further details on the numbers of times local search was applied.

3.4.3 Subtree Selection

We have employed three different subtree selection methods in this work.

- Equal Node Probability. Selection where each node in the parent tree is chosen with equal probability to be the root of the target subtree; algorithms using this subtree selection method are denoted with a '1'.
- Equal Depth Probability where the selection method first chooses a depth value in the range zero (i.e. the parent's root node) to the maximum depth of the parent tree, with uniform probability. At this point, one of the nodes at the selected depth is chosen with equal probability. Algorithms using this strategy are denoted with a '2'.
- Ito's Depth-fair Selection. Node selection using the depth-fair selection method of Ito et al. (Ito, Iba, and Sato, 1998). This method is similar to (2) above except that the probability of selecting a given depth halves for every increase in tree depth (subject to the usual normalization condition that the sum of depth selection probabilities is unity). This approach gives nodes at the higher levels of a tree a greater chance of being selected. Algorithms employing this method are denoted with a '3'.

All three methods of subtree selection embody different biases as to how nodes (i.e. target subtrees) are chosen.

3.4.4 Algorithm Comparisons

Clearly a fundamental objective in this work has been to make fair comparisons between some quite different algorithms. To compare the baseline generational GP, steady-state GP and RDO global algorithms is fairly straightforward: we allowed each to run for the same number of local search tree evaluations. This allows each algorithm to make the same number of 'moves' in its search, leading to a reasonable basis for comparison although we restate that the runtime of the generational RDO algorithm with subtree selection method '1' above (GenRDO-1) was typically 30 times longer than for the baseline generational GP (GenGP). Establishing a fair basis for comparison with the various local search algorithms, however, is more problematic. We have addressed this by measuring the process time of the GenRDO-1

algorithm on each benchmark problem, and then limiting the total runtime of one of the local search-based algorithms that uses generational global search followed by generational GP local search with Ito depth-fair selection[2] to this figure. The total number of tree evaluations in this algorithm was noted and used as a limit for all the other local search methods. Local search was continued by cycling over the population, attempting to improve one subtree in every individual per cycle, until the allowed number of local search tree evaluations was exhausted. Thus all algorithms were compared on the basis of being allowed equal amounts of computational 'effort' as gaged by numbers of tree evaluations.

3.4.5 Subtree Generation and Replacement

In conjunction with different methods of subtree selection, we have used a number of different methods to generate candidate subtrees to use as replacements. In all cases, the objective was to generate a replacement subtree with semantics more closely matched to the desired (back-propagated) semantics than those of the original selected subtree:

- Generational GP to Generate New Subtrees. A single objective generational GP was used to search for a tree better matching the desired semantics; apart from the objective function and restricting the local search GP to 100 generations, the evolutionary parameters were as detailed in Table 3.1. A hard limit was placed on the number of tree nodes in the local search GP. This limit on replacement subtree sizes was set equal to the node count of the original target tree to be replaced in order to prevent code growth in the parent. Candidate replacement subtrees were thus, at worst, the same size as the originals they sought to replace. (This is in quite deliberate contrast to the RDO operator (Pawlak, Wieloch, and Krawiec, 2015), which is ambivalent about code growth.) If an evolved subtree had a smaller MSE over the semantic target, it was used to replace the original subtree; otherwise, the parent tree remained unaltered.

- Steady-state GP to Generate New Subtrees. Similarly, a single-objective steady-state GP with hard limit on replacement subtree sizes was applied for tuning subtrees so as to better approximate the desired semantics.

[2] Designated as algorithm 'GenGP-GenGP-3' below.

- Random Generation of New Subtrees. Randomly generating replacement subtrees of the same or smaller node count than the original target subtree; again, this size restriction was designed to prevent growth of the parent tree. For a given parent tree, one cycle of local search comprised first selecting a target subtree, and then randomly generating a sequence of candidate replacement subtrees with randomly-generated node counts less than or equal to the node count of the target subtree. If a candidate subtree produced a closer semantic match than the original subtree, it was immediately used for replacement and the random subtree generation sequence terminated. The number of attempts at replacing a given subtree was limited to a maximum of 100, and if no suitable replacement was generated, the subtree was left unchanged. This search procedure was continued by cycling over the population, attempting to improve a single selected subtree in each individual, until the limit on the number of tree evaluations was reached.

- Using RDO as local search operator to Generate New Subtrees. We have also investigated using RDO as a local search method to improve the final population generated by the baseline global search algorithms – essentially, replacing the local search by random subtree generation with selection of replacements from an RDO-style static library. The RDO operator selects a program that exhibits the closest match to the desired semantics of a selected subtree. We have observed, however, that, when using RDO as a local search method, search over the static library does not necessarily yield a candidate replacement subtree with *better* semantics than the original target subtree. Consequently, we have employed two different criteria for accepting tree modification by a subtree identified from the static library: firstly, we always accept a best-matching candidate subtree ("Best matching subtree"). Second, we only accept a candidate subtree if it both has better-matching semantics to the selected target subtree, *and* the modified tree Pareto-dominates the original parent tree, i.e. it achieves a lower MSE and/or lower node count ("Better matching subtree"). As above, local search cycled over the population attempting to improve one subtree at each pass.

In what follows, we adopt the naming convention for describing a particular experimental configuration of:

- Global multiobjective search paradigm either generational('Gen') or steady-state ('SS').

- The global search method, either GP, or RDO.
- Local single-objective search method: generational('Gen') GP, steady-state ('SS') GP, random tree generation ('Ran'), or RDO ('RDO').
- The method for selecting the subtree for replacement: equal node probability ('1'), equal depth probability ('2'), or Ito's depth fair selection ('3').

Thus, "SSGP-GenGP-2" indicates a steady-state global GP followed by generational GP local search using equal depth probability method of subtree selection. "GenGP" and "SSGP" refer to the baseline global searches with no local search. In addition, for the reasons explained above, we have included two different acceptance strategies when using RDO as a local search operator: "Best matching subtree" and "Better matching subtree". These lead to additional variants, labeled '4', '5' & '6' only for global GP followed by RDO-based local search.

Summaries of the experiments conducted are shown in Table 3.3 for the methods employing generational global search, and in Table 3.4 for methods using steady-state global search.

3.4.6 Test Functions

Although the subject of regression test functions for GP has received detailed consideration (McDermott et al., 2012), we have employed a series of commonly-used benchmark univariate symbolic regression problems – see Table 3.2 – previously used in the GP literature. For each function, we generated 250 independent training sets each containing 20 data uniformly sampled over the domain; the independent test set for each function comprised 10,000 data. The best test MSE obtained from the final population (after any local search procedures) was taken as a measure of generalization performance, this being equivalent to the general procedure in single-objective GP.

3.4.7 Statistical Testing

We have made detailed statistical comparisons of the results obtained. Since we cannot make any distributional assumptions about the results, we have used the nonparametric Friedman test (Demšar, 2006) under the null hypothesis that all the ranks of the results are drawn from

TABLE 3.2: Test functions

Problem	Function	Domain
F1:Automatic French curve (Wahba and Wold, 1975)	$y = 4.26(\exp^{-x} - 4\exp^{-2x} + 3\exp^{-3x})$	[0...3.25]
F2:Sextic polynomial (Uy et al., 2011)	$y = x^6 + x^5 + x^4 + x^3 + x^2 + x$	[-1...+1]
F3:Uy5 (Uy et al., 2011)	$y = \sin x^2 \times \cos x + 1$	[-1...+1]
F4:Uy6 (Uy et al., 2011)	$y = \sin x + \sin(x + x^2)$	[-1...+1]
F5:Vladislavleva (Vladislavleva, Smits, and Hertog, 2009)	$y = 8\exp^{-x} x^3 \cos x \sin x(\cos x \sin^2 x - 1)$	[0...+10]
F6:Chebyshev polynomial (Ni and Rockett, 2015)	$y = 3\cos(3\cos^{-1} x)$	[-1...+1]
F7:Scaled sinc function (Ni and Rockett, 2015)	$y = 5\sin x/x$	(0...+10]
F8:Cubic polynomial (Uy et al., 2011)	$y = x^3 + x^2 + x$	[-1...+1]
F9:Quartic polynomial (Uy et al., 2011)	$y = x^4 + x^3 + x^2 + x$	[-1...+1]
F10:Quintic polynomial (Uy et al., 2011)	$y = x^5 + x^4 + x^3 + x^2 + x$	[-1...+1]
F11:Uy7 (Uy et al., 2011)	$y = \log(x+1) + \log(1+x^2)$	[0...+2]
F12:Uy8 (Uy et al., 2011)	$y = \sqrt{x}$	[0...+4]
F13:Seventh order polynomial (Ni and Rockett, 2015)	$y = 23.7(x+0.9)(x-0.9)(x-0.6)(x-0.6)(x+0.8)$ $(x+0.4)(x+0.3)$	[-1...+1]

the same distribution and therefore there is no difference between the varying treatments; we used the significance level of P $<=$ 0.05 to reject the null hypothesis. When the null hypothesis of the Friedman test was rejected, we used the Holm-Bonferroni post-hoc correction (Demšar, 2006) to the significance level in a Wilcoxon signed ranks test (Benavoli, Corani, and Mangili, 2016; Demšar, 2006) to judge the statistical differences between pairs of results.

3.5 Results & Discussion

Applying all the optimization approaches detailed in Tables 3.3 and 3.4 over each of the thirteen benchmark regression problems F1-13 in Table 3.2, and performing a Friedman test on the ranks of the best MSEs for all algorithms (treatments) and regression problems (subjects) indicated, we reach the conclusion that the null hypothesis – that each of the optimization approaches produces identical results – can be rejected with P-values < 0.0001. There is thus strong evidence of differences between the experimental treatments. For obtaining detailed information on which algorithms are statistically significantly different from each other, we have carried out a series of pairwise tests using the Wilcoxon signed ranks test with a Holm-Bonferroni post-hoc analysis to constrain the family-wise error rates for the multiple comparisons (Demšar, 2006).

Tables 3.5 and 3.6 show the mean ranks of test errors and tree sizes, respectively aggregated over all benchmark problems and treatments.

As a brief introductory overview, from Table 3.5 it is clear that the best-performing algorithm overall is SSGP-SSGP-3 followed by SSGP-GenGP-3. By contrast, the baseline SSGP algorithm ranks 8th overall, and the baseline GenGP algorithm 14th. GenRDO-1 is ranked third along with a number of other algorithms of various configurations. Regarding the significance of the grey-shaded cells in this table, there are no statistical differences between any of the 9th ranked group SSRDO-1 ... GenGP-Ran-3. On the other hand, there is a difference between SSRDO-1 and the 10th ranked GenGP-RDO-1, but no difference between GenGP-RDO-1 and the group SSGP-RDO-1 ... GenGP-Ran-3. We have highlighted this with the grey shading in the 10th column opposite the group SSGP-RDO-1 ... GenGP-Ran-3.

As regards node counts – rankings are shown in Table 3.6 where smaller rank denotes smaller trees – there is a broad inverse relationship between the rankings on test MSE and tree size. Algorithms involving steady-state approaches tend to be associated with larger trees, but tend to have smaller test MSEs. Again in this table, grey-shaded cells denote, for example, that there is no difference between any of the group SSGP-RDO-4 ... SSGP-SSGP-2, and SSGP-GenGP-1.

(In the more detailed discussion that follows, we use the shorthand terms "larger" and "smaller" in the sense of *statistically* larger (or smaller) at the 95% confidence level.)

The principal observations that can be drawn from these results are:

3.5.1 Comparison of Generational and Steady-state Global Strategies Without Local Search

In the absence of any local search, the global steady-state (SSGP) strategy clearly produces smaller test errors than the corresponding generational strategy (GenGP), with mean ranks of 19.585 and 31.820, respectively. The generally superior performance of the steady-state strategy has previously been observed in the context of multiobjective GA by Durillo et al. (Durillo et al., 2009). The average tree size of the models created by SSGP, however, is larger than the average tree size for GenGP strategy with mean ranks of 28.698 and 19.803, respectively. Since we are generally interested in models with smaller test errors and superior generalization, the results here suggest that, in the absence of local search, the steady-state

strategy is better than the much more widely used generational strategy, extending the observations in (Durillo et al., 2009) to another MOEA domain.

TABLE 3.3: Summary of experimental protocols used: Generational global search

	Genetic search operator		Subtree selection method			Local search method				
	standard xover+ mutation	RDO static library	Equal node (1)	Equal depth (2)	Ito depth -fair(3)	GenGP	SSGP	Random generation	RDO	
									Better matching subtree	Best matching subtree
GenGP	✓									
GenRDO-1		✓		✓						
GenRDO-2		✓			✓					
GenRDO-3		✓				✓				
GenGP-GenGP-1	✓		✓			✓				
GenGP-GenGP-2	✓			✓		✓				
GenGP-GenGP-3	✓				✓	✓				
GenGP-SSGP-1	✓		✓				✓			
GenGP-SSGP-2	✓			✓			✓			
GenGP-SSGP-3	✓				✓		✓			
GenGP-Ran-1	✓		✓					✓		
GenGP-Ran-2	✓			✓				✓		
GenGP-Ran-3	✓				✓			✓		
GenGP-RDO-1	✓		✓						✓	
GenGP-RDO-2	✓			✓					✓	
GenGP-RDO-3	✓				✓				✓	
GenGP-RDO-4	✓		✓							✓
GenGP-RDO-5	✓			✓						✓
GenGP-RDO-6	✓				✓					✓

TABLE 3.4: Summary of experimental protocols used: Steady-state global search

	Genetic search operator		Subtree selection method			Local search method				
	standard xover+ mutation	RDO static library	Equal node (1)	Equal depth (2)	Ito depth -fair(3)	GenGP	SSGP	Random generation	RDO	
									Better matching subtree	Best matching subtree
SSGP	✓									
SSRDO-1		✓		✓						
SSRDO-2		✓			✓					
SSRDO-3		✓				✓				
SSGP-GenGP-1	✓		✓			✓				
SSGP-GenGP-2	✓			✓		✓				
SSGP-GenGP-3	✓				✓	✓				
SSGP-SSGP-1	✓		✓				✓			
SSGP-SSGP-2	✓			✓			✓			
SSGP-SSGP-3	✓				✓		✓			
SSGP-Ran-1	✓		✓					✓		
SSGP-Ran-2	✓			✓				✓		
SSGP-Ran-3	✓				✓			✓		
SSGP-RDO-1	✓		✓						✓	
SSGP-RDO-2	✓			✓					✓	
SSGP-RDO-3	✓				✓				✓	
SSGP-RDO-4	✓		✓							✓
SSGP-RDO-5	✓			✓						✓
SSGP-RDO-6	✓				✓					✓

TABLE 3.5: Ranking of the mean squared test errors (MSEs) by algorithm; algorithms listed in the same column display no statistical difference. Conversely, a statistically significant difference is detected between algorithms in different columns. The gray-shaded cells denote that the algorithms shown to their immediate left column have no statistical difference with the GenGP-RDO-1 algorithm in column 10. The rightmost column shows the actual mean rank values.

Overall ranks of MSE values for all the algorithms															
Rank '1'	Rank '2'	Rank '3'	Rank '4'	Rank '5'	Rank '6'	Rank '7'	Rank '8'	Rank '9'	Rank '10'	Rank '11'	Rank '12'	Rank '13'	Rank '14'	Rank '15'	
SSGP-SSGP-3															5.321
	SSGP-GenGP-3														5.807
		GenGP-SSGP-3													7.389
		GenGP-GenGP-3													7.618
		SSGP-SSGP-2													8.567692
		SSGP-GenGP-2													8.752308
		GenRDO-1													9.851538
			GenGP-SSGP-2												11.08538
			SSGP-Ran-2												11.63615
			SSGP-Ran-1												11.97385
			SSGP-Ran-3												13.22385
				SSGP-GenGP-1											14.37231
				SSGP-SSGP-1											14.62154
					GenGP-Ran-1										16.96154
					SSGP-RDO-5										17.44308
					SSGP-RDO-4										17.46
					SSGP-RDO-6										18.29231
						GenGP-Ran-2									19.11231
							SSGP								19.58538
								SSRDO-1							20.47154
								SSGP-RDO-1							20.49
								GenGP-GenGP-2							20.51615
								GenGP-GenGP-1							21.00615
								GenGP-SSGP-1							21.56
								GenGP-Ran-3							21.62462
									GenGP-RDO-1						22.13923
										GenGP-RDO-4					25.48154
											GenRDO-2				26.14077
												GenGP-RDO-5			26.22538
													SSRDO-2		27.05846
													GenGP-RDO-2		27.47385
													SSGP-RDO-2		27.57308
													GenGP-RDO-6		28.64077
													SSRDO-3		31.67308
													GenGP		31.82
														GenRDO-3	33.98308
														GenGP-RDO-3	34.01308
														SSGP-RDO-3	34.03538

TABLE 3.6: Overall ranking of node counts by algorithm; algorithms listed in the same column display no statistical difference. Conversely, a statistically significant difference is detected between algorithms in different columns. The gray-shaded cells denote that algorithms to their immediate left show no statistical difference to the algorithms in that column. The rightmost column shows the actual mean rank values.

Overall ranks of tree sizes for all algorithms														
Rank '1'	Rank '2'	Rank '3'	Rank '4'	Rank '5'	Rank '6'	Rank '7'	Rank '8'	Rank '9'	Rank '10'	Rank '11'	Rank '12'	Rank '13'	Rank '14'	
SSRDO-3														6.314
	GenGP-Ran3													8.910
	GenRDO-3													8.919
		SSRDO-2												9.394
		GenGP-Ran-2												9.512
		GenGP-RDO-3												9.629
		SSGP-RDO-3												9.695
			GenGP-Ran-1											12.315
				SSRDO-1										14.881
				GenGP-RDO-6										15.365
				GenGP-RDO-5										15.467
					SSGP-Ran-2									15.634
					GenGP-RDO-4									15.989
					SSGP-Ran-1									16.026
						GenRDO-2								16.473
						GenGP-RDO-2								17.292
						SSGP-RDO-2								17.544
							SSGP-Ran-3							18.315
							GenGP							19.803
							GenGP-GenGP-2							20.369
							GenGP-SSGP-1							20.567
							GenGP-GenGP-1							21.477
							GenGP-SSGP-2							21.741
							GenGP-GenGP-3							22.775
							GenGP-SSGP-3							22.935
								SSGP-RDO-5						23.783
								SSGP-RDO-4						24.149
								SSGP-GenGP-2						24.863
								SSGP-SSGP-2						25.014
									SSGP-GenGP-1					25.172
										SSGP-RDO-6				25.234
										SSGP-GenGP-3				25.363
										SSGP-SSGP-3				25.463
										SSGP-SSGP-1				25.677
											SSGP			28.699
												GenRDO-1		31.23692
													GenGP-RDO-1	34.411
													SSGP-RDO-1	34.595

3.5.2 Influence of the Global Search Strategy on the Efficacy of a Given Local Search Method

Following on from the previous observation, we can examine the influence of the evolutionary global search strategies on local search methods. It is clear from Table 3.5 that a given local search algorithm following a steady-state global search performs better than the corresponding algorithm that uses generational GP local search, except for the three pairs: SSGP-RDO-1 vs. GenGP-RDO-1, SSGP-RDO-2 vs. GenGP-RDO-2, SSGP-RDO-3 vs. GenGP-RDO-3, between which no statistically significant differences were detected. (It is noteworthy that all six algorithms in this 'no difference' category use RDO as the local search method; we observe below that RDO does not appear to be particularly good as a a local search technique. Thus it seems likely that these six algorithms are not representative results.) Since the starting point for all local search is the final population produced by the global search strategy, there seems strong evidence that the generally superior population produced by the steady-state strategy facilitates more productive local search, regardless of the local search algorithm employed. It seems logical that starting from a 'better initial position' will help the subsequent local search to find superior solutions.

At the same time, comparing the average tree sizes generated by the various algorithms, the trees generated by generational global search are statistically smaller on a like-for-like basis than those created by a steady-state GP, again except for the three pairings listed above for which no statistically significant differences can be detected. As pointed out above, however, if presented with this trade-off, most practitioners would favour the methods yielding the smaller generalization errors.

3.5.3 Comparing RDO in Generational and Steady-state Global Strategies

Algorithms using RDO as the genetic operator exhibit different performances when used with generational compared to steady-state evolutionary strategies. Compared to the baseline GenGP, the RDO genetic operator used in a generational strategy yields performances that range from the seventh best performer (GenRDO-1) via a middle-ranking performer (GenRDO-2) to rapid deterioration to one of the worst algorithms (GenRDO-3). GenRDO-3 performs even worse than the baseline GenGP due to the fact that the population in these runs invariably collapsed to a single or small number of identical individuals, thereby dramatically damaging the searching ability of the algorithm due to lack of diversity.

- The Role of Evolutionary Strategy with Global RDO

  The general performance of RDO as a search operator in a steady-state strategy, however, shows a great difference. The average test errors of all the SSRDO algorithms are statistically worse than those of the baseline SSGP. The inference is that subtree replacement from the randomly-initialized static library harms the search efficiency of a steady-state GP. A possible reason for this might be that the RDO operator, which replaces selected subtrees with specific randomly-generated library programs, induces significant disruption during a steady-state evolution process. The evolution process of a generational GP is itself highly disruptive since the majority of chromosomes in each new generation are produced through crossover operations; in this circumstance, the RDO operator appears to improve the search efficiency and generates more accurate trees than the baseline GenGP. The steady-state strategy, however, relies on a continual advancement towards the Pareto front that RDO seems to repeatedly disrupt leading to poor overall search performance.

- The Role of Subtree Selection Strategy with Global RDO

  From the perspective of the subtree selection approach used with RDO, for both generational and steady-state strategies, algorithms selecting subtrees with equal node probability generate more accurate models than those using the equal depth selection method. Ito's depth fair subtree selection method produces the worst results. This suggests that the performance of the RDO operator is sensitive to the method of selecting subtrees.

3.5.4 The Role of the Generational and Steady-state Strategies for Local Search

From Table 3.5, clearly the SSGP-SSGP-3 is the best performer among all the algorithms. Unlike the previous observation that the global search ability of a steady-state GP is always better than a generational GP, when GP is used as a local search operator, the steady-state GP does not exhibit any consistent advantage over the generational GP.

When compared by subtree selection methods, however, algorithms using Ito's depth fair method produce the most accurate models. Algorithms selecting subtrees with equal node probabilities are ranked lowest among all the GP-based local search algorithms. This suggests that a subtree with a shorter path to the root node of its parent tree is likely to be more

influential on the entire tree in the overall evaluation; this conclusion is consistent with a hypothesis proposed by Igel et al. (Igel and Chellapilla, 1999). To verify this, we investigated the relationships between success rate and MSE reduction with the normalized depth of selected subtrees. The normalized depth of each subtree is calculated by dividing the depth of a selected subtree from the root node by the full depth of the whole tree. All the selected samples were divided into ten groups according to their normalized depth with increments of 0.1. A illustrative group of five of the above thirteen benchmark functions were used, and the corresponding graphs of the relationships between successful subtree replacement rate (Figure 3.1) and MSE reduction (Figure 3.2) with normalized depth of selected subtrees.

From the graphs in Figure 3.1, the success rates are roughly constant with increasing normalized depth values, which shows that subtrees of different normalized depths have almost identical probabilities of being successfully replaced. This suggests that the good performance of algorithms using Ito's depth fair subtree selection method is not caused by more frequent modification of subtrees near the root node of a GP tree. From the graphs in Figure 3.2, an inverse relationship between the magnitudes of MSE reduction with the normalized depth of selected subtrees can be observed. This implies that a more efficient optimization of GP trees can be achieved by selecting subtrees with shorter path to the root node. In other words, an improvement of subtrees near the root node is more likely to have a larger beneficial effect on the whole tree. We consider this the main reason that causes good performances of algorithms using Ito's depth fair selection method.

3.5.5 Influence of the Number of Cycles of Local Search

Whether it is possible to achieve comparable results with fewer generations of global GP search and/or less effort on the local search is of great practical interest. Taking the best performing SSGP-SSGP-3 algorithm as an example, we conducted an experiment to further explore the balance between these factors. Typically, the CPU runtime of one local search cycle over all the trees in a final population takes $\sim$12 seconds (on a given computer), which is far longer than that of the baseline steady-state global search (SSGP) lasting $\sim$4 seconds. Thus by far the greatest proportion of the computational effort is spent on the local search process. The influence of the numbers of local search cycles on the model accuracy is presented for five representative test functions Figure 3.3.

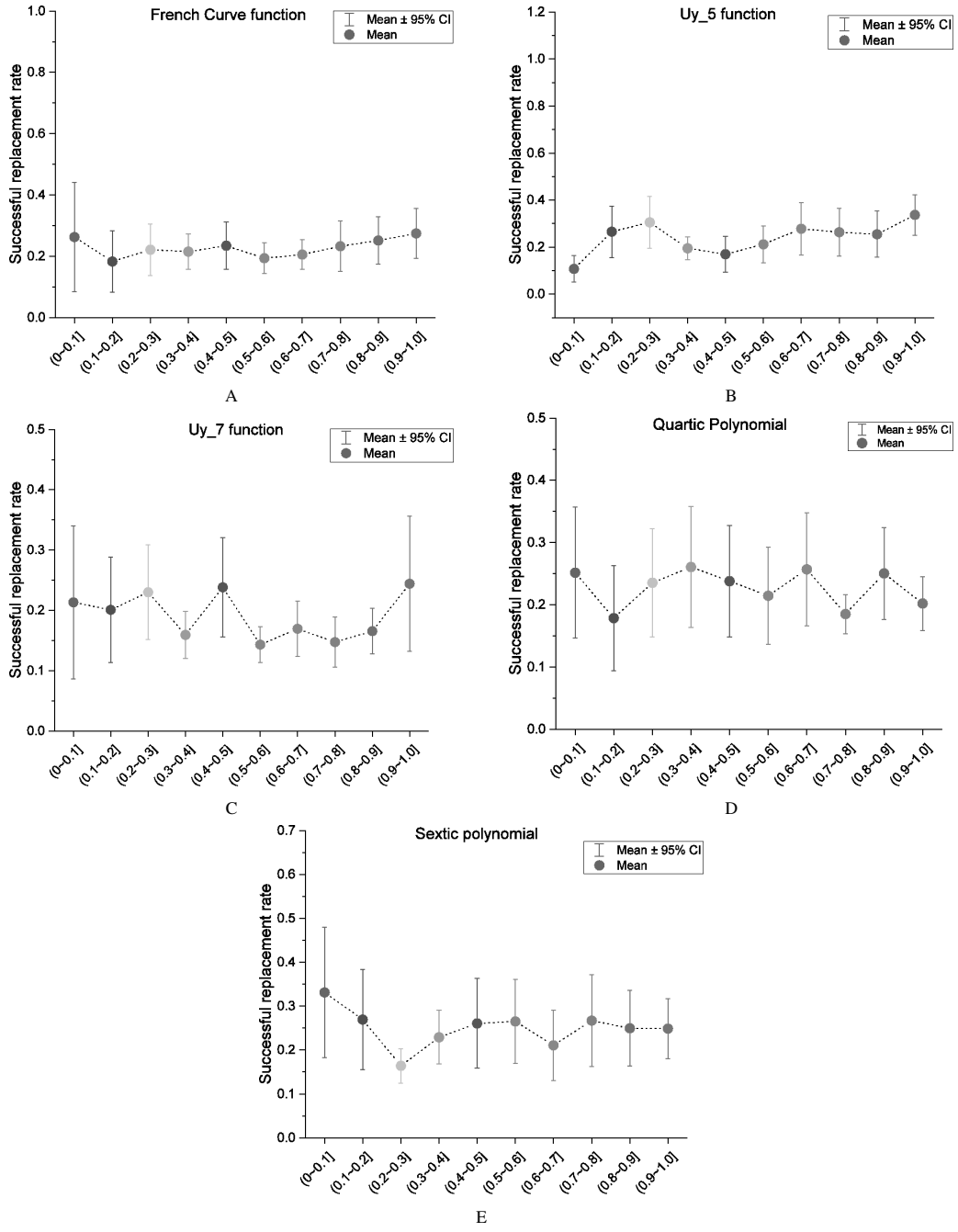

FIGURE 3.1: Relationship between successful replacement rate with normalized subtree depth over five benchmark functions.

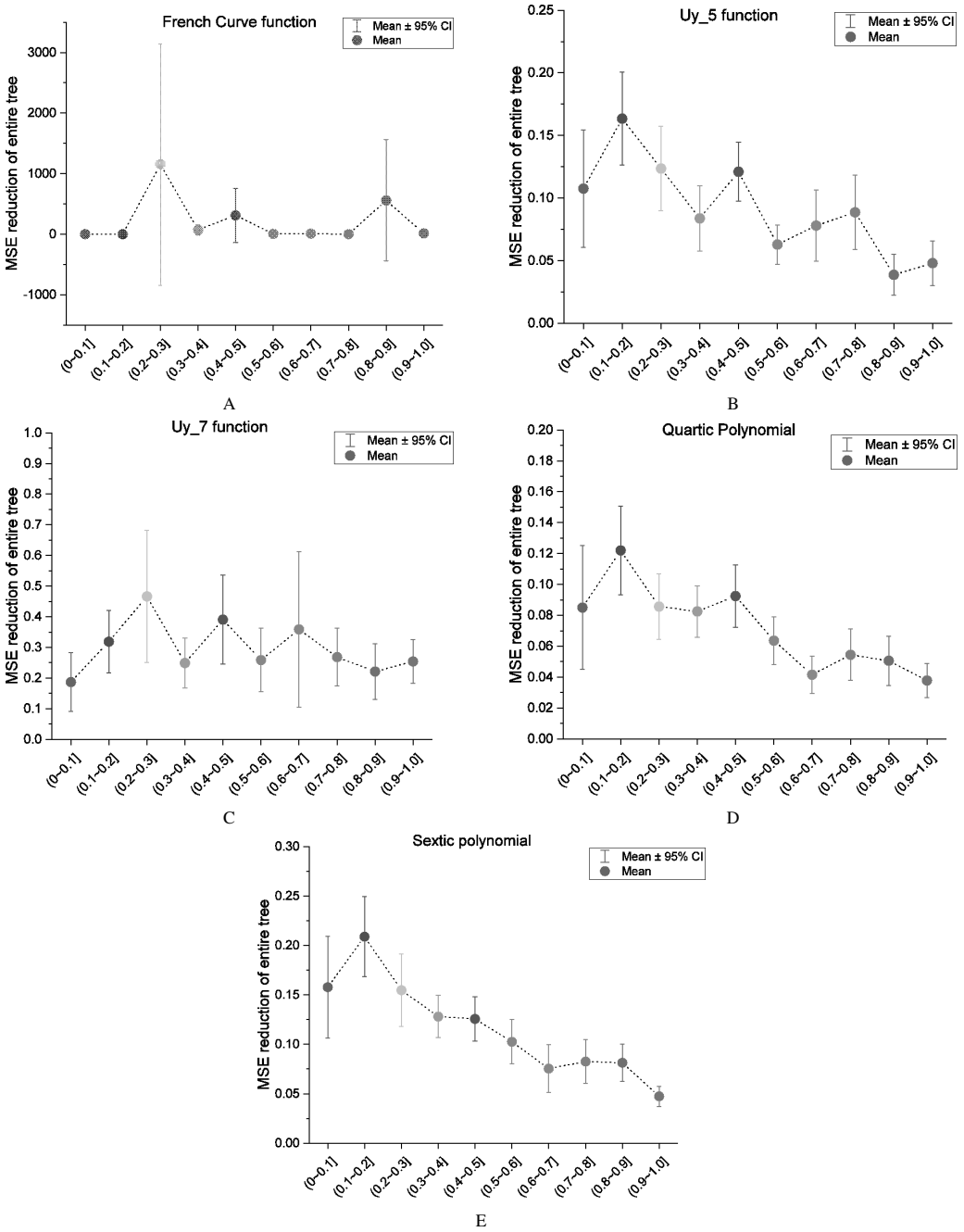

FIGURE 3.2: Relationship between MSE reduction with normalized subtree depth over five benchmark functions.

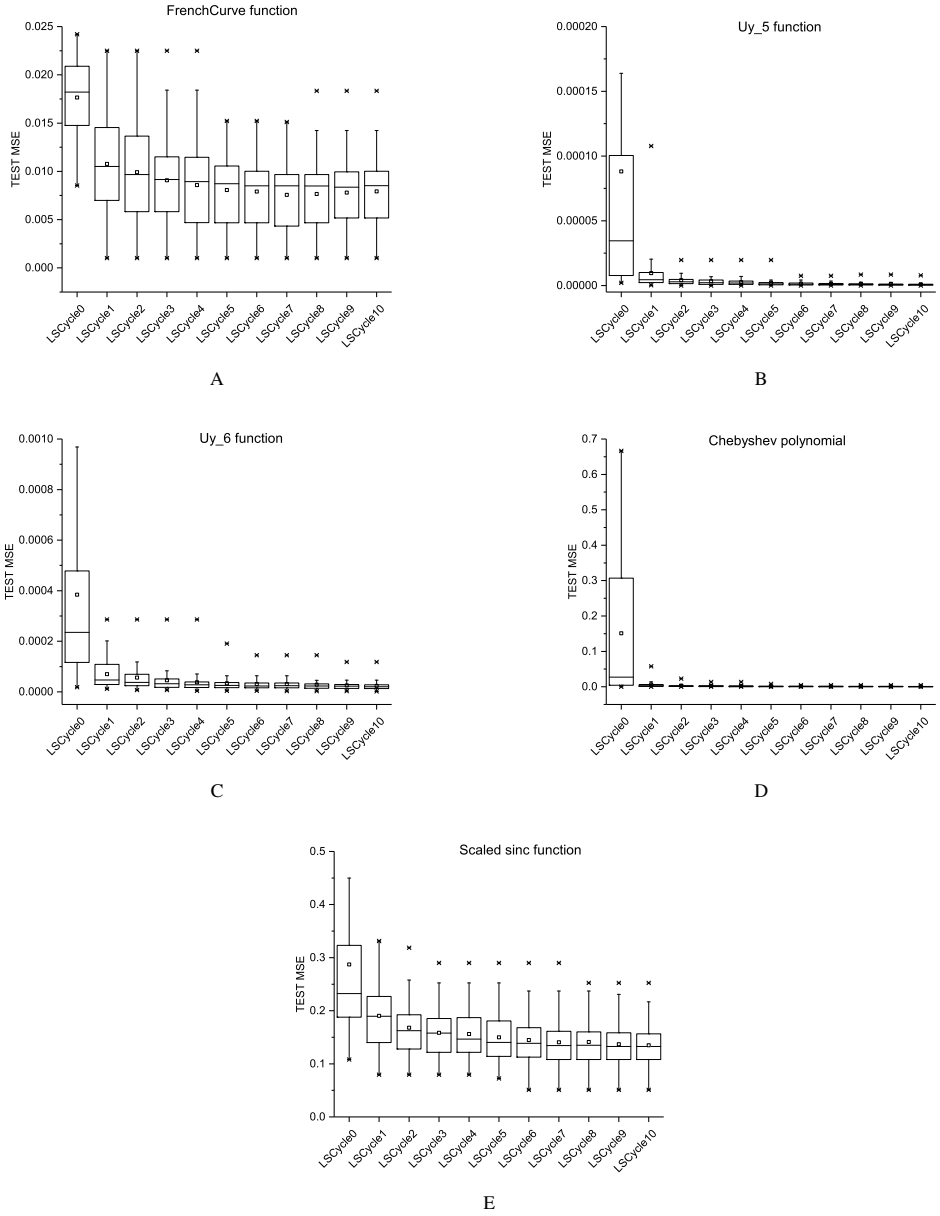

FIGURE 3.3: Relationship between test MSE and the number of local search cycles of SSGP-SSGP-3 over five benchmark functions; the number following 'LSCycle' denotes the number of local search cycles in the local search process.

From Figure3.3, it is clear that the test error reduces with increasing numbers of local search cycles. This reduction, however, slows significantly after 2 or 3 cycles of local search. In a sense, this is very welcome since local search is so time consuming – it appears that only a little local search is needed beyond which the benefits diminish rapidly.

3.5.6 Influence of Local Search on Expected Tree Sizes

Considering the tree size comparisons in Table 3.6, all the evolutionary local-search methods based on either steady-state or generational global search produce trees that are either smaller or statistically the same size as the trees produced by their corresponding baseline algorithms. Thus, for example, local search following SSGP tends to produce smaller trees than SSGP without local search.

Intriguingly, the observation that local search tends to *reduce* tree sizes seems counterintuitive given that the local search methods were designed only to prevent code growth, not to produce more parsimonious structures – see Section 3.4.5.

For a given parent tree in the final population, we observed that local GP search almost invariably reduced the size of the tree – namely, GP local search seems effective at finding smaller trees better matched to the desired backpropagated subtree semantics. Now the final population generated by the baseline algorithm comprises an (approximation to) the Pareto set of equivalent solutions ranging from underfitted solutions with high training MSE/few nodes through to overfitted solutions with small training MSE/large numbers of nodes; as a rough rule, the solution yielding the best test MSE tends to lie around the middle of the Pareto front. Although it tends to shrink the size of the trees, we observe that local GP search rarely improves the test error of the best-performing individual produced by the baseline GP such that it continues to be the best-performing individual after local search terminates. Rather, one of the overfitted individuals tends to be modified in the local search procedure and is promoted to having a better test error than the best individual produced by the global search method. This reinforces the approach of applying local search to the *whole* of the final population of the global search algorithm rather then just the best-performing individual produced by the global algorithm. Recently, Trujillo et al. (Trujillo et al., 2017) have made a similar observation for local search in the context of single-objective GP. More generally,

the same sort of phenomenon has been previously seen in decision trees, which are typically trained to overfitting and then heuristically pruned to improve generalization (Quinlan, 1993).

3.5.7 Performance of Random Subtree Generation as a Local Search Operator

The overall performances of the algorithms that use random tree search is variable. The SSGP-Ran-1,2,3 algorithms are all 4th ranked for MSE whereas the performances of GenGP-Ran-1,2,3 are more varied: the first two are better than SSGP, the last worse than SSGP but on a par with GenGP-GenGP-1,2 and GenGP-SSGP-1. The superior performance of the random subtree replacement algorithms that use SSGP as a global search algorithm is presumably connected to the general superiority of the steady-state strategy in global search.

Superficially, at least, there appears a similarity between local search by randomly generating replacement subtrees (the GenGP-Ran-1 ... GenGP-Ran-3 family of algorithms) and the RDO method. RDO constructs a large library of randomly-generated subtrees from which one is chosen to replace a target subtree in the parent. This generation-by-lookup table process could be viewed as an alternative way of randomly generating a subtree. GenRDO-1, however, is statistically better than random search implying this approach is not equivalent to random local search following global GP; at this point, nonetheless, we sound a note of caution about the size effect observed here. The reason for the apparent superiority of RDO-based methods is not completely clear and will be the subject of future work.

3.5.8 The Performance of RDO as a Local Search Operator

We have also investigated using RDO as a local search method (SS/Gen-RDO-1 to SS/Gen-RDO-6) to improve the final population generated by the baseline SS/GenGP algorithms – essentially, replacing the local search by random subtree generation with selection of replacements from an RDO-style static library. Again, superficially, these could be seen as equivalent processes. The results of using RDO for local search were overwhelmingly negative with little improvement in the population generated by the corresponding global SS/GenGP algorithm. We conclude, therefore, that RDO functions poorly as a local search method although clearly performs well as a genetic operator (in the generational strategy). Its superficial resemblance to a random local search operator would thus appear coincidental.

3.5.9 RDO Compared to Global GP + GP Local Search

One of the principal findings of this work is that using GP as a local search procedure is able to produce generalization performance that is better than the state-of-the-art GenRDO approach, and does so using trees of significantly smaller sizes; this observation applies to all of the thirteen test functions considered. To take a typical example, GenRDO-1 with the French curve function produced best test-error tree sizes in the range 29 to 723 with an average of 196. The SSGP-SSGP-3 algorithm, on the other hand, yielded trees of 23 to 467 nodes with an average of 135. We believe this results from the careful implementation of the GP local search method to avoid code growth – see Section 3.4.5.

The RDO genetic operator exhibits good search ability in a generational strategy, but with a steady-state strategy, the RDO operator performs even worse than the baseline SSGP algorithm. This implies the RDO search operator is sensitive to the evolutionary strategy. Moreover, the rapid performance deterioration from GenRDO-1 to GenRDO-3 indicates the RDO genetic operator is also sensitive to the subtree selection method. This is a disappointing characteristic of RDO since evolutionary methods are generally considered to be very robust to sub-optimal choices of parameters, etc. This robustness does not appear to extend to the RDO approach. On the contrary, GP local search appears much less sensitive than RDO to a different choice of subtree selection method. The use of Ito's method that prefers selecting target subtrees near the root node seems to encourage model generalization of the entire tree.

3.5.10 Computational Complexity Resulting from Different Local Search Strategies

We have also considered the additional computation resulting from various local search methods. Taking the French curve function as as example, and experiments run on a computer with a 3.40 GHz processor. The average CPU runtime for the baseline GenGP is around 0.68 seconds, and for the baseline SSGP around 3.83 seconds. Table 3.7 lists the average CPU runtime cost for one cycle of refinement using various local search methods. From this table, it is clear that the most time-consuming local search is SSGP-3 that also produces the most accurate models. Local search algorithms using RDO turn out to be the least time-consuming, but provide minimal improvement to (and are sometimes worse than) the corresponding baseline GP. Clearly, the SSGP local search strategies consume more time than their corresponding

GenGP methods. For both SSGP and GenGP local search, one cycle of optimization of the final population using Ito's depth fair subtree selection method takes longer than the equal-depth selection method, and the equal node subtree selection method the least. Generally, local search using random subtree generation takes about 7 seconds.

TABLE 3.7: CPU runtimes for one cycle of local search on the French curve function

Local search algorithm	GenGP-1	GenGP-2	GenGP-3	SSGP-1	SSGP-2	SSGP-3
runtime (seconds)	3.483	6.942	14.219	39.150	68.942	95.990
Local search algorithm	Ran-1	Ran-2	Ran-3	RDO-1	RDO-2	RDO-3
runtime (seconds)	7.994	6.749	7.112	2.841	2.558	2.539
Local search algorithm	RDO-4	RDO-5	RDO-6			
runtime (seconds)	2.561	2.599	2.571			

3.6 Discussion and Future Work

The work described in the thesis has been deliberately constrained to local search methods that change the 'shapes' of GP trees by altering sub-trees. Successful local search has also been reported using methods that introduce additional 'tuning' parameters into the tree nodes – for example, (Trujillo et al., 2017). An obvious area for future work is a quantitative comparison between these different approaches to local search, or indeed possible hybridization between them.

Although this thesis presents a large range of algorithms, methods of local search, and their combinations, much future work remains to be done. In carrying out the work reported here, we have deliberately adopted a 'breadth first' philosophy rather than seeking detailed explanations for every observation. That said, a very clear and fertile area for future work is to revisit the promising research directions that we have identified to gain a fuller understanding of the phenomena involved; in our experience, such studies tend to be time-consuming hence we have deferred them to future work.

Another area that warrants further study is the extension to more complex, higher dimensional test functions. In the present work, we have employed the univariate functions that tend to be regarded as "standard" within the GP community. While they represent a valid

starting point for a study, these functions have received some criticism and other, more challenging datasets have been proposed in the literature (McDermott et al., 2012). An important research issue is to establish whether the advantages of local search identified in the present thesis extend to higher dimensions. In addition, explicitly considering real-world datasets – which often present different challenges – would be a major extension of this work.

On the subject of test functions, one of the reviewers of our paper (Dou and Rockett, 2018) suggested that 'genomic'-type datasets – characterized by hundreds or thousands of features but only tens of records – would be an appropriate subject for study in the present work; such challenging datasets have recently been addressed by Chen et al. (Chen, Zhang, and Xue, 2017) using GP. In our view, the main research issues when applying GP to 'genomic' datasets are twofold: firstly, to constrain the complexity of a GP model to prevent overfitting when learning in what are effectively 'empty' pattern spaces, bearing in mind that one of the major advantages of GP is its ability to automatically adjust its own complexity. Secondly, genomic-type datasets are typically characterized by the presence of large numbers of uninformative/redundant features. In the context of such challenging learning problems, we think there is little reason that local search on its own would have much impact on datasets with these characteristics without also explicitly addressing the complexity constraint and feature selection challenges.

A further area that might warrant additional investigation is the mechanism of semantic back propagation that is the precursor for local search. In common with other reports, we have adopted the strategy of back propagating errors from the root node of a tree under the assumption that all of a given node's siblings possess the correct structure. Although a reasonable simplification, this would seem to significantly constrain the scope of any local search. In this context, we suggest a sensitivity-based approach (Saltelli et al., 2004) may improve search efficiency, and this too will be the subject of future work.

We have pointed out in Section 3.2 that memetic algorithms combine global exploratory search with local exploitative search. We believe our work fits very much within this paradigm. Since subtrees for replacement by local search are stochastically chosen, it is possible that consecutive passes of local search over a parent tree will select exactly the same target subtree leading to inefficient, duplicated search. Our use of Ito's selection strategy ('3') that tends to prefer subtrees rooted near the top of the parent will exacerbate this effect since there are fewer choices near the tree's root. We suggest improving the efficiency of our method with a

tabu-like approach whereby subtrees that have been subjected to local search are not then immediately re-subjected to it in the next pass of local search. This could easily be implemented by tagging nodes with a timestamp of when they are selected as targets, and examining this timestamp before proceeding with local search; this is an area for future work.

Finally, we note that GP has proved an extremely effective and practical technique for solving the combinatorial optimization problem of searching over a set of possible functions. Local search over the set of possible subtrees in a parent GP tree could thus be viewed as a recursive reduction of the overall problem. In light of this, it is perhaps logical that GP should perform well as a local search strategy.

3.7 Conclusions

The most significant conclusion from this work is that semantic-based GP local search is able to produce better generalization performance that is statistically different from the state of the art GenRDO-1 method of Pawlak et al. (Pawlak, Wieloch, and Krawiec, 2015), and achieves this with trees of significantly smaller size. This has obvious computational implications. A contributory factor to this reduction in tree size has been the careful design of the local search procedure so as to avoid tree growth. We observe that our GP local search seems to operate by pruning overfitted trees down to the point of best test performance rather than necessarily improving the best test case individuals generated by the global SS/GenGP algorithms. Trees generated by the (SS/GenGP)-(SS/GenGP)-3 approach tend to be (statistically) smaller than those generated by the corresponding baseline algorithms, while at the same time exhibiting better prediction performance.

We have also found that the RDO operator was obviously effective when used as a genetic operator within the generational paradigm. The performance of RDO with a steady-state strategy, however, is noticeably worse than with a generational strategy. The trees generated by the steady-state variants of RDO are less accurate than even those generated by the baseline SSGP algorithm. We infer that the disruption caused by RDO search counteracts the otherwise good search performance of the steady-state strategy.

Additionally, we observed significant effects of the method for selecting the subtree for local search. On the basis of the work here, the RDO operator appears sensitive to the

choice of selection operator, yielding performance that ranges from the seventh best performer (GenRDO-1) via the 12th-ranked performer (GenRDO-2) to population collapse and the lowly-ranked performer (GenRDO-3). GP local search, on the other hand, appears to display far less sensitivity to the choice of subtree selection method. The SSGP-SSGP-3 method ranks top while a less helpful choice of subtree selection method only reduces this form of GP local search to a middling (3rd or 5th ranked) performer rather than a bottom-ranked performer. The reason that the Ito's depth fair selection method performs best was investigated and it was concluded that the optimization of subtrees closer to the root are more influential in the improvement of the entire tree. Furthermore, the test error reduces with increasing numbers of cycles of local search although the gains appear modest after only two or three cycles of local search.

Chapter 4

GPML

We propose a GPML, an XML-based standard for the interchange of GP trees, and outline the benefits such a format would bring. We present a formal definition of this standard and describe details of an implementation. In addition, we present a case study where GPML is used to implement a model predictive controller for the control of a building heating plant.

4.1 Author's Contribution

1. GPML was proposed and implemented. Details about how to interpret a GP model into a GPML format was given. Additionally, how to parse and validate a GPML file were also described.

2. In terms of reliably sharing research finding in the GP community, GPML allows accurate interchange and calculation of GP models without repeating others' work often with incomplete information.

3. In terms of the ease of use, GPML can be used as a 'plug-in' component in larger applications, which improves usage efficiency and providing application modularity.

4.2 Introduction

Replicability is a cornerstone of the scientific method that facilitates the independent validation of research findings. Conversely, independent researchers being unable to replicate findings is a means by which erroneous research is rapidly purged from the scientific literature.

A great many of the studies published in evolutionary computing, including GP, are – perforce – both empirical and stochastic, and this imposes a particular difficulty from the point of view of replication. Addressing the GP community in particular, it has become the universal practice to include algorithm parameters such as population size, crossover and mutation rates, etc. in all publications. Although this is welcome, it generally does not give complete information to allow accurate replication of the experiments being reported. For example, complete reporting would require details of the random number generator algorithms employed, their seed values and other details characterizing the stochastic experiments. It could be argued that an adequate number of repetitions of the experiments 'average out' stochastic effects, but the generality of this claim is far from clear.

Adopting standard software for *all* GP might also be considered to address the replication issue, but we would argue that *independent* implementation is an important aspect of replication. Any bug in the 'standard' software might go undetected for many years, hindering progress in the field. There are cautionary tales here from other disciplines. For example, it has been reported that 1 in 5 of recent genetics research papers contain errors due to formatting problems in the Microsoft Excel spreadsheet program (Washington Post, 2016).

The purpose of this work is to propose a standardised format for the interchange of GP trees. A similar initiative has already been taken on benchmarking problems (White et al., 2013). As a complement to agreed benchmarks, a standard interchange format would offer a number of significant advantages:

1. The publication of the *actual GP trees*, as opposed to a (often partial) description of how the trees were produced, would speed scientific progress in the area.

2. The ability to make direct comparison with the results of other researchers would speed scientific progress, and eliminate the time, frustration and practical difficulties of trying to reproduce others' results, often with incomplete information.

3. Ultimately, the objective is for GP trees to be used as components in larger, complex systems. The coupling of the end-application and the GP training/test software is a major inconvenience, especially in embedded systems. A standard interchange format would allow trained/validated GP trees to be treated as a 'plug-in' component in larger systems.

Since journals, and an increasing number of conferences, provide repositories for supplementary material, it would be straightforward for authors to deposit, and thereby archive, data fully describing *actual* GP trees that could be accessed by other researchers.

Fortuitously, a suitable and mature framework for developing a GP interchange format already exists: the eXtensible Markup Language (XML) defined by the World Wide Web Consortium (World Wide Web Consortium, 2008). In essence, this work proposes a standard interchange format based on XML and XML Schemas, an XML validation framework. This will make it possible for researchers to share the *actual* trees generated during their research thereby exposing them to greater scrutiny/validation by the community as a whole. This should facilitate more rapid progress since researchers will not need to duplicate each other's work in order to make quantitative comparisons with alternative approaches and methods. In addition, GP trees could be treated as a 'plug-in' element for more complex systems, such as robotics; decoupling the training/testing from an application of trained GP trees will have a beneficial effect on the real-world deployment and exploitation of GP.

In Section 4.3, we briefly describe the relevant features of XML and its accompanying validation framework, XML Schemas. We describe an XML representation of a GP tree in Section 4.4, and an example implementation in Section 4.5. Section 4.6 concludes this chapter.

4.3 Extensible Markup Language (XML)

The XML is a standardized, highly-flexible, human-readable format for information exchange specified by the World Wide Web Consortium (W3C) (World Wide Web Consortium, 2008). Being a text-based representation, XML avoids the hardware-dependent difficulties of binary files. XML itself comprises a series of elements and attributes arranged in a hierarchical fashion – the intrinsic hierarchy in XML lends itself perfectly to describing GP trees, which are, of course, typically represented as acyclic hierarchical graphs. Further, the syntax of XML is very intuitive although human readability is probably a secondary consideration here since we envisage trees being written/read mainly by computer. Nonetheless, the ability to visually inspect the tree structure is valuable, and often requested by paper reviewers. (We give some simple examples of GP trees represented as XML in Section 4.4.)

In XML, an element is specified by a syntax such as:

```
<elementName option='A'>
...
</elementName>
```

where `elementName` and `option` are user-defined identifiers, and `A` is a quoted text string. Alternatively, an equivalent single-line version where the element does not nest other elements is:

```
<elementName option='A'/>
```

Crucially for the present application, XML elements can also embed other XML elements, as in:

```
<elementName option='A'>
    <otherElementName...>
    </otherElementName>
    <thirdElementName...>
    </thirdElementName>
...
</elementName>
```

and so on to arbitrary levels of nesting. XML can thus straightforwardly represent hierarchical structures. See (Castro, 2001) for a concise but comprehensive introduction to XML.

In a given application, an XML file has to conform to a specified structure. The need for validating the structure of XML files has led to the development of XML Schemas (Vlist, 2002) (World Wide Web Consortium, 2012) for this purpose. XML Schemas – themselves XML-compliant – are able to specify an XML file structure using a syntax reminiscent of the extended Backus Naur format (EBNF) widely used for specifying the grammar of programming languages; importantly for the present application, XML Schemas are able to specify recursive structures for validation.

In terms of implementation, a large number of proprietary and mature open source XML libraries are available, for example, Xerces (*Apache Xerces Project*), with bindings to a range of programming languages. In addition, many XML implementations for Matlab are available (e.g. (*XML Documents*)). Consequently, there seems no technical impediment to adopting XML as an interchange format.

We term the interchange format proposed here a GPML to denote its GP-specialisation over plain XML.

To date, XML has found little application in the GP community. In a rare example, Tanev and Shimohara (Tanev and Shimohara, 2010) have used the Document Object Model (DOM) (WHATWG, 2018), a representation of an XML structure in memory, directly for GP evolution although the DOM is probably not ideal for this purpose.

4.4 GPML Specification

In this section, we describe GPML, a standardised XML interchange format.

Making use of the intrinsic hierarchy, GPML recursively encodes the tree structure. Different types of nodes and their corresponding information are identified and recorded in various elements and attributes in GPML. By recursively interpreting the element name and attribute information of each node starting from the root node down, the GP tree can be saved and restored.

With elements and attributes arranged in a nested, hierarchical fashion, GPML can be used to describe GP trees intuitively. Typically, the nodes in GP trees can be classified as one of four types: terminal nodes, unary nodes, binary nodes and ternary nodes. These four different nodes correspond to four elements in GPML, namely, `terminalNode`, `constantNode`, `unaryNode`, `binaryNode` and `ternaryNode`, with user-specified information described in the respective attributes.

For the purpose of illustration, consider the simple GP tree shown in Figure 4.1 that represents the mapping $y = f(\mathbf{x})$ where $y \in \mathbb{R}$ and $\mathbf{x} \in \mathbb{R}^N$ (although there is no restriction with GPML on the input/output being real numbers, or indeed that the elements of $\mathbf{x}$ are even of the same type). This mapping can be straightforwardly represented by the GPML code shown in Listing 4.1. The intuitively obvious correspondences should be apparent.

In Figure 4.1, terminal nodes in the GP tree indicate elements of the input vector. We have used the `vectorIndex` attributes of the `terminalNode` elements as indices into the input vector $\mathbf{x}$. For constant nodes, the exact values are specified in the attributes in GPML.

In GP, each unary node has a single child node, which could be any type of node, including another unary node type. In GPML, a unary node is represented by an element `unaryNode`, and its operator is specified by its attribute. Similarly, binary nodes and ternary

nodes have two and three child nodes, respectively. In GPML, node operators are specified by their attributes, like the nodes shown in Figure 4.1 and their GPML representations in Listing 4.1.

Note that each GPML document has exactly one root element, which encloses all the other elements. In GPML, the element name of the XML root node is `gpTree`. The attribute of the root node, `noVectorElements`, defines the number of regressors used in the mapping, which can be exploited for validation of a GPML document. The indices of terminal nodes in GPML are restricted to the ranges of either $[0 \ldots (\texttt{noVectorElements} - 1)]$ or $[1 \ldots \texttt{noVectorElements}]$, depending on the (imple- mentation-defined) convention adopted for indexing vectors in the actual implementation. Consequently, the attribute `firstIndex` $\in \{\text{'0'}|\text{'1'}\}$ unambiguously associates tree inputs with elements in the input vector. The value of `vectorIndex` in a `terminalNode` element needs to fall in the appropriate range otherwise a validation error occurs that can be straightforwardly detected and notified to the user. It is also a trivial matter to use a tree that has been trained in a system with zero-index vectors in a (separate) system using vectors indexed from unity, and vice versa.

Although the example shown here is for a conventional GP tree, GPML is sufficiently expressive that it could be adapted to other GP variants, such as gene expression programming (Ferreira, 2002), linear GP (Brameier and Banzhaf, 2010), or Cartesian GP (Miller, 2011).

LISTING 4.1: GPML representation of the tree in Figure 4.1.

```
<?xml version='1.0' ?>
<gpTree noVectorElements='2' firstIndex='1'>
    <binaryNode operation='+'>
        <unaryNode operation='-'>
            <constantNode value='0.2'>
        </unaryNode>
        <binaryNode operation='*'>
            <terminalNode vectorIndex='1'/>
            <terminalNode vectorIndex='2'/>
        </binaryNode>
    </binaryNode>
</gpTree>
```

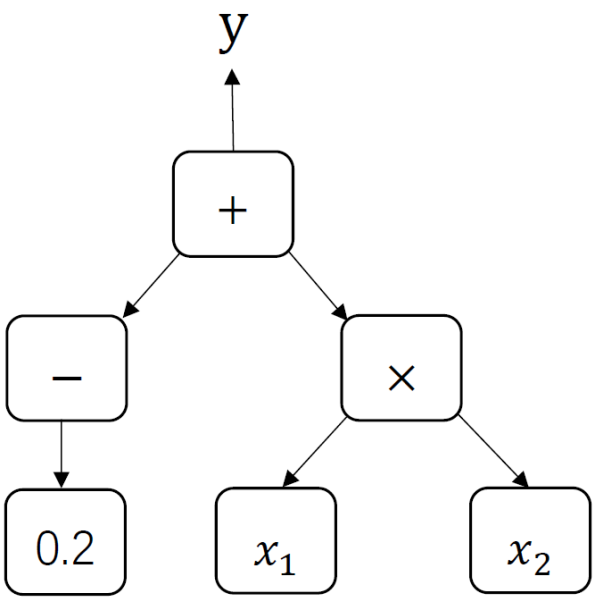

FIGURE 4.1: Simple example GP tree.

In terms of extended Backus-Naur form (ISO/IEC 14977:1996, 1996) typically used to specify programming languages, we can formally describe the topmost-level `gpTree` element of GPML using:

gp-tree = "<gpTree",
"noVectorElements=", *xml-positive-integer*,
"firstIndex=", "0" | "1" ">",
node,
"</gpTree>";

where the terminal symbol `noVectorElements` is an XML-defined primitive of positive integer type (World Wide Web Consortium, 2012), i.e. a number $\in \mathbb{N}^+$ since the GP mapping is presumed to have at least one input. The non-terminal symbol *xml-positive-integer* value indicates the number of elements in the vector of input variables for the mapping described by the GP tree. As discussed above, the `firstIndex` attribute denotes whether the described GP tree indexs the input vector starting from zero or one.

The non-terminal symbol *node* is defined as by:

node = *terminal-node* | *constant-node* | *unary-node* | *binary-node* | *ternary-node*;

where a *node* is one of either: a terminal node, a constant node, a unary node, a binary node or a ternary node. In their turn, the set of node types, which we believe covers the practical totality of GP, are formally defined by:

terminal-node = "<terminalNode", "vectorIndex=", *xml-non-negative-integer*,
"/>";

where *xml-non-negative-integer* is again an XML-defined primitive type for an integer quantity ≥ 0 (i.e. $\in \mathbb{N}^0$), and:

constant-node = "<constantNode", "value=", *xml-double*, "/>";

where *xml-double* is an XML primitive denoting a double-precision floating-point number.

A unary node is defined by EBNF as:

unary-node = "<unaryNode", "operation=", *operation*,
["parameterString=", *parameter-string*,]
">",
node,
"</unaryNode>";

in which *operation* is an XML-defined token type (i.e. a character string), and *parameter-String* is an optional XML token. The `operation` element indicates the (implementation dependent) operation executed by the unary node; for a unary minus, for example, this might be the string '`-`', or for an exponential function '`exp`'. For the cases where a unary node may take some (arbitrary number of additional) parameters, these can be specified using the (optional) '`parameterString`' by concatenating all the function's parameters in, say, a comma-separated list. These parameters can then be simply 'unpacked' by the implementation code. This method of passing any additional parameters is a carefully considered design decision for GPML, striking a balance between simplicity, generality and clarity of GPML syntax. For example, the GP induction of a decision tree (Cao and Rockett, 2015) would typically contain (only) binary nodes that test the state of an element in the input vector and follow the left or right child subtrees, respectively depending on whether the given input vector element was $<$ than or $\geq$ than some decision threshold. For this application, one possible implementation would be for a node's threshold value to be stored in the optional parameter attribute and extracted by the implementation code.

The *binary-node* and *ternary-node* types in GPML are defined by:

binary-node = "<binaryNode", "operation=", *operation*,
["parameterString=", *parameter-string*,]
">",
node,
node,
"</binaryNode>";

ternary-node = "<ternaryNode", "operation=", *operation*,
["parameterString=", *parameter-string*,]
">",
node,
node,
node,
"</ternaryNode>";

Yet again, *operation* indicates the operation to be performed by the binary or ternary nodes, respectively, and the optional *parameter-string* conveys any additional parameters required.

The unary, binary or ternary operations specified by the above node types are deliberately left undefined by GPML, and are implementation-dependent. Typically, in a regression problem, the unary operation will be one of: unary minus, exponential function, sine function, etc. Similarly, the binary operations implemented will be one of: addition, subtraction, multiplication, (protected) division, or analytic quotient (Ni, Drieberg, and Rockett, 2013), while the ternary operation will typically be `if-then-else`. When learning a boolean problem, on the other hand, the only unary operator would typically be the `NOT` operation, and the binary operations would comprise: `AND`, `OR`, `XOR`, etc. This facility means that GPML can be expanded to include arbitrary, domain-specific operations.

Note also how a `binaryNode`, for example, 'embeds' two *node* types, each of which is defined as being one of a: `terminalNode`, `constantNode`, `unaryNode`, (another) `binaryNode` or `ternaryNode`. The GPML syntax is thus able to recursively define a GP tree of unbounded extent. Similarly, a `gp-tree` 'embeds' a single *node* implying a single root node for the parent tree.

A user is, of course, free to add XML-compliant comments to a GPML file since these will be subsequently ignored by any XML parser.

4.5 Implementation

Quite deliberately, we do not specify or indeed restrict implementation details for GPML. In this section, we describe our initial implementation as a point of reference.

4.5.1 Writing GPML

Given a (trained) GP tree in memory, writing a valid GPML file involves a fairly straightforward recursive descent of the tree. With reference to Listing 4.1, the first task is to output the preamble of the GPML file that comprises the '`gpTree`' information, the '`noVectorElements`' attribute, and the closing '`>`' character. At this stage, the writing procedure recursively descends the tree (in whatever form this has been implemented) and on 'entering' a node,

it emits the appropriate GPML element definition (‘<terminalNode’, ‘<constantNode’, ‘<unaryNode’, ‘<binaryNode’ or ‘<ternaryNode’). Then:

- If the GP node is a terminal (either a ‘<terminalNode’ or ‘<constantNode’) then it remains only to emit either the ‘vectorIndex’ or ‘value’ attribute, respectively and a ‘/>’ element terminating sequence, and then to return from the recursive call.
- If the current node is a non-terminal GP node, the ‘operation’ attribute field together with the optional ‘parameterString’ field are emitted, and then the appropriate number of further recursive calls made to visit the child node(s) of the current node. On return from the last of these recursive calls, the function needs to emit a terminating ‘</unaryNode>’, ‘</binaryNode>’ or ‘</ternaryNode>’ field, and then return.

When the chain of recursive calls finally ends, the only remaining task is to emit the ‘</gpTree>’ terminating field.

4.5.2 Reading GPML

We have implemented the initial GPML system using the lightweight pugixml XML library[1], largely for simplicity and convenience. The pugixml library reads the specified XML file into memory as a DOM (WHATWG, 2018), a hierarchical structure that can be traversed using functions built into the XML library. Implementation in terms of a DOM is not the only possible approach; the Simple API for XML model is equally viable and possibly faster in execution although its use tends to be more involved. Having created a DOM of the tree, it is a straightforward matter to traverse this data structure, creating and linking GP nodes in an implementation-dependent manner.

4.5.3 Validating GPML

One of the important elements presented in this work is an XML Schema for the validation of GPML. The simple pugixml library we have used does not provide validating facilities although other XML libraries, for example, Xerces (*Apache Xerces Project*) do, however, these tend to be more involved to use. In practice, a range of other, convenient validation mechanisms are feasible, for example: the xmllint[2] command-line validator, which is part

[1] http://pugixml.org/
[2] http://xmlsoft.org/xmllint.html

of the `libxml` library . Alternatively, the open-source `jEdit`[3] text editor has an XML plugin that performs validation against a specified schema.

We have made the XML Schema for GPML freely available under a Gnu Public Licence (GPL-3) on the GitHub repository (https://github.com/pirlite2/gpml-schema).

4.6 Conclusions

In scientific research, reproducibility and transparency are of the utmost importance since science can only progress when the research results produced by one investigator can be independently corroborated by other researchers. The nature of GP research is empirical and stochastic, which imposes particular difficulties on reproducing research findings in the field. We have proposed an XML-based standard for the interchange of GP trees in the hope of speeding scientific progress in the area. GP trees specified in GPML can be evaluated and compared directly by researchers without the effort of having to reproduce others' trees, often with incomplete information.

In terms of implementation, due to its hierarchical structure, GPML can be flexibly represented in XML for which a number of mature, open source XML libraries are available. We have further proposed an XML Schema for the validation of GPML, which is available under a GPL licence at:

https://github.com/pirlite2/gpml-schema

In addition, in larger systems, a trained-and-validated GP tree can be embedded as a 'plug-in' component using GPML, providing convenience and modularity. We have demonstrated this capability by developing a dynamical model for a single-zone building model and using this model to successfully implement MPC of the building's internal environment in Section 5.4 and Section 5.5. This case study illustrates the practical flexibility of GPML as an interchange format.

[3] http://www.jedit.org/

Chapter 5

Model Predictive Control of Nonlinear Dynamic System Using GP

We present a novel approach to obtaining dynamic nonlinear models using GP for the MPC of buildings. Currently, the large-scale adoption of MPC in buildings is economically unviable due to the time and cost involved in the design and tuning of predictive models by expert control engineers. We have shown that GP is able to automate this process. We performed open-loop system identification over the data produced by an industry grade building simulator. The simulated building was subjected to an amplitude modulated pseudo-random binary sequence (APRBS), which allows the collected data to be sufficiently informative to capture the underlying system dynamics under relevant operating conditions.

We detail how we employed GP to construct the predictive model for MPC, and report results of using this model for controlling a simulated building. We observed that GP can produce models that allow the MPC of building to achieve the desired temperature band in a single zone space.

5.1 Author's Contribution

1. A novel GP-based MPC scheme was proposed for controlling thermal dynamics in a simulated test building. In the building MPC literature, this is the first time for building MPC has used GP models as controllers.

2. Based on data collected from the open-loop control process on a simulated test building, GP was used for identifying the thermal dynamics of the test building[1].

[1] In this work, Yuri Kaszubowski Lopes, one of our research team, contributed to the open-loop data generation of a simulated building and generated the MPC results.

3. The obtained GP model was translated into GPML format and used for the MPC.

4. The performance of GP-based building MPC was analyzed. GP was witnessed to be able to maintain the room temperature of the test building within ± 1 C of the desired setpoint.

5.2 Introduction

MPC (Camacho and Bordons, 2004) is a powerful control methodology well-suited to systems in which there is an appreciable delay between an input being applied and any observable response, and which may also have control constraints; (non-domestic) buildings are among such systems. Central to MPC is a predictive model of the dynamics of the system being controlled. Given a prediction horizon extending some number of discrete time steps into the future, the controller optimises the sequence of future inputs by minimising some objective function. Typically, this objective comprises a weighted sum over the prediction horizon of the deviations from a desired setpoint and the control effort, the magnitudes of the control changes. This latter term is designed to penalise rapid switching of the input and hence minimise actuator wear. At every time step, the future input sequence is optimised, the first of this input sequence applied to the system and the whole process repeated at the next time step. This forever advancing prediction interval gives the technique its alternative name of receding horizon control.

Although MPC has been widely employed in the chemical process industries, where it had its origins, applications to buildings are currently only at the research stage – see, for example, Rockett and Hathway (Rockett and Hathway, 2017) for a review. Critical to MPC, whatever its domain of application, is the performance of the predictive model.

The generally superior control of MPC in buildings compared to conventional rule-based approaches appears to offer significant energy savings – maybe up to 25% (Rockett and Hathway, 2017) – and make buildings MPC worth pursuing in order to reduce CO_2 emissions and improve internal environmental quality. However, at a roundtable discussion at a workshop on MPC in buildings held in Montréal in 2011, Henze (Henze, 2013) noted attendees estimated 70% of total costs for MPC implementation were consumed by the creation and calibration of the predictive model that lies at the heart of MPC. In fact, this figure agrees with

the 75% often quoted by the wider process-control community (Hussain, 1999). Traditionally, such models are produced by extensive fine-tuning by highly skilled control engineers. Although the high cost of predictive model creation may be tolerable in the highly-capital intensive environment of petrochemicals, Rockett and Hathway (Rockett and Hathway, 2017) have pointed out that such high costs currently make MPC economically unviable for the control of buildings. It is, therefore, critical for the economic uptake of MPC in buildings to create predictive models of the system dynamics using machine learning-based methods that can learn from data obtained from the building in operation rather than be hand-crafted by experts. Further, the characteristics of buildings change over time, either due to changes in use, internal alterations, or indeed external factors, such as the erection/demolition of adjacent buildings that change the solar gains or façade wind pressures on the building under control. Such changes will change the dynamics of the building and necessitate a recalibration of the predictive model in order to maintain optimised control. Rapid and low-cost recalibration is thus also essential to maximise the ongoing benefits of MPC in buildings.

Buildings are widely acknowledged to exhibit non-linear dynamics and therefore require a non-linear predictive model. The problem of formulating such a model has been discussed in a seminal paper by Sjöberg et al. (Sjöberg et al., 1995). Assuming sampling at discrete, equally-spaced time steps, the one-step-ahead prediction $\hat{y}_{k+1}$ of a dynamical system at time $(k+1)$ is given by:

$$\hat{y}_{k+1} = f(\mathbf{u}_k, \mathbf{u}_{k-1}, \ldots, \mathbf{u}_{k-n}, y_k, y_{k-1}, \ldots, y_{k-m}) \tag{5.1}$$

where $\mathbf{u}$ is a vector of input, or so-called *exogenous*, variables. The problem is to identify i) f, the non-linear function, ii) the value of n dictating how many of the previous inputs need to be considered, and iii) the value of m, the number of previous (autoregressive) outputs that need to be included. The sets of delayed variables $\{\mathbf{u}_k, \mathbf{u}_{k-1}, \ldots, \mathbf{u}_{k-n}\}$ and $\{y_k, y_{k-1}, \ldots, y_{k-m}\}$ are usually termed *lag sets*. To implement MPC we generally need a model that produces a set of accurate future predictions over the so-called *prediction horizon*, that is, N time steps into the future.

In principal, the search for f in (5.1) is over the set of all possible functions, but in practice f is often restricted to families, such as Volterra functions or neural networks (Nelles, 2001). Identification of the lag sets (i.e. the best combination of values of n and m) is typically

performed iteratively in a manner highly dependent on the expertise of a control engineer.

ANNs have been widely used for nonlinear dynamic system identification. In order to enhance the accuracy while minimising the model size, an architectural refinement stage is often required. For instance, NEAT (Stanley and Miikkulainen, 2002) uses a GA to evolve both model structure and the associated parameters of neural network models.

A further consideration with Volterra approximators and, especially, neural networks is the large number of parameters that have to be estimated during training, which implies a requirement for a large amount of training data. Moreover, with reference to (5.1), while training ANNs can approximate the function f, determining the lag sets specified by n and m usually requires the embedding of the NN training within some global search for the network inputs determined by n, m, the so-called *feature selection* problem.

To address the problem specified by (5.1), an increasing number of researchers have applied GP to the nonlinear dynamic systems identification problems (Grosman and Lewin, 2002; Feng, Lian, and Zhu, 2017) due to the advantage of being able to automatically optimize both model structure and its parameters during evolution. Basic GP, however, is often used to evolve the function f either as a simple regression problem (i.e. without the autoregressive terms $y_k, y_{k-1}, \ldots, y_{k-m}$), or using pre-defined lags sets, that is, pre-specification of n and m in (5.1).

Grosman and Lewin (Grosman and Lewin, 2002) used GP to learn the relationship between inputs and outputs of a mixing tank system. The performance of control based on the evolved solution was then assessed. They concluded that the GP based predictive controller provided significantly better regulatory and servo performance than the traditional proportional integral controller and internal model controller. Recently, Feng et al. (Feng, Lian, and Zhu, 2017) also investigated the performance of GP on non-linear dynamical systems and NMPC, and claimed that satisfactory performance of NMPC can be obtained based on GP models.

In the model training stage, however, both Grosman and Lewin (Grosman and Lewin, 2002), and Feng et al. (Feng, Lian, and Zhu, 2017) employed user-specified lag sets, which are normally very time-consuming to determine manually in practical applications.

Hinchliffe and Willis's (Hinchliffe and Willis, 2003) also used GP to evolve discrete-time models of dynamic processes, however, evolution of the appropriate lag set of input variables was included by adding unary back-shift operators to the GP's function set. The

experimental results suggest that the performance of GP shows little difference with filter-based neural networks in terms of model accuracy on an extruder case study. The significant point in Hinchliffe and Willis' (Hinchliffe and Willis, 2003) work is that their GP formulation is not only able to approximate model structure (f), but also construct appropriate lag sets and not require their pre-specification.

Taking advantage of the fact that the Hinchliffe and Willis GP scheme is able to evolve both model structure and lag sets automatically during the evolution process, in this work, we describe the use of GP for creating the dynamic model necessary for buildings MPC. We believe this to be the first report of the demonstration of buildings MPC using learned GP models. As is common in the control field, we have considered a system simulation in order to rapidly and comprehensively explore the issues involved.

In Section 5.3, we describe GP for modelling dynamical systems and give an example for a benchmark problem from the chemical engineering literature. We describe the building control methodology we have used in Section 5.4 together with the procedures necessary for successfully identifying a predictive GP model of the test building. In Section 5.5 we report typical results of the performance of the predictive GP model as well as the performance of the MPC scheme. In this work, we present only representative, typical results and defer detailed discussion of parameter settings, etc. to future work. We do, however, discuss these issues in Section 5.6. We conclude this work with Section 5.7.

5.3 Genetic Programming for Dynamic System Identification

As the name implies, the states of a dynamic system change over time. The essence of dynamical system identification is to develop mathematical models that describe the relationships between all the available inputs and outputs. The basic methodology of GP has been detailed in Section 2.2. Particularly, in order to enable GP to approximate dynamic systems, backshift (or delay) operators (Δ_1,Δ_2,Δ_3) and auto-regressive nodes are added to the function set and terminal set, respectively.

Unit time delay operator is introduced by defining a new type of the unary node Δ_1 such that given variable y_k measured at the k-th time step, the value of the regressor at time step

$(k-1)$ is given by:

$$\Delta_1(y_k) \rightarrow y_{k-1} \tag{5.2}$$

That is, the unit delay operator Δ_1 returns the value of the same variable at time step $(k-1)$. In fact, following the work of Hinchliffe and Willis (Hinchliffe and Willis, 2003), we define time delay operators for one, two and three unit delays. Δ_1 is defined in (5.2) above, $\Delta_2(y_k) \rightarrow y_{k-2}$, and $\Delta_3(y_k) \rightarrow y_{k-3}$. GP evolution is then able concatenate these delay operators to produce delays longer than three time steps, if required.

These delay operations can be straightforwardly incorporated in GPML as:

```
<unaryNode operation='delay1'> ... </unaryNode>
```

i.e. defining a new operation for a time lag of one unit, and so on for lags of two (`delay2`) and three (`delay3`) units of time. These delayed regressors can.of course, be evaluated from previous outputs of the GP tree.

A typical example of a dynamic GP tree is shown in Figure 5.1, and its functional expression is given in

$$y(k) = (-(\Delta_1(y_{k-1}))) + (0.2 * (-u_{k-1})). \tag{5.3}$$

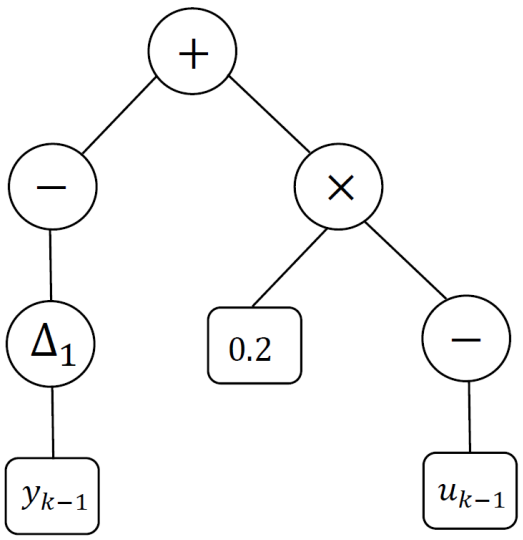

FIGURE 5.1: Example GP tree representing a simple dynamic system.

5.3.1 Identification of the Eaton-Rawlings Reactor

There have been many reports in recent years exploiting the potential of GP for system identification, particularly in chemical engineering applications (Vyas, Goel, and Tambe, 2015). In order to assess the suitability of GP for nonlinear dynamic system identification, one-step-head prediction of a well-known benchmark chemical process, the Eaton-Rawlings reactor model, was investigated. The Eaton-Rawlings reactor model (Pearson, 1999) describes a second-order reaction occurring in an isothermal continuous stirred-tank reactor. The dynamics of this reactor are expressed by the first-order, nonlinear ordinary differential equation

$$\frac{dy}{dt} = -hy^2 - \frac{yu}{V} + \frac{du}{V} \tag{5.4}$$

where y is the concentration in the continuous stirred-tank reactor, h is the kinetic rate constant for the reaction, V is the reactor volume, and d is the inlet concentration of the reactant. The manipulated variable u is the inlet flow rate.

If the manipulated input u is assumed to change only at regular sampling instants t_k, an exact discretisation is easily derived from the continuous-time equation; see (Pearson, 1999) for full details.

$$y(k) = \frac{[1-\tau(k-1)\mu(k-1)]y(k-1) + 2d\tau(k-1)\mu(k-1)}{1+\tau(k-1)[y(k-1)+\mu(k-1)]} \tag{5.5}$$

where

$$\tau(k-1) = \frac{\tanh\left[hT\sqrt{\mu^2(k-1)+2d\mu(k-1)}\right]}{\sqrt{\mu^2(k-1)+2d\mu(k-1)}} \tag{5.6}$$

$$\mu(k-1) = \frac{u(k-1)}{2hV} \tag{5.7}$$

In this experiment, h was 1.50 litre/mole-hr, V was 10.51 litre, d was 3.5 mole/litre. The inputs u_k were a sequence of steps of uniformly-distributed random amplitudes ranging from 0.5 to 5.0 litres per hour with a switching probability of 1.0. This input sequence was used to perturb the reactor model (5.5).

To facilitate direct comparison with previous, conventional modelling approaches, we

have followed the procedure in Pearson (Pearson, 1999) and generated 100 statistically-independent noise-free training sequences, each of length $P = 200$. The reactor responses to these input sequences were calculated according to the discretisation formula (5.5).

The performance of a GP solution was ranked by two objectives: tree size and the MSE. The parameter settings for the GP evolution are described in Table 5.1.

In each GP experiment, a set of solutions was obtained after training from which the candidate with the smallest MSE over a statistically-independent noise-free validation dataset was finally selected as the best model for this run. Thus, after 100 independent training processes, the best GP model with the smallest validation MSE among the selected 100 trees was picked as the final best solution. The best GP model selected had a validation MSE of 0.000121458.

TABLE 5.1: Evolutionary parameters used for the Eaton-Rawlings reactor system identification

Parameter	**Value**
Population size	100
Evolution strategy	Steady state
Initialization method	Ramped half-and-half
Maximum tree depth in initialization	6
Maximum number of tree evaluations	20000
Function set	$+$, $-$, $\times$, Analytic quotient (Ni, Drieberg, and Rockett, 2013), Δ_1,Δ_2,Δ_3
Terminal set	Input variables; constants in range $\{0.1, 0.2, \ldots, 0.9, 1.0\}$
Crossover frequency	1.0
Mutation frequency	1.0
Fitness measures	Tree size and MSE
Selection method	Pareto ranking

We made quantitative comparison with the model – a NARMAX – that exhibited the best performance compared to other hand-tuned model structures and lags studied by Pearson (Pearson, 1999); the same training datasets were used to train the NARMAX models by minimising the mean squared error metric (5.9) using the `NLopt` nonlinear optimization library[2]. The best NARMAX model is given by:

$$y(k) = y_0 + \alpha\, y(k-1) + \beta\, u(k-1) + \gamma u(k-1)\, y(k-1) \tag{5.8}$$

[2] https://nlopt.readthedocs.io/en/latest/

where y_0, α, β, and γ are unknown parameters to be determined by minimizing the objective function, and Q is the length of the training sequence:

$$J(y_0, \alpha, \beta, \gamma) = \sum_{k=1}^{Q-1} [\hat{y}(k+1) - y(k+1)]^2 \tag{5.9}$$

where $\hat{y}(k+1)$ is the one step ahead predicted value at time k, and the $y(k+1)$ is the measured value at time $k+1$.

The best validation MSE of the NARMAX models was 0.000120801, nearly equal to the value of obtained from the best GP tree. In the best NARMAX model, the parameter y_0 is 0.298058, α is 0.145929, β is 0.117087 and γ is -0.0188017. The residuals of the best NARMAX and GP models over the corresponding validation sets are shown in Figure 5.2 from which it can be seen that the GP tree exhibits comparable model accuracy to the best NARMAX model. The residuals of the NARMAX model show a number of negative spikes lower than -0.02, with the absolute value of the biggest residual around 0.04. The GP model shows two significant positive spikes, but with most of the residuals lying in a small range around zero.

The encouraging approximation ability of the GP model on the benchmark Eaton-Rawlings problem suggests that GP is suitable for more general nonlinear dynamic system identification problems. Particularly, GP does not require the functional form of the model to be pre-specified – rather, this evolves during training. This advantage makes GP a potential technique for identifying a wide range of real world, nonlinear dynamic systems for which the underlying physical principles are not known.

5.4 Building Control Methodology

In this section, we describe the procedures employed for the MPC of buildings using predictive models obtained through a GP-based system identification. Figure 5.3 depicts the components of the simulation system used in this work. The MPC framework used here was assembled by Dr. Yuri Kaszubowski Lopes. We used an industry-grade simulator for building physics, described in Section 5.4.1, to simulate the responses of a test building. This simulator provides a standardised interface that allows the interconnection of external software units, and we used this facility for two separate tasks: first, for the open-loop collection of system identification (SID) data detailed in Section 5.4.3, and second for the simulation of

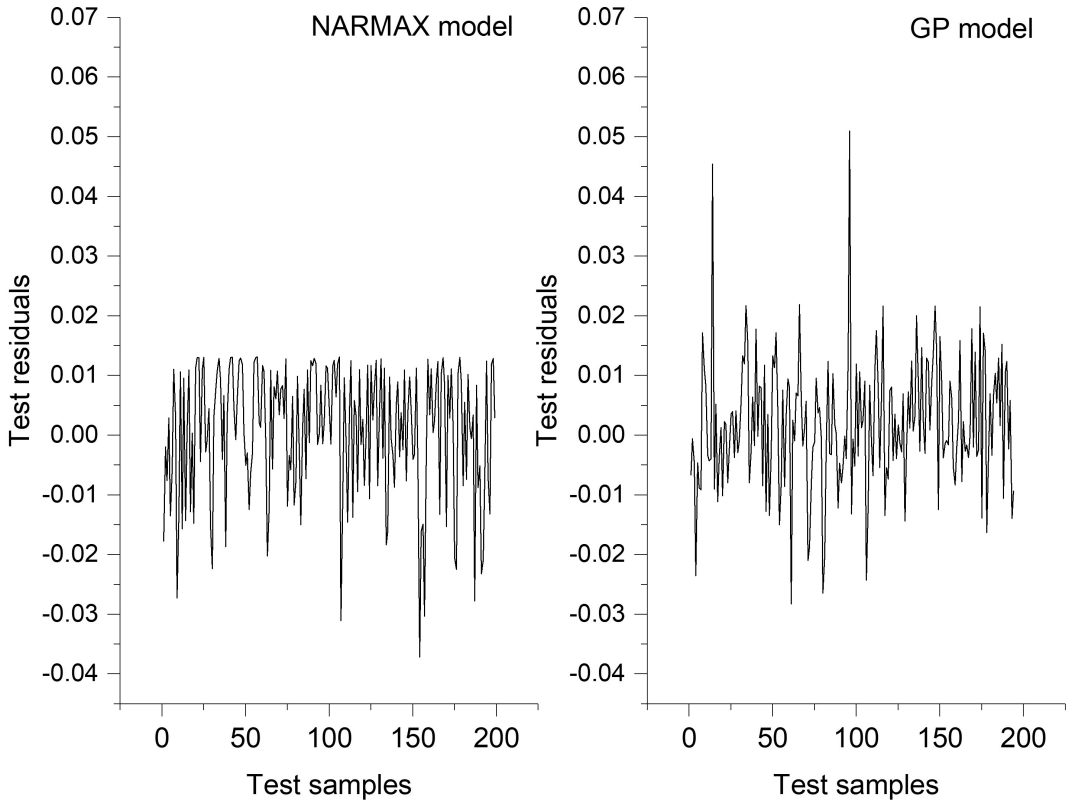

FIGURE 5.2: Test residual comparison between the NARMAX model and the best GP model

the building under model-predictive control, as explained in Section 5.4.6. In Section 5.4.5, we describe how we employed GP using the collected SID data to obtain the required predictive models for MPC.

5.4.1 Building Simulator – EnergyPlus

EnergyPlus is a building energy simulator used to model energy consumption based on dynamic heat transfer calculations (Crawley et al., 2000). The description of the building is provided to EnergyPlus as a text file – the input data file (IDF) – that follows a prescribed format. The file controls all aspects of the simulation from the building geometry and fabric to the building services and other simulation parameters, such as occupancy.

The (key) influence of the external weather is incorporated into the building simulation using a separate file containing weather data, a so-called *weather file*. This allows repeating the computations under different climatic conditions. In this work, we have used design weather files generated from UK meteorological data collected at a station located in Manchester, UK. Two different weather files, both containing one year's data, were used in this

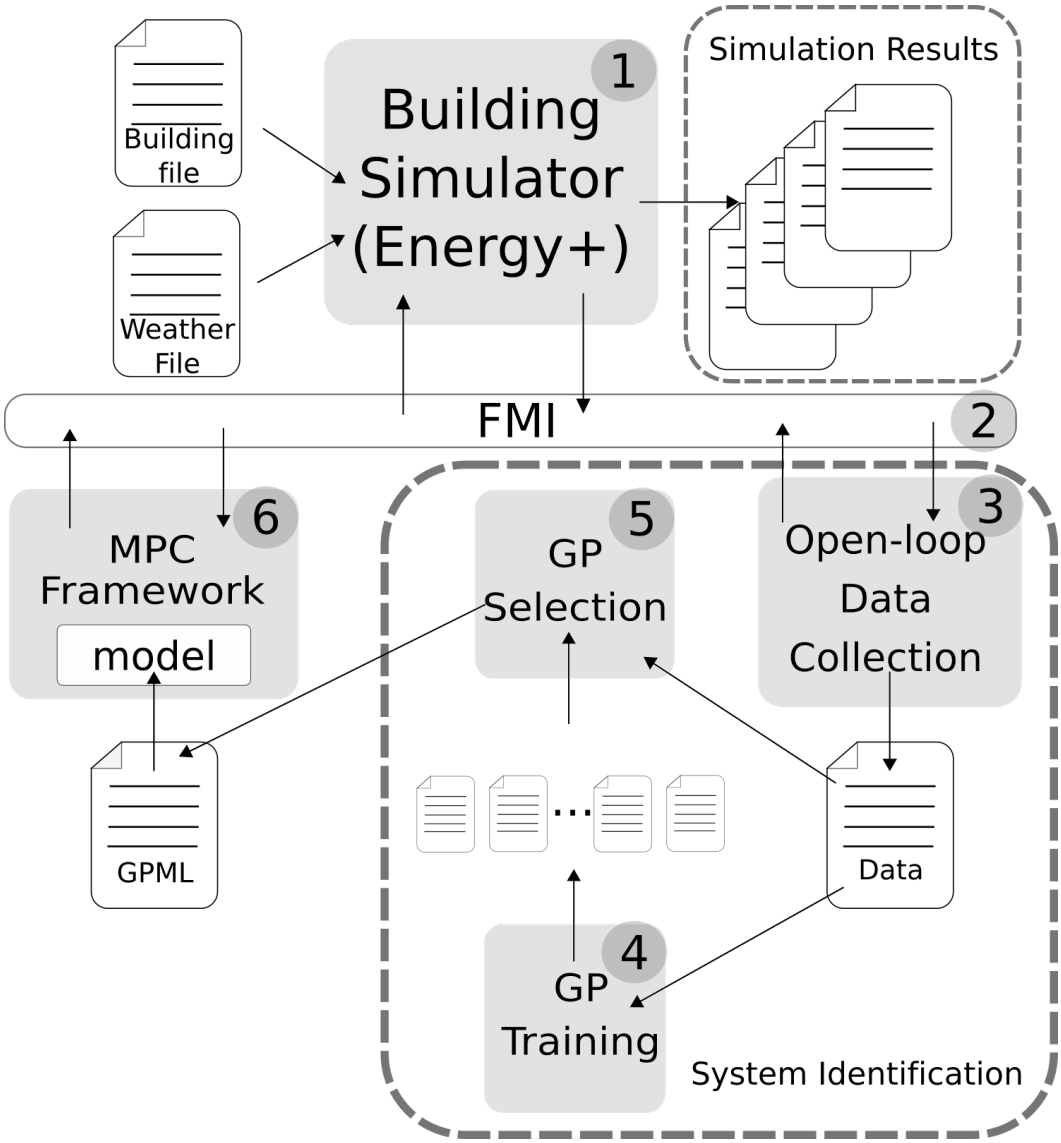

FIGURE 5.3: Overview of the process employed in this work for MPC (contributed by Dr. Yuri Kaszubowski Lopes).

work: the training and validation datasets were extracted from the first weather file, while the second file was used to test the model for an unbroken whole year.

EnergyPlus supports Functional Mock-up Interfaces (FMIs) (Blochwitz et al., 2011), a standardised interface for coupling software units for co-simulation. These software units can add a variety of functionalities to the simulation and are referred to as 'slaves'. The main simulator – EnergyPlus in our case – is referred to as the 'master'. In practice, the FMI defines a set of C language function prototypes that need to be implemented by the slave unit. The master will then call these functions at appropriate times to perform various operations, such as send data, perform calculations, read data, etc. Here we make two distinct uses of the

FMI functionality (see Figure 5.3): first, we inject an excitation sequence to perform open-loop system identification (described in Section 5.4.3). Second, we use FMIs to control the building's heating during MPC experiments (described in Section 5.4.6).

5.4.2 Test Building Description

For this initial report of implementing MPC using a learned dynamic model, we developed a single zone space with a radiator and varying supply of fresh air. The simulated test building is illustrated in Figure 5.4. The zone is a square room with dimensions of 10 m $\times$ 10 m and a height of 3 m. All four walls contain one double glazed window unit measuring 2 m $\times$ 2 m with a sill height of 0.5 m, and placed at the centre of the external walls. This design has a window-to-wall ratio of 13% with equal exposure to North, East, South and West directions. The single zone space has been set to be located in Manchester, UK, which has an oceanic climate (Köppen classification = Cfb) and classified as ASHRAE (American Society of Heating, Refrigeration and Air Conditioning Engineers) climate zone 5c. The construction sets and internal gains for this climate recommended by ASHRAE Standard 189.1 (ASHRAE, 2009) were considered for this space to make sure a realistic set of inputs was defined in the building model for estimating the internally-generated heat as well as the heat loss from the façades. We have used a setpoint temperature of 20 C. The airflow through the space consists of infiltration (0.00023 m^3/s per m^2 of exterior surface) and ventilation that varies due to occupancy and is based on 10 l/s per person in accordance with the Chartered Institute of Building Services Engineers (CIBSE) Guide A (CIBSE, 2015). Heat gains from people, lighting and electrical equipment were also considered on a schedule. The simulated building used here was constructed by Dr. Esmail Saber.

5.4.3 Data Acquisition

The design of appropriate excitation signals for collecting identification data is the most crucial step in system identification as the gathered data are required to be informative enough to capture the underlying system dynamics under all relevant operating conditions. Mathematically, the amplitude of the excitation signal should cover the full range so as to maximise the power of the excitation signal and thus the signal-to-noise ratio. The spectrum of the input signal should excite all frequencies of interest.

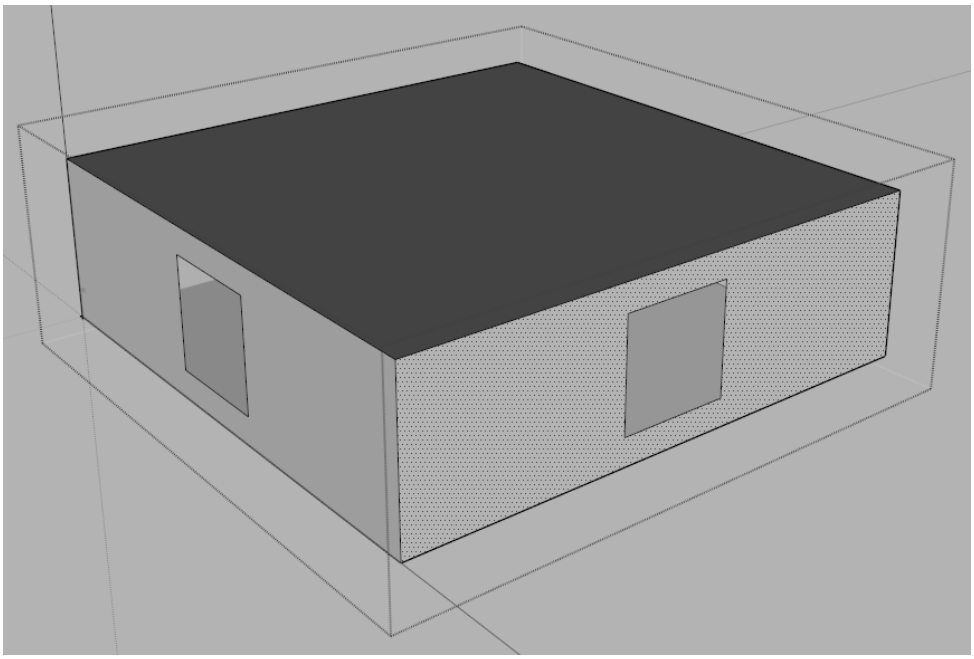

FIGURE 5.4: SketchUp representation of the simulated building (contributed by Dr. Esmail Saber).

For linear systems, *pseudo-random binary sequences* (PRBSs) (Söderström and Stoica, 1989) are commonly used for system identification. In a PRBS, the signal switches between two fixed amplitudes in such a way that that the autocorrelation function of the sequence approximates the properties of white noise, and hence excites all modes of the system. For nonlinear systems – such as that under consideration here – switching between two fixed amplitudes cannot capture the nonlinear behaviour (Nelles, 2001), and so we have employed *APRBSs* (Nelles, 2001) in which the amplitude of a conventional PRBS is randomly varied, thereby probing the nonlinear characteristics of the system.

A PRBS sequence was generated using linear feedback shift registers where the length of the excitation sequence is controlled by a characteristic polynomial of some degree n, and where each polynomial coefficient was either 0 or 1. The maximum repetition period is given by $(2^n - 1)$ (i.e. the maximum length of the sequence before it starts to repeat itself). The consecutive occurrence of the same bit is referred to as a plateau. We employed the polynomial $x^7 + x^6 + 1$, resulting in a sequence length of 127 bits with 64 plateaux, depicted in Figure 5.5(a). The interval between the minimum and maximum radiator flows (0.00 to 0.11 kg/s) was divided by the number of plateaux in the PRBS resulting in a set of different amplitude levels, which were randomly assigned to the PRBS's plateaux, thereby generating the APRBS (Nelles, 2001); an example sequence is shown in Figure 5.5(b). The process of

randomly assigning amplitude levels to the PRBS plateaux was repeated to obtain a set of excitation sequences that covered an entire year. Note that each repetition of the process is likely to generate a completely different APRBS cycle, as seen in Figures 5.5(b-c).

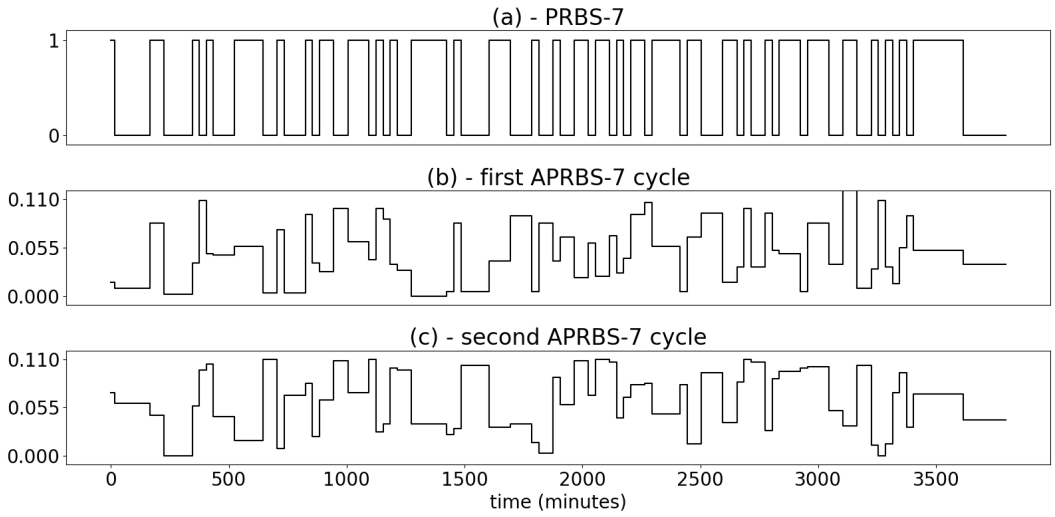

FIGURE 5.5: A PRBS-7 sequence (a) and two different examples of APRBS-7 cycles (b-c) generated over a minimum-to–maximum amplitude range of 0.0 to 0.11 kg/s.

In addition to the characteristic polynomial and the interval of the input sequence, an APRBS is specified by a minimum hold-time T_h that is the duration of each bit; Nelles (Nelles, 2001) suggests that the minimum hold-time should be the same as the dominant time constant of the process. In our case, our only input that can be excited is the mass flow rate through the radiator so we estimated the dominant time constant as approximately 30 minutes by applying a step excitation to the simulated zone. As a result, a single APRBS-7 cycle takes around 3,810 minutes (about 2.6 days), as depicted in Figure 5.5, and 138 complete APRBS cycles are required to cover an entire year.

5.4.4 Input Selection

In system identification, the selection of model inputs is key to model accuracy. Too many redundant or irrelevant input variables hampers the search while increasing the computational burden. Conversely, if input variables of significant influence are omitted, the model will have systematic errors and be more likely have poor prediction accuracy. The input and output variables used in this paper are listed in Table 5.2. All the variables were scaled so as to have most of the values falling into the range 0 to 1. The scaling factors used are listed in Table 5.2.

TABLE 5.2: Variables used in the system identification model

Variable	Variable name	Type	Scaling factor
T_{out}	Outdoor Air Drybulb Temperature (C)	Input	21.0
Q_{solar}	Sum of Direct and Diffuse Solar Radiation (W/m^2)	Input	839.8
MFR	System Node Mass Flow Rate (kg/s)	Input	0.11
y	Zone Air Temperature (C)	Output	21.0

5.4.5 Genetic Programming for Building Identification

- Training & validation

  A dynamical predictive GP model was developed based on an EnergyPlus simulation model and the open-loop excitation data (see Section 5.4.3). Two different weather files were employed: One weather file, denoted `TRY`, comprising 365 days and 35,040 samples, was used to generate data for model training and validation (model selection). The other dataset, denoted `DSY` and of the same size, was used for estimating the model generalisation and prediction accuracy. A particular challenge with this MPC application is that weather conditions play a very important role in determining the internal temperatures of the building but they cannot be experimentally perturbed in the same way as the radiator MFR variable. We have thus used two weather files to allow an evaluation of performance over a complete year independent of the training/validation data.

  The selected variables (see Section 5.4.4) were sampled every 15 minutes for training, validation and testing of the GP models since the MPC process predicts temperatures on this time interval.

  Given a GP model f, at time k the prediction of the zone temperature $\hat{y}_{(k+i)}$ at time $(k+i)$ is approximated from:

$$\hat{y}_{(k+i)} = f(\mathbf{u}_{k+i-1}, \mathbf{u}_{k+i-2}, ..., \mathbf{u}_k, \mathbf{u}_{k-1}, \ldots, \hat{y}_{(k+i-1)}, \hat{y}_{(k+i-2)}, \ldots, y_k, y_{k-1}, \ldots) \tag{5.10}$$

where i ranges from 1 to N, the length of the prediction horizon, and $\mathbf{u}$ are the indexed sequence of exogenous input vectors. In this experiment, a $\mathbf{u}$ vector consisted of three variables: T_{out}, Q_{solar}, and MFR – see Table 5.2. The zone temperature y is the predicted variable.

In the training phase, all the input and output information is known since $\mathbf{u}$ consists of measured values determined by the APRBS excitation sequence (Section 5.4.3) and the known weather data; the zone temperatures up to and including the current time k are also known. For multi-step ahead prediction, the required autoregressive values *later than* time k use previously predicted values from a series of one-step ahead predictions. For example, at time k, $\hat{y}_{(k+1)} = f(\ldots, y_k, y_{k-1}, \ldots, y_{k-m})$. To predict two steps ahead, $\hat{y}_{(k+2)} = f(\ldots, \hat{y}_{(k+1)}, y_k, y_{k-1}, \ldots)$. Note the use of a *predicted* zone temperature $\hat{y}_{(k+1)}$ at time $(k+2)$ since when the model is used in its ultimate control application, the actual value $y_{(k+1)}$ will be unknown as it lies in the future – it therefore has to be estimated. Similarly, the prediction three steps ahead $\hat{y}_{(k+2)}$ uses both $\hat{y}_{(k+1)}$ and $\hat{y}_{(k+2)}$, and so on. Previously predicted values are used $\forall i \in [1 \ldots N]$, as necessary.

Two objectives are used to measure the performance of candidate models during evolution: tree size and MSE. The tree size indicates the complexity of a GP model, and provides parsimony selection pressure that favours simpler models during the evolutionary process. We also seek to minimise the MSE over the training dataset by:

$$MSE = \frac{\sum_{k=k'}^{P} \sum_{i=1}^{N} [\hat{y}_{(k+i)} - y_{(k+i)}]^2}{[P - \max(n,m)] \times N} \tag{5.11}$$

where N is the length of the prediction horizon, and P is the largest index on the training dataset used. Since we require the GP model to provide accurate predictions over the *whole* prediction horizon, minimising (5.11) provides a selective evolutionary pressure to achieve this.

The values of upper limit on the outer summation and the normalising term in (5.11) requires some clarification: Suppose we have Q records of available training data. To train a model that predicts N steps ahead, the final N records of the dataset can only be used for evaluating (5.11) – the index k cannot exceed $(Q-N)$. Similarly,

for an autoregressive model with n lagged $\mathbf{u}$ values and m lagged y values, the first $\max(n,m)$ records of the training set are required to calculate the very first prediction. Consequently, k', the lower limit on the outer summation in (5.11), cannot be less that $[\max(n,m)+1]$. In summary, the MSE in (5.11) is calculated over $Q-N-[\max(n,m)-1]$ records. (Conventionally, the lower limit of this outer summation is taken as $k=1$ and any lagged inputs with (strictly) negative k indices are taken as zero. We have not used this approach here as, in our experience, this sometimes produces anomalous transient predictions.)

The detailed GP parameter settings for building identification are shown in the Table 5.3.

TABLE 5.3: Evolutionary parameters used in this work

Parameter	**Value**
Population size	100
Evolution strategy	Steady state
Initialization method	Ramped half-and-half
Maximum tree depth in initialisation	6
Maximum number of tree evaluations	20000
Function set	$+$, $-$, $\times$, Analytic quotient (Ni, Drieberg, and Rockett, 2013), Δ_1,Δ_2,Δ_3
Terminal set	Input variables; constant values from $\{0.1, 0.2, \ldots, 2.0\}$
Crossover frequency	1.0
Mutation frequency	1.0
Fitness measures	Tree size & MSE (see (5.11))
Selection method	Pareto ranking

- Exporting the Selected GP Tree

  After model validation, the best GP model was selected for use in the EnergyPlus MPC framework. Rather than the cumbersome inconvenience of embedding the GP training within the MPC framework, we have exported the trained GP model using the GPML (Dou, Kaszubowski Lopes, and Rockett, 2018). GPML is an XML-based standard for the interchange of GP trees. The implementations of reading and writing GPML are simple and straightforward since a number of mature, open source XML libraries are available. A trained-and-validated GP tree can thus be directly embedded

as a 'plug-in' component using GPML in larger systems, which provides both convenience and modularity.

5.4.6 MPC Test Framework

Based on the model f in (5.1) approximated with a GP as described in Sections 5.4.3 - 5.4.5, a control law could, in principle, be obtained as the inverse of f – that is a mapping from desired states to the necessary inputs. However, as f is dynamic and nonlinear, calculating its inverse is not a trivial task. The alternative is to perform an explicit optimisation at the current time k of the set of values $\mathscr{U}_k = \{\mathbf{u}_k, \mathbf{u}_{(k+1)}, \ldots, \mathbf{u}_{(k+N-1)}\}$ using f, where N is the prediction horizon, and adjusting $\mathscr{U}_k$ to yield the desired sequence of setpoint temperatures. Thus we obtain $\mathscr{U}_k$ from:

$$\mathscr{U}_k = \text{argmin} \sum_{i=1}^{N} J(k+i) \tag{5.12}$$

where N is the length of the prediction horizon, and J is defined as:

$$J(k+i) = \underbrace{(\Delta T_{k+i})^2}_{\text{temperature}} + \underbrace{\lambda_0 \Delta U_{k+i}}_{\text{control effort}} + \underbrace{\lambda_1 \mathbf{u}_{k+i}}_{\text{energy}} \tag{5.13}$$

where $\Delta T_{k+i} = \hat{y}_{(k+i)} - r_{(k+i)}$ is the difference between the predicted $\hat{y}_{(k+i)}$ and setpoint (i.e. desired) $r_{(k+i)}$ temperatures at time $k+i$, and $\Delta U_{k+i} = \mathbf{u}_{k+i} - \mathbf{u}_{k+i-1}$ is the control effort. The first term in (5.13) obviously penalises deviation from the desired setpoint temperature, while the second term seeks to minimise the extent of changes in the control variable; such a term is frequently included in an MPC setup to minimise wear on the system's actuators.

The third term in (5.13) seeks to minimise the sum of the input quantity, which in the present case is a proxy for input energy. We found it necessary to include this term in (5.13) to ensure that heating was turned off outside working hours when the constraint on the setpoint temperature was relaxed to simply being >6 C to ensure frost protection. Unless this explicit energy minimisation term was included, (5.12) could be minimised by maintaining the zone temperature at 20 C – obviously, >6 C – but setting the sum of the control efforts (ΔU) to zero. That is, not turning off heating at the end of the working day thereby undesirably maintaining heating during the night.

Note that the second and third terms in (5.13) are weighted by the regularisation constants λ_0 and λ_1, which place differing relative penalties on each of the three factors in (5.13). Predictions of zone temperature $\hat{y}_{(k+i)}$ are calculated using the GP model described in Section 5.4.5.

The minimisation in (5.12) was performed using an implementation of the nonlinear, derivative-free optimiser COBYLA (Powell, 1994) together with the Multi-Level Single-Linkage (MLSL) procedure (Rinnooy Kan and Timmer, 1987a; Rinnooy Kan and Timmer, 1987b) from the `NLopt` nonlinear optimization library[3].

The detailed MPC optimisation parameter settings for building testing are shown in the Table 5.4.

TABLE 5.4: MPC optimisation parameters used in this work.

Category	**Parameter**	**Value**
MPC	Prediction horizon N	12 steps
MLSL (global)	Population size	4
	Maximum number of evaluations	131,072
	Stop when objective value less than	0.001
COBYLA (local)	Maximum number of evaluations	8,192
	Stop when objective value less than	0.001
Fitness function	λ_0	10.0
	λ_1	10.0

5.5 Results

5.5.1 Model Performance

We conducted thirty GP training runs, each with an independent initial population, to obtain 30 individual models with the best validation set MSE per run. Among these, the model with the smallest validation MSE was finally selected as the best model. January's data (2,880 records) was used as the training dataset and February's data as validation dataset. Typically, the CPU runtime of each independent experiment takes ∼400 s (on a computer with a 3.30 GHz CPU).

One notable point is that in the training process, T_{out} and Q_{solar} are always assumed known, and we have used measured values. During the validation and test phases, however,

[3] https://nlopt.readthedocs.io/en/latest/

the practical use of the model means that future values of T_{out} and Q_{solar} are unknown. Consequently, we have used *persistent predictions* for future (as yet unknown) weather variables. Namely, the value of the weather variable at time k is assumed to persist unchanged for the whole of the current prediction horizon. Persistent weather prediction is known to be reasonably accurate over the short-term (Antonanzas et al., 2016; Corne et al., 2013) while having the advantage of being simple to implement.

The residuals (i.e. errors) for each of the i-step ahead predictions ($i \in [1 \dots N]$) measured over the 12-month independent test set for the best GP model are shown in Figure 5.6.

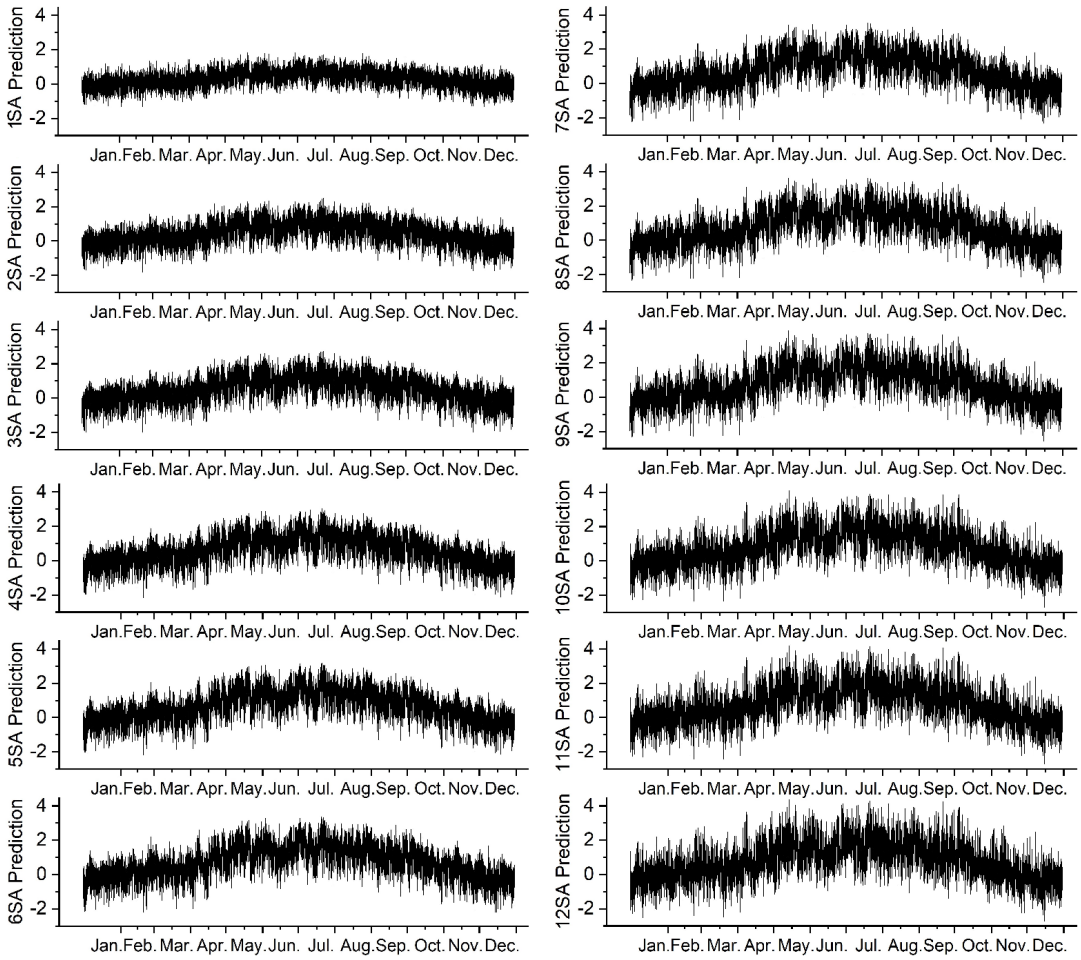

FIGURE 5.6: Residuals of the selected GP model over the test dataset. Each plot shows the residuals for a given number of steps ahead – for example, "OSA" = 1-step head, "2SA"= 2-steps ahead, etc. The units of the ordinate axes are Celsius.

From Figure 5.6, residuals of the one step ahead prediction fall into range from -1.25 to 1.75 C; even in the summer months (June to August) , most residuals are below 1.5 C. Unsurprisingly, as the predictions extend further into the future, the envelope of residuals expands

although only slightly. In addition, a distinct seasonal 'bow' becomes more noticeable with increasing i value.

Our choice of the metric in (5.11) was deliberately designed to give equal weight to the prediction errors over the whole prediction horizon. In 12-step-ahead prediction, residuals in January, February, March, November and December range from -2 to 2.5 C. Residuals in the months such as May, June, July and August reach their highest values of 4 C and with an average lower bound 0 C error. After increasing during summer, the sizes of the residuals decrease later in the year. In summary, the GP model provides promising predicted temperatures: the magnitudes of the residuals expand with increasing prediction step size i with the biggest residual less than 4 C. The best model presented in GPML form can be found at Appendix A and:

https://figshare.com/articles/gpTreeConstantWFInVall_xml/7398797

The best GP model has 79 nodes, including all types of nodes described in Table 5.3. The XML Schema for GPML can be found at Appendix B and:

https://github.com/pirlite2/gpml-schema

5.5.2 MPC Performance

Figure 5.7 shows a typical result of controlling the test building during the month of February (from the 32$^{\text{nd}}$ to 59$^{\text{th}}$ day of the test year) using the MPC framework from Section 5.4. The test weather data is independent of the data used to train/validate the predictive model. The reference value $r_{(k+i)}$ was set to match 20 C during working hours (9:00 to 17:00). To achieve frost protection out-of-hours, the reference value was set at $\geq$ 6 C. That is, we only apply a penalty out-of-hours if the temperature falls *below* the frost-protection reference value. Formally, $\Delta T_{(k+i)}$ out-of-hours is defined as:

$$\Delta T_{(k+i)} = \begin{cases} \hat{y}_{(k+i)} - r_{(k+i)} & \text{if } \hat{y}_{(k+i)} < r_{(k+i)}. \\ 0 & \text{otherwise.} \end{cases} \tag{5.14}$$

As described in Section 5.5.1, we have predicted future (unknown) weather values using persistence, that is, assuming the variable has the same value over the whole of the prediction horizon as it does at the start.

The upper plot in Figure 5.7 shows the zone temperatures, and also a ± 1 C range during working hours (dotted lines). The lower plot shows the MFR control variable (hot water flow through the radiator).

From Figure 5.7, it is clear that zone temperatures are mostly being maintained within the ± 1 C band during working hours. A noteworthy feature of the MFR control variable is that it varies fairly smoothly over the day, and is being reduced to small values towards the ends of the working days; we infer that MPC is exploiting the energy stored in the building's fabric to maintain the setpoint temperature up to the end of the working day, thereby avoiding direct heating, if possible, which may lead to energy savings.

Considering the thermal performance of MPC, it successfully maintained the zone temperature within the ± 1 C range for 83.3% and ± 2 C for 95.2% of the (working) time. The 1 C band is generally deemed comfortable, with a small proportion (less than a quarter) of occupants feeling mildly uncomfortable at the extremes of the 2 C band (CIBSE, 2015). Although there will be some discomfort for the short periods outside these bands that fall at either end of the working day, these are rare.

5.6 Discussion and Future Work

As stated above, scope of this work is to present what we believe to be the first report of buildings MPC using a predictive model learned data acquired from the building. Our aim here has been to document the methodology we have used although a great deal of work remains to be done both in terms of 'fine tuning' this, and in extending it. We can identify a number of interwoven topics that will be the subject of future work, and will be published as future work.

Although we have reported only one instance of a predictive model trained on 30 days open-loop excitation data, optimisation of the training process clearly needs to be approached systematically. Although the test residuals shown in Figure 5.6 are clearly adequate to produce acceptable control, as evidenced by Figure 5.7, the model residuals exhibit a noticeable seasonal effect – an upward 'bowing' in the middle of the plots. In the summer months, the actual temperatures are systematically somewhat higher than those predicted by the model, but in the present application with only heating of the building, this turns out not to make a

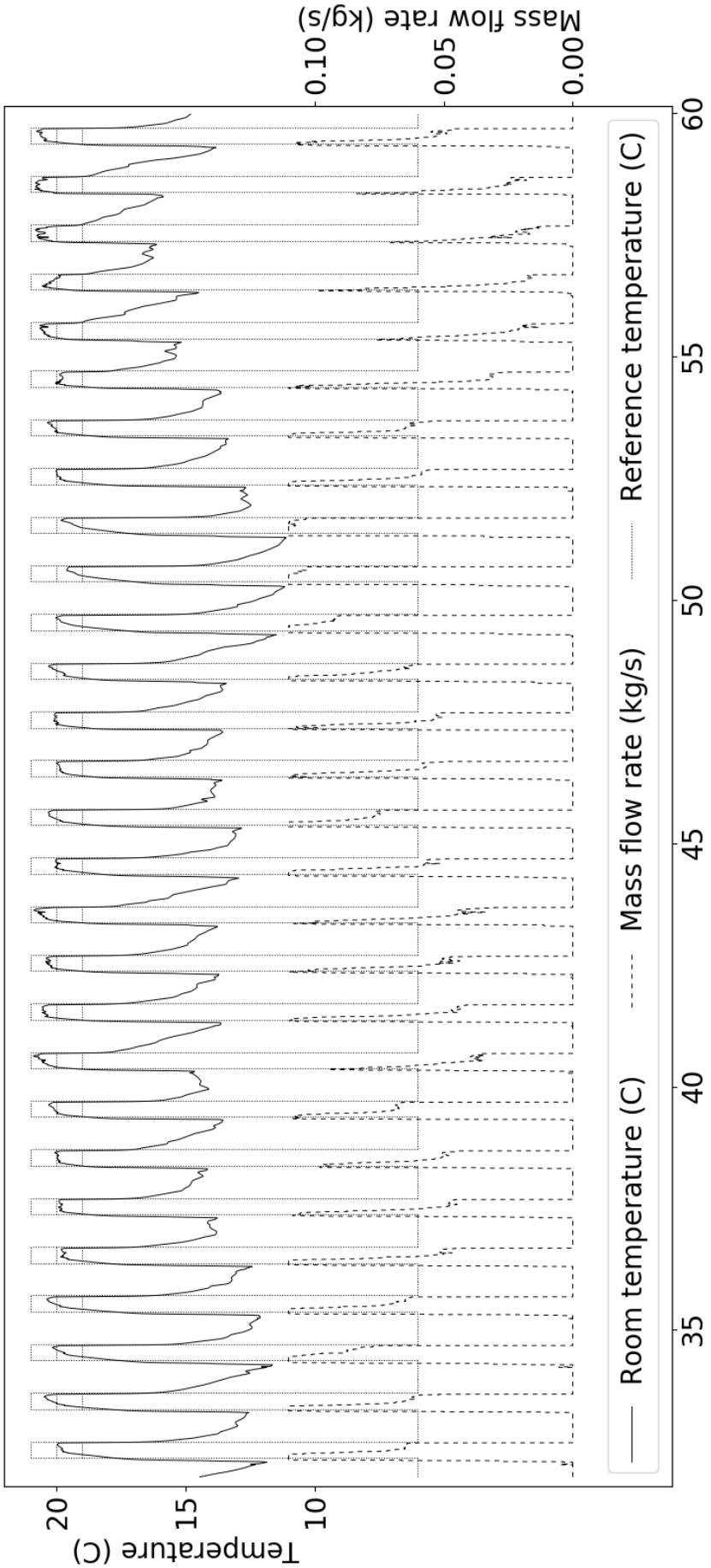

FIGURE 5.7: Typical results of MPC controlling a single-zone room for the (test) month of February. The upper plot is the zone temperature, and the lower plot the mass flow rate (MFR) controlled variable. The rectangular upper plot represents the temperature schedule.

great difference since heating is not necessary in the summer months. For a more complicated building, however, that includes cooling as well as heating, the seasonal effects may produce unacceptable conditions. Consequently, improving the model quality is therefore clearly an area for future work, and a number of factors need to be examined.

The duration of the open-loop excitation experiment: Although we report only data for 30 days of system identification, it seems possible to train adequate models with shorter data sequences than this. The trade-off between the length of the SID experiment and model quality needs to be explored. Naively, one would expect model quality to improve with longer SID sequences (= more training data), but extending the system identification experiment has implications for both the amount of energy used as well as the practicality of conducting the experiment.

For the sake of simplicity, we have assumed persistent weather predictions. That is, we assume the weather inputs have the same values over the entirety of the prediction horizon as they do at the start of the prediction horizon. Persistence is known to be acceptable in the short term, but it is possible that more elaborate methods may give improved results.

The length of the prediction horizon has been chosen to be around six times the characteristic time constant of the building (although the step response of the building is clearly not a first-order characteristic). The trade-offs inherent in selecting a prediction horizon are well-known in the MPC literature (Camacho and Bordons, 2004): a longer horizon allows a more relaxed planning timeframe and tends to avoid overly aggressive control moves, while producing more uncertain predictions due to the length of time into the future over which they are being made. Shorter prediction horizons face the converse issues.

The MPC framework we have reported uses the rather conventional objective of penalising deviation from a setpoint, in this case zone temperature, together with an appropriately weighted term is to minimise control effort (a proxy for actuator wear). In addition, we have included a term to minimise energy consumption over the prediction horizon; this latter term proved necessary for proper operation out of working hours. Clearly the regularisation constants (here denoted λ_0 and λ_1) will have an influence on the control although quite how significant this will be needs to be investigated.

Such a regularisation framework has been commonly used in previously published reports on buildings MPC (Rockett and Hathway, 2017), and appears to have been adopted

straightforwardly from its widespread use in chemical engineering and related industrial applications of MPC. In process engineering, of course, it is frequently important to maintain some optimal process temperature to maximise yield, etc. Maintaining zone temperatures within a small band, however, is generally unnecessary in buildings – indeed relaxing the temperature constraints on a zone can have beneficial energy-saving advantages. More generally, tuning regularisation constants is known to be problematic and time-consuming, and to a large extent, a conventional regularisation framework militates against our overall objective of automating implementation of MPC in buildings in order to make it economically viable. Consequently, alternative minimisation objective functions may well be more appropriate in a building setting. For example, minimising energy usage (over the prediction horizon) subject to the constraints of maintaining zone temperatures within, say, a ± 2 C band.

The system identification experiments reported here involve open-loop excitation of the building. It is well-known that open-loop excitation can drive the systems states to extremes since there is no feedback control to prevent this. Apart from potentially consuming significant amounts of energy, performing an open-loop system identification experiment on an occupied building would probably be unacceptable. Indeed, it is highly likely that the occupants would take atypical actions, such as opening windows and doors, to make the internal conditions more acceptable to themselves, thereby undermining the validity of the system identification data. Rockett and Hathway (Rockett and Hathway, 2017) have already suggested closed-loop system identification as a way of addressing the shortcomings of open-loop identification: closed-loop identification (Gevers, 2005) maintains the system under control using an initial (crude) predictive model while typically applying perturbations to the desired setpoint from which an improved, control-capable model can be derived. This process of closed-loop re-estimation can, of course be repeated periodically as-and-when the building's characteristics change.

Finally, although the work presented here has been done in simulation – as is very common in the initial steps of the control project – the ultimate 'proof' of the methodology is to demonstrate its use on a real building. This too is future work.

5.7 Conclusions

In this work, we have reported the first use of GP to obtain predictive models for the MPC of buildings. Currently, the large-scale adoption of MPC in buildings is rendered uneconomic by the time and cost involved in the design and tuning of predictive models by expert control engineers. We have shown that GP is able to automate this process using an open-loop excitation experiment. The resulting MPC simulation is able to maintain the internal temperature of a single-zone test building to within ± 1 C of the desired setpoint most of the time; we further infer that MPC is able to effectively exploit the heat stored in the building's fabric towards the end of a working day rather than applying direct heating. The results of this work have significant implications for enabling the wide-scale deployment of MPC in non-domestic buildings, and for the potential reduction in CO_2 emissions by improving the efficiency of building operation.

Chapter 6

Conclusions

In this thesis, a series of semantic-based local search methods within a multiobjective GP framework were investigated for improving the search efficiency in empirical modelling. Practically, when a trained model is obtained after an evolutionary process, the format in which the tree can be shared and how to effectively apply it in further complex systems are critical questions to be solved. An XML-based standard for the interchange of GP trees, GPML, was proposed, which provides a simple way for researchers to share the *actual* trees generated during their research and allows GP trees to be used as a 'plug-in' element for solving further complex applications, such as control problems. Taking advantage of GPML, GP was researched for identifying the thermal dynamics of buildings which are acknowledged to be complex and nonlinear. Then, the control performance of the obtained GP model was further estimated through a MPC process. This chapter reviews the work that has been done and outlines the contributions claimed in this thesis.

GP has exhibited good approximation ability on modelling complex systems (Poli, Langdon, and McPhee, 2008). However, the fact that conventional GP modifies trees at the syntactic level can impair the search efficiency due to no regard of how program fitness is changed. To address this issue, semantically-aware methods were researched and shown to be of great help in improving search power during the evolution process (Ffrancon and Schoenauer, 2015; Vanneschi, Castelli, and Silva, 2014). Many semantic-based local search methods (Azad and Ryan, 2014; Iba, Garis, and Sato, 1994; Topchy and Punch, 2001) integrated in GP optimize solutions by fine-tuning node functionality or numerical coefficients. In this thesis, we proposed semantic-based local search methods within a multiobjective GP framework. This combinatorial method optimizes candidates by changing the tree morphologies, which is comparatively less explored in this field. Besides, the performance of a new genetic

operator, RDO (Pawlak, Wieloch, and Krawiec, 2015), that acts both as a genetic operator and local search operator in both steady state and generational GP were also studied and compared with our methods.

The main contributions claimed from the semantic-based local search experiment are summarized as follows.

1. A range of semantic-based local search methods within a multiobjective GP framework were implemented and assessed for approximating a series of commonly-used univariate symbolic benchmark regression functions. The performance of GP supplemented by semantically-aware local search methods was explored.

2. This thesis extends consideration of the influence of local search to a multi-objective GP framework since the balance between program fitness with model complexity is a key research issue in practical problems.

3. This thesis optimized GP solutions by focusing on changing tree morphologies, unlike techniques that produce better models by fine-tuning model parameters or node functions.

4. The performance of RDO both as genetic operator and local search operator in steady state and generational multi-objective GP was explored. In generational GP, RDO was clearly powerful when adopted as a genetic operator. However, in steady state GP, the performance is worse than that with a generational scheme. Such deterioration caused by RDO hinders the search efficiency of the steady state strategy. The RDO search operator is clearly sensitive to the evolutionary strategy.

5. The proposed semantic-based GP local search is able to create models with statistically better generalization performance and smaller tree size than a generational GP which uses RDO as a genetic operator.

6. When GP served as local search operator, Ito's depth fair selection method performs best since the optimization of subtrees closer to the root node has more obvious influence on the improvement of the entire tree.

GP and its combinatorial methods have exhibited good performance in approximating symbolic regression functions. When GP models are used for modelling real-world complex systems or further applications like control, the interchange of GP trees is an essential

requirement. Practically, due to the empirical and stochastic nature of GP, it is difficult to reproduce research findings just relying on a brief description of how models are generated, which is often incomplete. The replication deficiency may hinder progress in this research field. We proposed an XML-based standardized format for the interchange of GP trees, named GPML.

The main contributions claimed from the proposal of GPML are:

1. GPML is implemented. The instruction of how to translate a GP tree into a GPML format is detailed. In addition, how to read and validate a GPML file were also described.

2. GPML provides convenience for accurate interchange and calculation of GP models. GP trees represented by GPML can be evaluated and compared directly by researchers without reproducing others' work (often) with incomplete information. It allows researchers to correctly share their research findings with the hope of speeding scientific progress in this field.

3. GPML allows GP trees to be used as a 'plug-in' component in larger systems, improving usage efficiency and providing application modularity.

MPC is a powerful control strategy and has been widely applied in chemical industrial processes. However, the applications of MPC on buildings are little explored (Rockett and Hathway, 2017). Applying MPC on buildings is worth pursuing because MPC is more likely to offer significant energy savings compared to conventional rule-based techniques. The characteristics of buildings are widely acknowledged to be nonlinear and dynamic, and traditional models used for building are hand-crafted by highly skilled control engineers. The high costs of the creation and calibration of the predictive model make MPC economically unviable in practice. Consequently, being able to automatically evolve both model structure and parameters, GP is a promising tool for approximating building dynamics and further used for MPC.

The main contributions[1] claimed from the nonlinear dynamic system identification and MPC based on GP are:

[1]Our group member Yuri Kaszubowski Lopes provided the data of a simulated building used for open-loop system identification and provided the MPC results.

1. A novel building MPC methodology based on GP was proposed and implemented. We believe this is the first time GP was been used to produce predictive models for building MPC.

2. A simulated test building was implemented by EnergyPlus. The required weather data was collected by the UK Meteorological Office at a station located at Manchester, UK. An excitation sequence (A PRBS-7 sequence) was designed and used to perform open-loop system identification.

3. Data of the open-loop control process on the simulated test building was recorded and further served as training, validation and test data sets for GP-based system identification.

4. GP-based system identification was implemented. The obtained GP model was represented in GPML format and used for the further control process.

5. The building MPC was implemented using the learned GP model. The experimental result indicates that GP is able to automate the open loop system identification problem. GP-based building MPC can maintain the internal temperature of the test building within ± 1 C of the desired setpoint. This experiment encourages the attempts of employing GP-based MPC strategy in non-domestic buildings.

Chapter 7

Future Work

The work has been done so far in this thesis shows GP is a powerful machine learning technique for solving nonlinear system identification and MPC problems. GP has a strong potential to be applied in more complex systems, such as buildings. Several recommendations yet to be incorporated into the current framework point to future research directions.

7.1 Local Search in GP

For the investigation of the effects of GP local search, all the benchmarks used in this thesis are bidimensional datasets. A discussion about the possibility of using more complex higher-dimensional datasets is worth checking in the future to assess the performance of GP on dealing with complex problems with more independent variables.

Additionally, in this thesis, model complexity is measured by counting the nodes, which is the most straightforward method and has been criticized for being deceptive. Recently, remarkable research undertaken by Ni and Rockett (Ni and Rockett, 2015) that utilized Tikhonov regularization as a complexity measure and was proven better than the conventional complexity measure. This new model complexity measurement can be incorporated in GP to explore its influence on the local search process.

7.2 Building MPC Based on GP Models

It is observed that GP can produce models that allow the MPC of building to maintain the desired temperature band ranging ± 1 C of the desired setpoint in a single zone space. The control performance of GP models on multi-zone space is an interesting topic and worth being estimated in the future for practical application in real non-domestic buildings.

MPC relies heavily on the quality of the model obtained. The goodness-of-fit of the learned GP model further depends on the dataset that used for training. Therefore, proper design of the excitation signal for system identification is a critical requirement, which is considered the most difficult step in the process of system identification. Apart from the A PRBS-7 sequence that we used in this thesis, a future effort on exploring other excitation signals, such as A PRBS-7 sequences transformed by Hadamard matrix, will be made to find out whether fitter models can be generated by using other excitation signals for system identification of buildings more efficiently.

Moreover, in building identification (Section 5.5.1), the model residuals (Figure 5.7) exhibit a noticeable seasonal effect over the test dataset. Since the test set used contains data for a full year, while the training set contains only data for the first month, it is reasonable to suspect that the upward 'bowing' in test residuals is due to the fact that the training set did not provide complete system dynamics. Therefore, the relationship between the length of training dataset and the seasonal effect should be studied. In addition, the validation set used for model selection in the experiment also contains only one month of data. Insufficient information may make it difficult to select a single model that best fits the characteristics of a building. Therefore, in future work, an assembly of models should be investigated to reduce the risk of the selection of inappropriate single models (Chan and Pauwels, 2018).

This thesis researched open-loop GP-based MPC of the building. Due to the lack of feedback control scheme, open-loop excitation can drive the system to extreme states, which inevitably consumes significant amounts of energy. To address this drawback, research on closed-loop system identification will be undertaken to derive control-capable models to maintain systems under control. Additionally, the effects of occupant activity will also be considered in future work.

Appendix A

The Best Model Presented in GPML Form

The model presented in GPML used for building MPC in Section5.5.1 is shown:

```
<?xml version='1.0' ?>
<gpTree noVectorElements="4" firstIndex="1">
    <binaryNode operation="+">
        <binaryNode operation="AQ">
            <binaryNode operation="+">
                <binaryNode operation="AQ">
                    <terminalNode vectorIndex="4"/>
                    <constantNode value="1.0000"/>
                </binaryNode>
                <binaryNode operation="*">
                    <constantNode value="0.2000"/>
                    <binaryNode operation="+">
                        <unaryNode operation="delay3">
                            <unaryNode operation="delay3">
                                <constantNode value="0.8000"/>
                            </unaryNode>
                        </unaryNode>
                        <binaryNode operation="AQ">
                            <unaryNode operation="delay2">
                                <terminalNode vectorIndex="4"/>
```

```
                    </unaryNode>
                    <unaryNode operation="delay2">
                        <binaryNode operation="AQ">
                            <binaryNode operation="-">
                                <constantNode value="1.8000"/>
                                <constantNode value="0.2000"/>
                            </binaryNode>
                            <unaryNode operation="-">
                                <terminalNode vectorIndex="4"/>
                            </unaryNode>
                        </binaryNode>
                    </unaryNode>
                </binaryNode>
            </binaryNode>
        </binaryNode>
        <constantNode value="1.0000"/>
    </binaryNode>
    <binaryNode operation="*">
        <constantNode value="0.2000"/>
        <binaryNode operation="+">
            <unaryNode operation="delay3">
                <unaryNode operation="delay3">
                    <constantNode value="0.8000"/>
                </unaryNode>
            </unaryNode>
            <binaryNode operation="AQ">
                <binaryNode operation="AQ">
                    <binaryNode operation="+">
                        <terminalNode vectorIndex="3"/>
                        <terminalNode vectorIndex="1"/>
                    </binaryNode>
```

```
        <constantNode value="1.0000"/>
    </binaryNode>
    <unaryNode operation="delay1">
        <binaryNode operation="AQ">
            <unaryNode operation="delay3">
                <binaryNode operation="AQ">
                    <binaryNode operation="AQ">
                        <binaryNode operation="*">
                            <terminalNode vectorIndex="4"/>
                            <terminalNode vectorIndex="3"/>
                        </binaryNode>
                        <binaryNode operation="+">
                            <unaryNode operation="-">
                                <binaryNode operation="AQ">
                                    <unaryNode operation="delay2">
                                        <terminalNode vectorIndex="1"/>
                                    </unaryNode>
                                    <binaryNode operation="-">
                                        <constantNode value="0.2000"/>
                                        <terminalNode vectorIndex="1"/>
                                    </binaryNode>
                                </binaryNode>
                            </unaryNode>
                            <terminalNode vectorIndex="1"/>
                        </binaryNode>
                    </binaryNode>
                    <binaryNode operation="+">
                        <binaryNode operation="*">
                            <terminalNode vectorIndex="4"/>
                            <terminalNode vectorIndex="2"/>
                        </binaryNode>
                        <unaryNode operation="delay3">
```

```
                    <unaryNode operation="delay2">
                        <unaryNode operation="delay3">
                            <terminalNode vectorIndex="3"/>
                        </unaryNode>
                    </unaryNode>
                </unaryNode>
            </binaryNode>
        </binaryNode>
    </unaryNode>
    <binaryNode operation="+">
        <binaryNode operation="-">
            <binaryNode operation="+">
                <binaryNode operation="-">
                    <terminalNode vectorIndex="4"/>
                    <terminalNode vectorIndex="4"/>
                </binaryNode>
                <binaryNode operation="-">
                    <terminalNode vectorIndex="4"/>
                    <terminalNode vectorIndex="3"/>
                </binaryNode>
            </binaryNode>
            <unaryNode operation="-">
                <binaryNode operation="+">
                    <terminalNode vectorIndex="1"/>
                    <terminalNode vectorIndex="4"/>
                </binaryNode>
            </unaryNode>
        </binaryNode>
        <binaryNode operation="-">
            <binaryNode operation="-">
                <constantNode value="1.2000"/>
                <constantNode value="1.2000"/>
```

```
                                    </binaryNode>
                                    <unaryNode operation="delay1">
                                        <terminalNode vectorIndex="4"/>
                                    </unaryNode>
                                </binaryNode>
                            </binaryNode>
                        </binaryNode>
                    </unaryNode>
                </binaryNode>
            </binaryNode>
        </binaryNode>
    </binaryNode>
</gpTree>
```

Appendix B

GPML Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" version="3.0">
  <xs:annotation>
    <xs:documentation> XML Schema for Genetic Programming Markup Language (GPML) v3
            Copyright (C) 2017-2019  - Peter Rockett

            This program is free software: you can redistribute it and/or modify
            it under the terms of the GNU General Public License as published by
            the Free Software Foundation, either version 3 of the License, or
            (at your option) any later version.

            This program is distributed in the hope that it will be useful,
            but WITHOUT ANY WARRANTY; without even the implied warranty of
            MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
            GNU General Public License for more details.

            You should have received a copy of the GNU General Public License
            along with this program.  If not, see http://www.gnu.org/licenses/.
      </xs:documentation>
  </xs:annotation>
```

```
<!-- GPML data type definitions-->
    <xs:simpleType name="singleCharacterType">
        <xs:restriction base="xs:string">
            <xs:length value="1"/>
        </xs:restriction>
    </xs:simpleType>

    <xs:simpleType name="scalarDataType">
        <xs:union memberTypes="xs:boolean xs:integer xs:double singleCharacterType xs:string"/>
    </xs:simpleType>

    <xs:annotation>
        <xs:documentation>
            Defines format of string within an explicitly listed tuple to be enclosed in single
            quotes; single quotes within the string need to be represented with the escaped "%apos" sequence
        </xs:documentation>
    </xs:annotation>
    <xs:simpleType name="listTupleStringDataType">
        <xs:restriction base="xs:string">
            <xs:pattern value="'[.^']*'"/>
        </xs:restriction>
    </xs:simpleType>

    <xs:simpleType name="listDataType">
        <xs:union memberTypes="xs:boolean xs:integer xs:double singleCharacterType listTupleStringDataType"/>
    </xs:simpleType>

    <xs:annotation>
        <xs:documentation>
        Defines tuple as a text content list; literal string constants are demarcated by being enclosed in apostrophes
        </xs:documentation>
    </xs:annotation>
```

```
<xs:simpleType name="tupleType">
        <xs:list itemType="listDataType"/>
</xs:simpleType>

<!-- input node definition-->
<xs:annotation>
        <xs:documentation>
                tupleIndex is index into the input tuple.
        </xs:documentation>
</xs:annotation>
<xs:complexType name="scalarTerminalNodeType">
<xs:attribute name="tupleIndex" type="xs:nonNegativeInteger" use="required"/>
</xs:complexType>

<xs:simpleType name="tupleTerminalNodeType">
        <xs:restriction base="xs:string"/>
</xs:simpleType>

<!-- Constant node definition -->
<xs:simpleType name="dataTypeType">
        <xs:restriction base="xs:token">
                <xs:enumeration value="boolean"/>
                <xs:enumeration value="integer"/>
                <xs:enumeration value="double"/>
                <xs:enumeration value="char"/>
                <xs:enumeration value="string"/>
                <xs:enumeration value="tuple"/>
        </xs:restriction>
</xs:simpleType>

<xs:simpleType name="constDataType">
        <xs:union memberTypes="xs:boolean xs:integer xs:double singleCharacterType xs:string tupleType"/>
</xs:simpleType>
```

```
    <xs:complexType name="constantNodeType">
        <xs:simpleContent>
            <xs:extension base="constDataType">
                <xs:attribute name="dataType" type="dataTypeType" use="optional" default="double"/>
            </xs:extension>
        </xs:simpleContent>
    </xs:complexType>

    <xs:annotation>
        <xs:documentation>
            Restricts "adfNameType" to be a string comprising a letter or underscore followed by any number
            of alphanumeric characters or underscores. This follows conventional subroutine naming
            conventions in programming languages
        </xs:documentation>
    </xs:annotation>
    <xs:simpleType name="adfNameType">
        <xs:restriction base="xs:token">
            <xs:pattern value="[a-zA-Z_][a-zA-Z_0-9^ ]*"/>
        </xs:restriction>
    </xs:simpleType>

    <xs:complexType name="adfCallNodeType">
        <xs:attribute name="name" type="adfNameType" use="required"/>
    </xs:complexType>

<!-- Internal node definitions-->
    <xs:complexType name="unaryNodeType">
        <xs:sequence minOccurs="1" maxOccurs="1">
            <xs:choice>
                <xs:element name="input" type="scalarTerminalNodeType"/>
                <xs:element name="tupleInput" type="tupleTerminalNodeType"/>
                <xs:element name="constant" type="constantNodeType"/>
                <xs:element name="adfCall" type="adfCallNodeType"/>
```

```
                <xs:element name="unary" type="unaryNodeType"/>
                <xs:element name="binary" type="binaryNodeType"/>
                <xs:element name="ternary" type="ternaryNodeType"/>
                <xs:element name="nAry" type="nAryNodeType"/>
            </xs:choice>
        </xs:sequence>
        <xs:attribute name="operation" type="xs:token" use="required"/>
        <xs:attribute name="parameterString" type="xs:token" use="optional"/>
</xs:complexType>

<xs:complexType name="binaryNodeType">
        <xs:sequence minOccurs="2" maxOccurs="2">
            <xs:choice>
                <xs:element name="input" type="scalarTerminalNodeType"/>
                <xs:element name="tupleInput" type="tupleTerminalNodeType"/>
                <xs:element name="constant" type="constantNodeType"/>
                <xs:element name="adfCall" type="adfCallNodeType"/>
                <xs:element name="unary" type="unaryNodeType"/>
                <xs:element name="binary" type="binaryNodeType"/>
                <xs:element name="ternary" type="ternaryNodeType"/>
                <xs:element name="nAry" type="nAryNodeType"/>
            </xs:choice>
        </xs:sequence>
        <xs:attribute name="operation" type="xs:token" use="required"/>
        <xs:attribute name="parameterString" type="xs:token" use="optional"/>
        </xs:complexType>

<xs:complexType name="ternaryNodeType">
        <xs:sequence minOccurs="3" maxOccurs="3">
            <xs:choice>
                <xs:element name="input" type="scalarTerminalNodeType"/>
                <xs:element name="tupleInput" type="tupleTerminalNodeType"/>
                <xs:element name="constant" type="constantNodeType"/>
```

```
                <xs:element name="adfCall" type="adfCallNodeType"/>
                <xs:element name="unary" type="unaryNodeType"/>
                <xs:element name="binary" type="binaryNodeType"/>
                <xs:element name="ternary" type="ternaryNodeType"/>
                <xs:element name="nAry" type="nAryNodeType"/>
            </xs:choice>
        </xs:sequence>
        <xs:attribute name="operation" type="xs:token" use="required"/>
        <xs:attribute name="parameterString" type="xs:token" use="optional"/>
</xs:complexType>

<xs:complexType name="nAryNodeType">
        <xs:sequence minOccurs="1" maxOccurs="unbounded">
            <xs:choice>
                <xs:element name="input" type="scalarTerminalNodeType"/>
                <xs:element name="tupleInput" type="tupleTerminalNodeType"/>
                <xs:element name="constant" type="constantNodeType"/>
                <xs:element name="adfCall" type="adfCallNodeType"/>
                <xs:element name="unary" type="unaryNodeType"/>
                <xs:element name="binary" type="binaryNodeType"/>
                <xs:element name="ternary" type="ternaryNodeType"/>
                <xs:element name="nAry" type="nAryNodeType"/>
            </xs:choice>
        </xs:sequence>
<xs:attribute name="arity" type="xs:positiveInteger" use="required"/>
<xs:attribute name="operation" type="xs:token" use="required"/>
<xs:attribute name="parameterString" type="xs:token" use="optional"/>
</xs:complexType>

<!-- ADF definition type-->
<xs:complexType name="adfDefinitionType">
        <xs:choice>
            <xs:element name="input" type="scalarTerminalNodeType"/>
```

```
            <xs:element name="tupleInput" type="tupleTerminalNodeType"/>
            <xs:element name="constant" type="constantNodeType"/>
            <xs:element name="adfCall" type="adfCallNodeType"/>
            <xs:element name="unary" type="unaryNodeType"/>
            <xs:element name="binary" type="binaryNodeType"/>
            <xs:element name="ternary" type="ternaryNodeType"/>
            <xs:element name="nAry" type="nAryNodeType"/>
        </xs:choice>
        <xs:attribute name="name" type="adfNameType" use="required"/>
</xs:complexType>

<!-- Overall tree definition-->
<xs:simpleType name="firstIndexType">
        <xs:restriction base="xs:integer">
            <xs:enumeration value="0"/>
            <xs:enumeration value="1"/>
        </xs:restriction>
</xs:simpleType>

<xs:complexType name="gpTreeType">
        <xs:sequence>
            <!-- Define zero or more ADF definitions -->
            <xs:element name="adfDefinition" type="adfDefinitionType" minOccurs="0" maxOccurs="unbounded"/>
            <!-- Define a single tree root node -->
            <xs:choice>
                <xs:element name="input" type="scalarTerminalNodeType"/>
                <xs:element name="tupleInput" type="tupleTerminalNodeType"/>
                <xs:element name="constant" type="constantNodeType"/>
                <xs:element name="adfCall" type="adfCallNodeType"/>
                <xs:element name="unary" type="unaryNodeType"/>
                <xs:element name="binary" type="binaryNodeType"/>
                <xs:element name="ternary" type="ternaryNodeType"/>
                <xs:element name="nAry" type="nAryNodeType"/>
```

```
                </xs:choice>
            </xs:sequence>
            <xs:attribute name="noTupleElements" type="xs:positiveInteger" use="required"/>
            <xs:attribute name="firstIndex" type="firstIndexType" use="required"/>
        </xs:complexType>

        <xs:element name="gpTree" type="gpTreeType">
            <xs:unique name="testUniqueADFnames">
                <!-- Enforce unique ADF names -->
                <xs:selector xpath="./adfDefinition"/>
                <xs:field xpath="@name"/>
            </xs:unique>
        </xs:element>

</xs:schema>
```

References

Antonanzas, J. et al. (2016). “Review of photovoltaic power forecasting”. In: *Solar Energy* 136, pp. 78–111. DOI: 10.1016/j.solener.2016.06.069.

Apache Software Foundation. *Apache Xerces Project*. [Accessed 25 March 2017]. URL: http://xerces.apache.org/.

Archetti, Francesco et al. (2006). “Genetic programming for human oral bioavailability of drugs”. In: *8th Annual Conference on Genetic and Evolutionary Computation (GECCO ’06)*. Seattle, Washington, USA, pp. 255–262. DOI: 10.1145/1143997.1144042.

ASHRAE (2009). *Standard for the Design of High-performance Green Buildings except Low-rise Residential Buildings,* tech. rep. ASHRAE 189.1. Atlanta GA: American Society of Heating, Refrigerating and Air-conditioning Engineers.

Azad, Raja Muhammad Atif and Conor Ryan (2014). “A simple approach to lifetime learning in genetic programming-based symbolic regression”. In: *Evol. Comput.* 22.2, pp. 287–317. DOI: 10.1162/EVCO_a_00111.

Beadle, Lawrence (2009). “Semantic and Structural Analysis of Genetic Programming”. PhD thesis. University of Kent, pp. 182–196.

Beadle, Lawrence and Colin G. Johnson (2009). “Semantically driven mutation in genetic programming”. In: *11th Conference on Congress on Evolutionary Computation (CEC’09)*. Trondheim, Norway, pp. 1336–1342.

Benavoli, Alessio, Giorgio Corani, and Francesca Mangili (2016). “Should we really use post-hoc tests based on mean-ranks?” In: *J. Mach. Learn. Res.* 17.1, pp. 152–161.

Besselink, B. et al. (2013). “A comparison of model reduction techniques from structural dynamics, numerical mathematics and systems and control”. In: *Journal of Sound and Vibration* 332.19, pp. 4403 –4422. ISSN: 0022-460X. DOI: https://doi.org/10.1016/j.jsv.2013.03.025. URL: http://www.sciencedirect.com/science/article/pii/S0022460X1300285X.

Billings, Stephen A (2013). *Nonlinear system identification: NARMAX methods in the time, frequency, and spatio-temporal domains*. John Wiley & Sons.

Blochwitz, T. et al. (2011). "The Functional Mockup Interface for tool independent exchange of simulation models". In: *8th International Modelica Conference*. Dresden, Germany, pp. 105–114. DOI: 10.3384/ecp11063105.

Bloemen, H. H. J., T. J. J. van den Boom, and H. B. Verbruggen (2000). "Model-based predictive control for Hammerstein systems". In: *Proceedings of the 39th IEEE Conference on Decision and Control (Cat. No.00CH37187)*. Vol. 5, 4963–4968 vol.5. DOI: 10.1109/CDC.2001.914719.

Bloemen, H. H. J., T. J. J. Van Den Boom, and H. B. Verbruggen (2001). "Model-based predictive control for Hammerstein?Wiener systems". In: *International Journal of Control* 74.5, pp. 482–495. DOI: 10.1080/00207170010014061. eprint: https://doi.org/10.1080/00207170010014061. URL: https://doi.org/10.1080/00207170010014061.

Brameier, M. F. and W. Banzhaf (2010). *Linear Genetic Programming*. Springer.

Camacho, Eduardo F and Carlos Bordons (2004). *Model Predictive Control*. 2nd. London: Springer.

Cao, G., E. M. . Lai, and F. Alam (2017). "Gaussian process model predictive control of unknown non-linear systems". In: *IET Control Theory Applications* 11.5, pp. 703–713. ISSN: 1751-8644. DOI: 10.1049/iet-cta.2016.1061.

Cao, G., E. M. Lai, and F. Alam (2016a). "Gaussian Process Model Predictive Control of unmanned quadrotors". In: *2016 2nd International Conference on Control, Automation and Robotics (ICCAR)*, pp. 200–206. DOI: 10.1109/ICCAR.2016.7486726.

Cao, G., E. M-K Lai, and F. Alam (2016b). "Gaussian Process based Model Predictive Control for Linear Time Varying systems". In: *2016 IEEE 14th International Workshop on Advanced Motion Control (AMC)*, pp. 251–256. DOI: 10.1109/AMC.2016.7496359.

Cao, Yilong and Peter I. Rockett (2015). "The use of vicinal-risk minimization for training decision trees". In: *Applied Soft Computing* 31.0, pp. 185–195. DOI: 10.1016/j.asoc.2015.02.043.

Castelli, Mauro, Sara Silva, and Leonardo Vanneschi (2015). "A C++ framework for geometric semantic genetic programming". In: *Gen. Prog. Evol. Mach.* 16.1, pp. 73–81. DOI: 10.1007/s10710-014-9218-0.

Castelli, Mauro, Leonardo Trujillo, and Leonardo Vanneschi (2015). "Energy consumption forecasting using semantic-based genetic programming with local search optimizer". In: *Intell. Neuroscience* 2015, 57:57–57:57. DOI: 10.1155/2015/971908.

Castelli, Mauro et al. (2015). "Geometric semantic genetic programming with local search". In: *Annual Conference on Genetic and Evolutionary Computation (GECCO '15)*. Madrid, Spain, pp. 999–1006. DOI: 10.1145/2739480.2754795.

Castro, Elizabeth (2001). *XML for the World Wide Web*. Berkeley, CA: Peachpit Press.

Chan, Felix and Laurent L. Pauwels (2018). "Some theoretical results on forecast combinations". In: *International Journal of Forecasting* 34.1, pp. 64 –74. ISSN: 0169-2070. DOI: https://doi.org/10.1016/j.ijforecast.2017.08.005. URL: http://www.sciencedirect.com/science/article/pii/S0169207017300912.

Chan, Kwong Ho and Jie Bao (2007). "Model Predictive Control of Hammerstein Systems with Multivariable Nonlinearities". In: *Industrial & Engineering Chemistry Research* 46.1, pp. 168–180. DOI: 10.1021/ie0609113. eprint: https://doi.org/10.1021/ie0609113. URL: https://doi.org/10.1021/ie0609113.

Chen, C. (2011). "Robust Self-Organizing Neural-Fuzzy Control With Uncertainty Observer for MIMO Nonlinear Systems". In: *IEEE Transactions on Fuzzy Systems* 19.4, pp. 694–706. ISSN: 1063-6706. DOI: 10.1109/TFUZZ.2011.2136349.

Chen, Q., M. Zhang, and B. Xue (2017). "Feature selection to improve generalization of genetic programming for high-dimensional symbolic regression". In: *IEEE T. Evol. Comp.* 21.5, pp. 792–806. DOI: 10.1109/TEVC.2017.2683489.

Cheng, C.M. et al. (2017). "Volterra-series-based nonlinear system modeling and its engineering applications: A state-of-the-art review". In: *Mechanical Systems and Signal Processing* 87, pp. 340 –364. ISSN: 0888-3270. DOI: https://doi.org/10.1016/j.ymssp.2016.10.029. URL: http://www.sciencedirect.com/science/article/pii/S0888327016304393.

Cherkassky, Vladimir and Filip M Mulier (2007). *Learning from Data: Concepts, Theory and Methods*. 2nd. Wiley-IEEE Press.

CIBSE (2015). *CIBSE Guide A: Environmental Design*. Tech. rep. London, UK: The Chartered Institution of Building Services Engineers.

Coello Coello, Carlos A. (1999). "A Comprehensive Survey of Evolutionary-Based Multiobjective Optimization Techniques". In: *Knowledge and Information Systems* 1.3, pp. 269–

308. ISSN: 0219-3116. DOI: 10.1007/BF03325101. URL: https://doi.org/10.1007/BF03325101.

Corne, David et al. (2013). "Short term wind speed forecasting with evolved neural networks". In: *15th Annual Conference Companion on Genetic and Evolutionary Computation*. Amsterdam, The Netherlands, pp. 1521–1528. DOI: 10.1145/2464576.2482731.

Costa, J. ., A. Lagrange, and A. Arliaud (2003). "Acoustic echo cancellation using nonlinear cascade filters". In: *2003 IEEE International Conference on Acoustics, Speech, and Signal Processing, 2003. Proceedings. (ICASSP '03).* Vol. 5, pp. V–389. DOI: 10.1109/ICASSP.2003.1199972.

Costelloe, Dan and Conor Ryan (2009). "On Improving Generalisation in Genetic Programming". In: *Genetic Programming*. Ed. by Leonardo Vanneschi et al. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 61–72. ISBN: 978-3-642-01181-8.

Cramer, Nichael Lynn (1985). "A Representation for the Adaptive Generation of Simple Sequential Programs". In: *Proceedings of the 1st International Conference on Genetic Algorithms*. Hillsdale, NJ, USA: L. Erlbaum Associates Inc., pp. 183–187. ISBN: 0-8058-0426-9. URL: http://dl.acm.org/citation.cfm?id=645511.657085.

Crawford-Marks, Raphael and Lee Spector (2002). "Size Control Via Size Fair Genetic Operators In The PushGP Genetic Programming System". In: *GECCO*.

Crawley, Drury B. et al. (2000). "EnergyPlus: Energy simulation program". In: *ASHRAE Journal* 42.4, pp. 49–56.

Dabhi, Vipul K. and Sanjay Chaudhary (2015). "Empirical modeling using genetic programming: a survey of issues and approaches". In: *Natural Computing* 14.2, pp. 303–330. ISSN: 1572-9796. DOI: 10.1007/s11047-014-9416-y. URL: https://doi.org/10.1007/s11047-014-9416-y.

Deb, Kalyanmoy et al. (2000). "A Fast Elitist Non-dominated Sorting Genetic Algorithm for Multi-objective Optimization: NSGA-II". In: *Parallel Problem Solving from Nature PPSN VI*. Ed. by Marc Schoenauer et al. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 849–858. ISBN: 978-3-540-45356-7.

Demšar, Janez (2006). "Statistical comparisons of classifiers over multiple data sets". In: *J. Mach. Learn. Res.* 7, pp. 1–30.

Dignum, Stephen and Riccardo Poli (2008). "Operator Equalisation and Bloat Free GP". In: *Genetic Programming*. Ed. by Michael O'Neill et al. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 110–121. ISBN: 978-3-540-78671-9.

Dou, Tiantian, Yuri Kaszubowski Lopes, and Peter Rockett (2018). *GPML: An XML-based standard for the interchange of genetic programming trees*. Technical Report. Dept. of Electronic and Electrical Engineering, University of Sheffield. URL: http://eprints.whiterose.ac.uk/134140/.

Dou, Tiantian and Peter Rockett (2018). "Comparison of semantic-based local search methods for multiobjective genetic programming". In: *Genetic Programming and Evolvable Machines*. , In press. DOI: 10.1007/s10710-018-9325-4.

Du, Jingjing, Chunyue Song, and Ping Li (2009). "Multilinear Model Control of Hammerstein-like Systems Based on an Included Angle Dividing Method and the MLD-MPC Strategy". In: *Industrial & Engineering Chemistry Research* 48.8, pp. 3934–3943. DOI: 10.1021/ie8009395. eprint: https://doi.org/10.1021/ie8009395. URL: https://doi.org/10.1021/ie8009395.

Durillo, Juan J. et al. (2009). "On the effect of the steady-state selection scheme in multi-objective genetic algorithms". In: *5th International Conference Evolutionary Multi-Criterion Optimization (EMO 2009)*. Nantes, France, pp. 183–197. DOI: 10.1007/978-3-642-01020-0_18.

Fairbank, Michael et al. (2014). "An adaptive recurrent neural-network controller using a stabilization matrix and predictive inputs to solve a tracking problem under disturbances". In: *Neural Networks* 49, pp. 74 –86. ISSN: 0893-6080. DOI: https://doi.org/10.1016/j.neunet.2013.09.010. URL: http://www.sciencedirect.com/science/article/pii/S0893608013002438.

Feng, Qi, Haowei Lian, and Jindong Zhu (2017). "Model predictive control of nonlinear dynamical systems based on genetic programming". In: *36th Chinese Control Conference (CCC)*. Dalin, China, pp. 4540–4545. DOI: 10.23919/ChiCC.2017.8028072.

Ferreira, C. (2002). "Gene Expression Programming in Problem Solving". In: *Soft Computing and Industry: Recent Applications*. Ed. by Rajkumar Roy et al. Springer London, pp. 635–653.

Ffrancon, Robyn and Marc Schoenauer (2015). "Memetic semantic genetic programming". In: *Conference on Genetic and Evolutionary Computation (GECCO '15)*. Madrid, Spain, pp. 1023–1030. DOI: 10.1145/2739480.2754697.

Fonseca, Carlos and Peter J Fleming (1998). "Multiobjective optimization and multiple constraint handling with evolutionary algorithms - Part I: A unified formulation". In: *IEEE Transactions on Systems, Man & Cybernetics - Part A: Systems & Humans* 28.1, pp. 26–37. DOI: 10.1109/3468.650319.

Fonseca, Carlos M. and Peter J. Fleming (1993). "Genetic Algorithms for Multiobjective Optimization: FormulationDiscussion and Generalization". In: *Proceedings of the 5th International Conference on Genetic Algorithms*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., pp. 416–423. ISBN: 1-55860-299-2. URL: http://dl.acm.org/citation.cfm?id=645513.657757.

Friedberg, R. M. (1958). "A Learning Machine: Part I". In: *IBM Journal of Research and Development* 2.1, pp. 2–13. ISSN: 0018-8646. DOI: 10.1147/rd.21.0002.

Fruzzetti, K.P., A. Palazoğlu, and K.A. McDonald (1997). "Nolinear model predictive control using Hammerstein models". In: *Journal of Process Control* 7.1, pp. 31 –41. ISSN: 0959-1524. DOI: https://doi.org/10.1016/S0959-1524(97)80001-B. URL: http://www.sciencedirect.com/science/article/pii/S095915249780001B.

Fu, Z. et al. (2013). "Nonlinear Systems Identification and Control Via Dynamic Multitime Scales Neural Networks". In: *IEEE Transactions on Neural Networks and Learning Systems* 24.11, pp. 1814–1823. ISSN: 2162-237X. DOI: 10.1109/TNNLS.2013.2265604.

Gagné, Christian et al. (2006). "Genetic Programming, Validation Sets, and Parsimony Pressure". In: *Genetic Programming*. Ed. by Pierre Collet et al. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 109–120. ISBN: 978-3-540-33144-5.

Galván-López, E., M. O'Neill, and A. Brabazon (2009). "Towards understanding the effects of locality in GP". In: *8th Mexican International Conference on Artificial Intelligence*, pp. 9–14. DOI: 10.1109/MICAI.2009.17.

Gandomi, Amir Hossein and Amir Hossein Alavi (2011). "Applications of Computational Intelligence in Behavior Simulation of Concrete Materials". In: *Computational Optimization and Applications in Engineering and Industry*. Ed. by Xin-She Yang and Slawomir

Koziel. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 221–243. ISBN: 978-3-642-20986-4. DOI: 10.1007/978-3-642-20986-4_9. URL: https://doi.org/10.1007/978-3-642-20986-4_9.

Gevers, Michel (2005). "Identification for control: From the early achievements to the revival of experiment design". In: *European Journal of Control* 11.4-5, pp. 335–352. DOI: 10.3166/ejc.11.335-352.

Giraud-Carrier, Christophe (2002). "Unifying Learning with Evolution Through Baldwinian Evolution and Lamarckism". In: *Advances in Computational Intelligence and Learning: Methods and Applications*. Ed. by Hans-Jürgen Zimmermann et al. Dordrecht: Springer Netherlands, pp. 159–168. DOI: 10.1007/978-94-010-0324-7_11.

Goldberg, David E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. 1st. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc. ISBN: 0201157675.

Gonçalves, Ivo et al. (2016). "Arbitrarily Close Alignments in the Error Space: A Geometric Semantic Genetic Programming Approach". In: *Genetic and Evolutionary Computation Conference Companion*. GECCO '16 Companion. Denver, CO, pp. 99–100. DOI: 10.1145/2908961.2908988.

Gotmare, Akhilesh, Rohan Patidar, and Nithin V. George (2015). "Nonlinear system identification using a cuckoo search optimized adaptive Hammerstein model". In: *Expert Systems with Applications* 42.5, pp. 2538 –2546. ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2014.10.040. URL: http://www.sciencedirect.com/science/article/pii/S0957417414006642.

Grancharova, A., J. Kocijan, and T. A. Johansen (2007). "Explicit stochastic Nonlinear Predictive Control based on Gaussian process models". In: *2007 European Control Conference (ECC)*, pp. 2340–2347. DOI: 10.23919/ECC.2007.7068422.

Grosman, Benyamin and Daniel R Lewin (2002). "Automated nonlinear model predictive control using genetic programming". In: *Computers and Chemical Engineering* 26.4-5, pp. 631–640. DOI: 10.1016/S0098-1354(01)00780-3.

Gruau, Frédéric and Darrell Whitley (1993). "Adding Learning to the Cellular Development of Neural Networks: Evolution and the Baldwin Effect". In: *Evol. Comput.* 1.3, pp. 213–233. DOI: 10.1162/evco.1993.1.3.213.

Gruber, J.K., C. Bordons, and A. Oliva (2012). "Nonlinear MPC for the airflow in a PEM fuel cell using a Volterra series model". In: *Control Engineering Practice* 20.2, pp. 205 –

217. ISSN: 0967-0661. DOI: https://doi.org/10.1016/j.conengprac.2011.10.014. URL: http://www.sciencedirect.com/science/article/pii/S0967066111002322.

Gruber, J.K. et al. (2011). "Nonlinear MPC based on a Volterra series model for greenhouse temperature control using natural ventilation". In: *Control Engineering Practice* 19.4, pp. 354 –366. ISSN: 0967-0661. DOI: https://doi.org/10.1016/j.conengprac.2010.12.004. URL: http://www.sciencedirect.com/science/article/pii/S0967066110002649.

Gruber, J.K. et al. (2013). "Computationally efficient nonlinear Min–Max Model Predictive Control based on Volterra series models—Application to a pilot plant". In: *Journal of Process Control* 23.4, pp. 543 –560. ISSN: 0959-1524. DOI: https://doi.org/10.1016/j.jprocont.2013.01.007. URL: http://www.sciencedirect.com/science/article/pii/S0959152413000176.

Guven, Aytac (2011). "A multi-output descriptive neural network for estimation of scour geometry downstream from hydraulic structures". In: *Advances in Engineering Software* 42.3, pp. 85 –93. ISSN: 0965-9978. DOI: https://doi.org/10.1016/j.advengsoft.2010.12.005. URL: http://www.sciencedirect.com/science/article/pii/S0965997810001699.

Guzelbey, Ibrahim H., Abdulkadir Cevik, and Ahmet Erklig (2006). "Prediction of web crippling strength of cold-formed steel sheetings using neural networks". In: *Journal of Constructional Steel Research* 62.10, pp. 962 –973. ISSN: 0143-974X. DOI: https://doi.org/10.1016/j.jcsr.2006.01.008. URL: http://www.sciencedirect.com/science/article/pii/S0143974X06000174.

Guzelbey, Ibrahim H., Abdulkadir Cevik, and Mehmet Tolga Gögüş (2006). "Prediction of rotation capacity of wide flange beams using neural networks". In: *Journal of Constructional Steel Research* 62.10, pp. 950 –961. ISSN: 0143-974X. DOI: https://doi.org/10.1016/j.jcsr.2006.01.003. URL: http://www.sciencedirect.com/science/article/pii/S0143974X06000150.

Han, H. and J. Qiao (2014). "Nonlinear Model-Predictive Control for Industrial Processes: An Application to Wastewater Treatment Process". In: *IEEE Transactions on Industrial Electronics* 61.4, pp. 1970–1982. ISSN: 0278-0046. DOI: 10.1109/TIE.2013.2266086.

Han, Hong-Gui and Jun-Fei Qiao (2011). "Adaptive Dissolved Oxygen Control Based on Dynamic Structure Neural Network". In: *Appl. Soft Comput.* 11.4, pp. 3812–3820. ISSN: 1568-4946. DOI: 10.1016/j.asoc.2011.02.014. URL: http://dx.doi.org/10.1016/j.asoc.2011.02.014.

Harries, Kim and Peter Smith (1997). "Exploring alternative operators and search strategies in genetic programming". In: *2nd Annual Conference on Genetic Programming*, pp. 147–155.

Haykin, Simon (1994). *Neural Networks: A Comprehensive Foundation*. 1st. Upper Saddle River, NJ, USA: Prentice Hall PTR. ISBN: 0023527617.

Haynes, Thomas (1998). "Collective Adaptation: The Exchange of Coding Segments". In: *Evolutionary Computation* 6, pp. 311–338.

Henze, Gregor P. (2013). "Editorial – Model predictive control for buildings: A quantum leap?" In: *Journal of Building Performance Simulation* 6.3. Special Issue on Model Predictive Control for Buildings, pp. 157–158. DOI: 10.1080/19401493.2013.778519.

Hinchliffe, Mark P. and Mark J. Willis (2003). "Dynamic systems modelling using genetic programming". In: *Computers and Chemical Engineering* 27.12, pp. 1841–1854. DOI: 10.1016/j.compchemeng.2003.06.001.

Holland, John H. (1992). *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. Cambridge, MA, USA: MIT Press. ISBN: 0262082136.

Horn, J., N. Nafpliotis, and D. E. Goldberg (1994). "A niched Pareto genetic algorithm for multiobjective optimization". In: *Proceedings of the First IEEE Conference on Evolutionary Computation. IEEE World Congress on Computational Intelligence*, 82–87 vol.1. DOI: 10.1109/ICEC.1994.350037.

Hosen, Mohammad Anwar, Mohd Azlan Hussain, and Farouq S. Mjalli (2011). "Control of polystyrene batch reactors using neural network based model predictive control (NN-MPC): An experimental investigation". In: *Control Engineering Practice* 19.5, pp. 454 – 467. ISSN: 0967-0661. DOI: https://doi.org/10.1016/j.conengprac.2011.01.007. URL: http://www.sciencedirect.com/science/article/pii/S0967066111000177.

Huo, Hai-Bo et al. (2008). "Nonlinear model predictive control of SOFC based on a Hammerstein model". In: *Journal of Power Sources* 185.1, pp. 338 –344. ISSN: 0378-7753.

DOI: https://doi.org/10.1016/j.jpowsour.2008.06.064. URL: http://www.sciencedirect.com/science/article/pii/S0378775308012706.

Hussain, Mohamed Azlan (1999). "Review of the applications of neural networks in chemical process control - simulation and online implementation". In: *Artificial Intelligence in Engineering* 13.1, pp. 55–68. DOI: 10.1016/S0954-1810(98)00011-9.

Iba, Hitoshi, Hugo de Garis, and Taisuke Sato (1994). "Genetic programming with local hill-climbing". In: *3rd Conference on Parallel Problem Solving from Nature (PPSN III): International Conference on Evolutionary Computation*. Jerusalem, Israel, pp. 302–311. DOI: 10.1007/3-540-58484-6_274.

Igel, Christian and Kumar Chellapilla (1999). "Investigating the influence of depth and degree of genotypic change on fitness in genetic programming". In: *1st Annual Conference on Genetic and Evolutionary Computation (GECCO'99) – Volume 2*. Orlando, Florida, pp. 1061–1068.

ISO/IEC 14977:1996 (1996). *Information technology – Syntactic metalaguage – Extended BNF*. ISO/IEC 14977:1996, Geneva, Switzerland.

Ito, Takuya, Hitoshi Iba, and Satoshi Sato (1998). "Non-destructive depth-dependent crossover for genetic programming". In: *1st European Workshop on Genetic Programming (EuroGP '98)*. London, UK, pp. 71–82.

Jackson, David (2010). "Promoting phenotypic diversity in genetic programming". In: *11th International Conference on Parallel Problem Solving from Nature: Part II (PPSN'10)*, pp. 472–481.

Jalaleddini, K. and R. E. Kearney (2013). "Subspace Identification of SISO Hammerstein Systems: Application to Stretch Reflex Identification". In: *IEEE Transactions on Biomedical Engineering* 60.10, pp. 2725–2734. ISSN: 0018-9294. DOI: 10.1109/TBME.2013.2264216.

Jeraj, J. and V. J. Mathews (2006). "Stochastic mean-square performance analysis of an adaptive Hammerstein filter". In: *IEEE Transactions on Signal Processing* 54.6, pp. 2168–2177. ISSN: 1053-587X. DOI: 10.1109/TSP.2006.873587.

Jeraj, Janez, V. John Mathews, and Joel Dubow (2002). "A stable adaptive Hammerstein filter employing partial orthogonalization of the input signals". In: *IEEE Transactions on Signal Processing* 54, pp. 1412–1420.

Jin, Yaochu, Markus Olhofer, and Bernhard Sendhoff (2001). "Dynamic Weighted Aggregation for Evolutionary Multi-objective Optimization: Why Does It Work and How?" In: *Proceedings of the 3rd Annual Conference on Genetic and Evolutionary Computation*. GECCO'01. San Francisco, California: Morgan Kaufmann Publishers Inc., pp. 1042–1049. ISBN: 1-55860-774-9. URL: http://dl.acm.org/citation.cfm?id=2955239.2955427.

Juárez-Smith, Perla and Leonardo Trujillo (2016). "Integrating local search within neat-GP". In: *Genetic and Evolutionary Computation Conference Companion*. Denver, CO, pp. 993–996. DOI: 10.1145/2908961.2931659.

Klenske, E. D. et al. (2016). "Gaussian Process-Based Predictive Control for Periodic Error Correction". In: *IEEE Transactions on Control Systems Technology* 24.1, pp. 110–121. ISSN: 1063-6536. DOI: 10.1109/TCST.2015.2420629.

Knowles, Joshua D. and David W. Corne (2000). "Approximating the Nondominated Front Using the Pareto Archived Evolution Strategy". In: *Evol. Comput.* 8.2, pp. 149–172. ISSN: 1063-6560. DOI: 10.1162/106365600568167. URL: http://dx.doi.org/10.1162/106365600568167.

Kocijan, J. et al. (2004). "Gaussian process model based predictive control". In: *Proceedings of the 2004 American Control Conference*. Vol. 3, 2214–2219 vol.3. DOI: 10.23919/ACC.2004.1383790.

Kocijan, Juš (2016a). "System Identification with GP Models". In: *Modelling and Control of Dynamic Systems Using Gaussian Process Models*. Cham: Springer International Publishing. Chap. 2, pp. 21–102. ISBN: 978-3-319-21021-6. DOI: 10.1007/978-3-319-21021-6_2. URL: https://doi.org/10.1007/978-3-319-21021-6_2.

Kocijan, Juš (2016b). *Modelling and control of dynamic systems using gaussian process models*. Advances in industrial control. Cham: Springer. URL: http://cds.cern.ch/record/2112684.

Kotanchek, Mark, Guido Smits, and Ekaterina Vladislavleva (2008). "Trustable symbolic regression models: using ensembles, interval arithmetic and pareto fronts to develop robust and trust-aware models". In: *Genetic Programming Theory and Practice V*. Ed. by Rick Riolo, Terence Soule, and Bill Worzel. Boston, MA: Springer US, pp. 201–220. ISBN: 978-0-387-76308-8. DOI: 10.1007/978-0-387-76308-8_12. URL: https://doi.org/10.1007/978-0-387-76308-8_12.

Koza, John R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, MA, USA: MIT Press. ISBN: 0-262-11170-5.

Krasnogor, N. (2004). "Self generating metaheuristics in bioinformatics: The proteins structure comparison case". In: *Genet. Program. Evol. Mach.* 5.2, pp. 181–201. DOI: 10.1023/B:GENP.0000023687.41210.d7.

Krawiec, Krzysztof (2001). "Genetic programming with local improvement for visual learning from examples". In: *Computer Analysis of Images and Patterns, 9th International Conference, CAIP 2001 Warsaw, Poland, September 5-7, 2001, Proceedings*. Warsaw, Poland, pp. 209–216. DOI: 10.1007/3-540-44692-3_26.

Krawiec, Krzysztof and Pawel Lichocki (2009). "Approximating geometric crossover in semantic space". In: *11^{th} Annual Conference on Genetic and Evolutionary Computation (GECCO '09)*. Montreal, Canada, pp. 987–994. DOI: 10.1145/1569901.1570036.

Kumar, Divya and Hector Budman (2014). "Robust nonlinear MPC based on Volterra series and polynomial chaos expansions". In: *Journal of Process Control* 24.1, pp. 304 – 317. ISSN: 0959-1524. DOI: https://doi.org/10.1016/j.jprocont.2013.03.003. URL: http://www.sciencedirect.com/science/article/pii/S095915241300036X.

La Cava, William et al. (2015). "Genetic programming with epigenetic local search". In: *Annual Conference on Genetic and Evolutionary Computation (GECCO '15)*. Madrid, Spain, pp. 1055–1062. DOI: 10.1145/2739480.2754763.

Langdon, W. B. and R. Poli (1998). "Fitness causes bloat: Mutation". In: *Genetic Programming*. Ed. by Wolfgang Banzhaf et al. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 37–48. ISBN: 978-3-540-69758-9.

Ławryńczuk, Maciej (2013). "Nonlinear Predictive Control Based on Least Squares Support Vector Machines Hammerstein Models". In: *Adaptive and Natural Computing Algorithms*. Ed. by Marco Tomassini et al. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 246–255. ISBN: 978-3-642-37213-1.

Le, N. et al. (2016). "Complexity measures in genetic programming learning: A brief review". In: *IEEE Congress on Evolutionary Computation (CEC)*, pp. 2409–2416. DOI: 10.1109/CEC.2016.7744087.

Li, Han-Xiong, Chenkun Qi, and Yongguang Yu (2009). "A spatio-temporal Volterra modeling approach for a class of distributed industrial processes". In: *Journal of Process*

Control 19.7, pp. 1126 –1142. ISSN: 0959-1524. DOI: https://doi.org/10.1016/j.jprocont.2009.02.005. URL: http://www.sciencedirect.com/science/article/pii/S0959152409000377.

Lu, C. and C. Tsai (2008). "Adaptive Predictive Control With Recurrent Neural Network for Industrial Processes: An Application to Temperature Control of a Variable-Frequency Oil-Cooling Machine". In: *IEEE Transactions on Industrial Electronics* 55.3, pp. 1366–1375. ISSN: 0278-0046. DOI: 10.1109/TIE.2007.896492.

Luke, Sean (2003). "Modification Point Depth and Genome Growth in Genetic Programming". In: *Evolutionary Computation* 11.1, pp. 67–106. DOI: 10.1162/106365603321829014. eprint: https://doi.org/10.1162/106365603321829014. URL: https://doi.org/10.1162/106365603321829014.

Majeed, Hammad (2007). "The importance of semantic context in tree based GP and its application in defining a less destructive, context aware crossover for GP". PhD thesis. University of Limerick.

Maner, Bryon R. et al. (1996). "Nonlinear model predictive control of a simulated multivariable polymerization reactor using second-order Volterra models". In: *Automatica* 32.9, pp. 1285 –1301. ISSN: 0005-1098. DOI: https://doi.org/10.1016/0005-1098(96)00086-6. URL: http://www.sciencedirect.com/science/article/pii/0005109896000866.

Mastromauro, R. A., M. Liserre, and A. Dell'Aquila (2008). "Study of the Effects of Inductor Nonlinear Behavior on the Performance of Current Controllers for Single-Phase PV Grid Converters". In: *IEEE Transactions on Industrial Electronics* 55.5, pp. 2043–2052. ISSN: 0278-0046. DOI: 10.1109/TIE.2008.917117.

Mathworks Inc. *XML Documents*. [Accessed 25 March 2017]. URL: http://www.mathworks.com/help/matlab/xml-documents.html.

McDermott, James et al. (2012). "Genetic Programming Needs Better Benchmarks". In: *14th Conference on Genetic and Evolutionary Computation*. New York, NY, USA, pp. 791–798. DOI: 10.1145/2330163.2330273.

Mcphee, Nicholas Freitag and Justin Darwin Miller (1995). "Accurate Replication in Genetic Programming". In: *Genetic Algorithms: Proceedings of the Sixth International Conference (ICGA95*. Morgan Kaufmann, pp. 303–309.

McPhee, Nicholas Freitag, Brian Ohs, and Tyler Hutchison (2008). "Semantic building blocks in genetic programming". In: *11th European Conference on Genetic Programming (EuroGP'08)*. Naples, Italy, pp. 134–145.

Menold, P.H., F. Allgöwer, and R.K. Pearson (1997). "Nonlinear structure identification of chemical processes". In: *Computers & Chemical Engineering* 21. Supplement to Computers and Chemical Engineering, S137 –S142. ISSN: 0098-1354. DOI: https://doi.org/10.1016/S0098-1354(97)87492-3. URL: http://www.sciencedirect.com/science/article/pii/S0098135497874923.

Mete, Selcuk, Saban Ozer, and Hasan Zorlu (2016). "System identification using Hammerstein model optimized with differential evolution algorithm". In: *AEU - International Journal of Electronics and Communications* 70.12, pp. 1667 –1675. ISSN: 1434-8411. DOI: https://doi.org/10.1016/j.aeue.2016.10.005. URL: http://www.sciencedirect.com/science/article/pii/S1434841116309323.

Metenidis, Mihai Florin, Marcin Witczak, and Józef Korbicz (2004). "A novel genetic programming approach to nonlinear system modelling: application to the DAMADICS benchmark problem". In: *Engineering Applications of Artificial Intelligence* 17.4. Selected Problems of Knowledge Representation, pp. 363 –370. ISSN: 0952-1976. DOI: https://doi.org/10.1016/j.engappai.2004.04.009. URL: http://www.sciencedirect.com/science/article/pii/S095219760400048X.

Miller, J. F., ed. (2011). *Cartesian Genetic Programming*. Springer.

Montaña, José L. et al. (2011). "Penalty Functions for Genetic Programming Algorithms". In: *Computational Science and Its Applications - ICCSA 2011*. Ed. by Beniamino Murgante et al. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 550–562. ISBN: 978-3-642-21928-3.

Moraglio, Alberto, Krzysztof Krawiec, and Colin G. Johnson (2012). "Geometric semantic genetic programming". In: *12th International Conference on Parallel Problem Solving from Nature - Volume Part I (PPSN'12)*. Taormina, Italy, pp. 21–31. DOI: 10.1007/978-3-642-32937-1_3.

Moscato, Pablo et al. (1989). "On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms". In: *Caltech Concurrent Computation Program, C3P Report* 826, p. 1989.

Na, Jing et al. (2012). "Adaptive neural network predictive control for nonlinear pure feedback systems with input delay". In: *Journal of Process Control* 22.1, pp. 194 –206. ISSN: 0959-1524. DOI: https://doi.org/10.1016/j.jprocont.2011.09.003. URL: http://www.sciencedirect.com/science/article/pii/S0959152411001880.

Nelles, Oliver (2001). *Nonlinear System Identification: From Classical Approaches to Neural Networks and Fuzzy Models*. Berlin: Springer.

Neri, Ferrante, Carlos Cotta, and Pablo Moscato (2012). *Handbook of Memetic Algorithms*. Vol. 379. Springer.

Ngatchou, P., A. Zarei, and A. El-Sharkawi (2005). "Pareto Multi Objective Optimization". In: *Proceedings of the 13th International Conference on, Intelligent Systems Application to Power Systems*, pp. 84–91. DOI: 10.1109/ISAP.2005.1599245.

Ni, J., R. H. Drieberg, and P. I. Rockett (2013). "The use of an analytic quotient operator in genetic programming". In: *IEEE T. Evolut. Comput.* 17.1, pp. 146–152. DOI: 10.1109/TEVC.2012.2195319.

Ni, Ji and Peter Rockett (2015). "Tikhonov regularization as a complexity measure in multiobjective genetic programming". In: *IEEE T. Evolut. Comput.* 19.2, pp. 157–166.

Nikolaev, N. Y. and H. Iba (2001). "Regularization approach to inductive genetic programming". In: *IEEE Transactions on Evolutionary Computation* 5.4, pp. 359–375. ISSN: 1089-778X. DOI: 10.1109/4235.942530.

Orcioni, Simone (2014). "Improving the approximation ability of Volterra series identified with a cross-correlation method". In: *Nonlinear Dynamics* 78.4, pp. 2861–2869. ISSN: 1573-269X. DOI: 10.1007/s11071-014-1631-7. URL: https://doi.org/10.1007/s11071-014-1631-7.

Ozer, Saban, Hasan Zorlu, and Selcuk Mete (2016). "System identification application using Hammerstein model". In: *Sādhanā* 41.6, pp. 597–605. ISSN: 0973-7677. DOI: 10.1007/s12046-016-0505-8. URL: https://doi.org/10.1007/s12046-016-0505-8.

Pala, Murat (2006). "A new formulation for distortional buckling stress in cold-formed steel members". In: *Journal of Constructional Steel Research* 62.7, pp. 716 –722. ISSN: 0143-974X. DOI: https://doi.org/10.1016/j.jcsr.2005.09.011. URL: http://www.sciencedirect.com/science/article/pii/S0143974X05001768.

Pan, Y. and J. Wang (2012). "Model Predictive Control of Unknown Nonlinear Dynamical Systems Based on Recurrent Neural Networks". In: *IEEE Transactions on Industrial Electronics* 59.8, pp. 3089–3101. ISSN: 0278-0046. DOI: 10.1109/TIE.2011.2169636.

Patwardhan, Rohit S., S. Lakshminarayanan, and Sirish L. Shah (1998). "Constrained nonlinear MPC using Hammerstein and Wiener models: PLS framework". English. In: *American Institute of Chemical Engineers.AIChE Journal* 44.7, p. 1611. URL: https://search.proquest.com/docview/199371830?accountid=13828.

Pawlak, T. P., B. Wieloch, and K. Krawiec (2015). "Semantic backpropagation for designing search operators in genetic programming". In: *IEEE T. Evolut. Comput.* 19.3, pp. 326–340. DOI: 10.1109/TEVC.2014.2321259.

Pearson, R.K. (1999). *Discrete-time Dynamic Models*. Oxford University Press.

Phan, M. C. and M. T. Hagan (2013). "Error Surface of Recurrent Neural Networks". In: *IEEE Transactions on Neural Networks and Learning Systems* 24.11, pp. 1709–1721. ISSN: 2162-237X. DOI: 10.1109/TNNLS.2013.2258470.

Poli, Riccardo (2000). "Why the schema theorem is correct also in the presence of stochastic effects". In: *Evolutionary Computation, 2000. Proceedings of the 2000 Congress on*. Vol. 1. IEEE, pp. 487–492.

— (2003). "A Simple but Theoretically-motivated Method to Control Bloat in Genetic Programming". In: *Proceedings of the 6th European Conference on Genetic Programming*. EuroGP'03. Essex, UK: Springer-Verlag, pp. 204–217. ISBN: 3-540-00971-X. URL: http://dl.acm.org/citation.cfm?id=1762668.1762688.

Poli, Riccardo, William B. Langdon, and Stephen Dignum (2007). "On the Limiting Distribution of Program Sizes in Tree-Based Genetic Programming". In: *Genetic Programming*. Ed. by Marc Ebner et al. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 193–204. ISBN: 978-3-540-71605-1.

Poli, Riccardo, William B. Langdon, and Nicholas Freitag McPhee (2008). *A Field Guide to Genetic Programming*. Lulu Enterprises, UK Ltd. ISBN: 1409200736, 9781409200734.

Poli, Riccardo and Nicholas Freitag McPhee (2008). "Parsimony Pressure Made Easy". In: *Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation*. GECCO '08. Atlanta, GA, USA: ACM, pp. 1267–1274. ISBN: 978-1-60558-130-9. DOI: 10.1145/1389095.1389340. URL: http://doi.acm.org/10.1145/1389095.1389340.

Powell, Michael J.D. (1994). "A direct search optimization method that models the objective and constraint functions by linear interpolation". In: *Advances in Optimization and Numerical Analysis*. Ed. by Susana Gomez and Jean-Pierre Hennart. Vol. 275. Springer Netherlands, pp. 51–67. DOI: 10.1007/978-94-015-8330-5_4.

Quinlan, J. Ross (1993). *C4.5: Programs for Machine Learning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Ralston, J. C. and A. M. Zoubir (1995). "Identification of time-varying Hammerstein systems". In: *1995 International Conference on Acoustics, Speech, and Signal Processing*. Vol. 3, 1685–1688 vol.3. DOI: 10.1109/ICASSP.1995.479929.

Rasmussen, Carl Edward (1997). "Evaluation of Gaussian Processes and Other Methods for Non-linear Regression". AAINQ28300. PhD thesis. Toronto, Ont., Canada, Canada. ISBN: 0-612-28300-3.

Rinnooy Kan, A. H. G. and G. T. Timmer (1987a). "Stochastic global optimization methods part I: Clustering methods". In: *Mathematical Programming* 39.1, pp. 27–56. DOI: 10.1007/BF02592070.

— (1987b). "Stochastic global optimization methods part II: Multi level methods". In: *Mathematical Programming* 39.1, pp. 57–78. DOI: 10.1007/BF02592071.

Rissanen, Jorma (1978). "Modeling by shortest data description". In: *Automatica* 14.5, pp. 465–471.

Rockett, Peter and Elizabeth A Hathway (2017). "Model-predictive control for non-domestic buildings: Critical review and prospects". In: *Building Research & Information* 45.5, pp. 556–571. DOI: 10.1080/09613218.2016.1139885.

Ruberto, Stefano et al. (2014). "ESAGP – A Semantic GP Framework Based on Alignment in the Error Space". In: *Genetic Programming*. Berlin, Heidelberg, pp. 150–161.

Saltelli, Andrea et al. (2004). *Sensitivity Analysis in Practice: A Guide to Assessing Scientific Models*. John Wiley & Sons.

Sbeity, F. et al. (2012). "Sub and ultra harmonic extraction using several Hammerstein models". In: *2012 IEEE International Conference on Complex Systems (ICCS)*, pp. 1–5. DOI: 10.1109/ICoCS.2012.6458608.

Schaffer, J. David (1985). "Multiple Objective Optimization with Vector Evaluated Genetic Algorithms". In: *Proceedings of the 1st International Conference on Genetic Algorithms*.

Hillsdale, NJ, USA: L. Erlbaum Associates Inc., pp. 93–100. ISBN: 0-8058-0426-9. URL: http://dl.acm.org/citation.cfm?id=645511.657079.

Schmidt, Michael D. and Hod Lipson (2009). "Incorporating Expert Knowledge in Evolutionary Search: A Study of Seeding Methods". In: *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation*. GECCO '09. Montreal, Québec, Canada: ACM, pp. 1091–1098. ISBN: 978-1-60558-325-9. DOI: 10.1145/1569901.1570048. URL: http://doi.acm.org/10.1145/1569901.1570048.

Sentoni, G. et al. (1996). "Approximate models for nonlinear process control". In: *AIChE Journal* 42.8, pp. 2240–2250. DOI: 10.1002/aic.690420813. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/aic.690420813. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/aic.690420813.

Seyab, R.K. Al and Y. Cao (2008). "Nonlinear system identification for predictive control using continuous time recurrent neural networks and automatic differentiation". In: *Journal of Process Control* 18.6, pp. 568 –581. ISSN: 0959-1524. DOI: https://doi.org/10.1016/j.jprocont.2007.10.012. URL: http://www.sciencedirect.com/science/article/pii/S095915240700159X.

Sharma, Mr Sunny and Rajinder Singh Virk (2014). "A review towards evolutionary multiobjective optimization algorithms". In: *An International Journal of Engineering Sciences, Vol13/37-Vol13* 3.

Shen, Q. et al. (2014). "Novel Neural Control for a Class of Uncertain Pure-Feedback Systems". In: *IEEE Transactions on Neural Networks and Learning Systems* 25.4, pp. 718–727. ISSN: 2162-237X. DOI: 10.1109/TNNLS.2013.2280728.

Sjöberg, Jonas et al. (1995). "Nonlinear black-box modeling in system identification: A unified overview". In: *Automatica* 31.12, pp. 1691–1724. DOI: 10.1016/0005-1098(95)00120-8.

Söderström, Torsten and Petre Stoica (1989). *System Identification*. New York: Prentice Hall.

Soule, T. and J. A. Foster (1998). "Removal bias: a new cause of code growth in tree based evolutionary programming". In: *1998 IEEE International Conference on Evolutionary Computation Proceedings. IEEE World Congress on Computational Intelligence (Cat. No.98TH8360)*, pp. 781–786. DOI: 10.1109/ICEC.1998.700151.

Srinivas, N. and K. Deb (1994). "Muiltiobjective Optimization Using Nondominated Sorting in Genetic Algorithms". In: *Evolutionary Computation* 2.3, pp. 221–248. ISSN: 1063-6560. DOI: 10.1162/evco.1994.2.3.221.

Stanley, Kenneth O. and Risto Miikkulainen (2002). "Evolving Neural Networks through Augmenting Topologies". In: *Evolutionary Computation* 10.2, pp. 99–127. DOI: 10.1162/106365602320169811.

Stinstra, Erwin, Gijs Rennen, and Geert Teeuwen (2008). "Metamodeling by symbolic regression and Pareto simulated annealing". In: *Structural and Multidisciplinary Optimization* 35.4, pp. 315–326. ISSN: 1615-1488. DOI: 10.1007/s00158-007-0132-4. URL: https://doi.org/10.1007/s00158-007-0132-4.

Tai, Nguyen Trong and Kyoung Kwan Ahn (2012). "A hysteresis functional link artificial neural network for identification and model predictive control of SMA actuator". In: *Journal of Process Control* 22.4, pp. 766 –777. ISSN: 0959-1524. DOI: https://doi.org/10.1016/j.jprocont.2012.02.007. URL: http://www.sciencedirect.com/science/article/pii/S095915241200042X.

Tan, Li and J. Jiang (2001). "Adaptive Volterra filters for active control of nonlinear noise processes". In: *IEEE Transactions on Signal Processing* 49.8, pp. 1667–1676. ISSN: 1053-587X. DOI: 10.1109/78.934136.

Tanev, I. and K. Shimohara (2010). "XML-based genetic programming framework: Design philosophy, implementation, and applications". In: *Artificial Life and Robotics* 15.4, pp. 376–380.

Tingqi, Yuan, Liu Changlu, and Liu Wenjiang (2000). "Identification of Hammerstein model based on dynamical separation technology". In: *Proceedings of the 3rd World Congress on Intelligent Control and Automation (Cat. No.00EX393)*. Vol. 3, 2124–2128 vol.3. DOI: 10.1109/WCICA.2000.862976.

Topchy, Alexander and W. F. Punch (2001). "Faster genetic programming based on local gradient search of numeric leaf values". In: *3rd Annual Conference on Genetic and Evolutionary Computation (GECCO'01)*. San Francisco, California, pp. 155–162.

Trujillo, Leonardo et al. (2017). "Local search is underused in genetic programming". In: *Genetic Programming Theory and Practice XIV*. Ann Arbor, MI: Springer.

Tsoukalas, Lefteri H. and Robert E. Uhrig (1996). *Fuzzy and Neural Approaches in Engineering*. 1st. New York, NY, USA: John Wiley & Sons, Inc. ISBN: 0471160032.

Đukić, Savo D and Andrija T Sarić (2012). "Dynamic model reduction: An overview of available techniques with application to power systems". In: *Serbian journal of electrical engineering* 9.2, pp. 131–169.

Uy, Nguyen Quang et al. (2010). "The role of syntactic and semantic locality of crossover in genetic programming". In: *11th International Conference on Parallel Problem Solving from Nature: Part II (PPSN'10)*. Krakow, Poland, pp. 533–542.

Uy, Nguyen Quang et al. (2011). "Semantically-based crossover in genetic programming: Application to real-valued symbolic regression". In: *Genet. Program. Evol. Mach.* 12.2, pp. 91–119. DOI: 10.1007/s10710-010-9121-2.

Vanneschi, Leonardo, Mauro Castelli, and Sara Silva (2014). "A survey of semantic methods in genetic programming". In: *Genet. Program. Evol. M.* 15.2, pp. 195–214. DOI: 10.1007/s10710-013-9210-0.

Vanneschi, Leonardo et al. (2013). "A New Implementation of Geometric Semantic GP and Its Application to Problems in Pharmacokinetics". In: *Genetic Programming*. Berlin, Heidelberg, pp. 205–216.

Vapnik, Vladimir (1998). "Statistical Learning Theory; John Wiley and Sons". In:

Vazquez, Rafael and Miroslav Krstic (2008a). "Control of 1-D parabolic PDEs with Volterra nonlinearities, Part I: Design". In: *Automatica* 44.11, pp. 2778 –2790. ISSN: 0005-1098. DOI: https://doi.org/10.1016/j.automatica.2008.04.013. URL: http://www.sciencedirect.com/science/article/pii/S000510980800277X.

— (2008b). "Control of 1D parabolic PDEs with Volterra nonlinearities, Part II: Analysis". In: *Automatica* 44.11, pp. 2791 –2803. ISSN: 0005-1098. DOI: https://doi.org/10.1016/j.automatica.2008.04.007. URL: http://www.sciencedirect.com/science/article/pii/S0005109808002586.

Vladislavleva, E. J., G. F. Smits, and D. den Hertog (2009). "Order of nonlinearity as a complexity measure for models generated by symbolic regression via Pareto genetic programming". In: *IEEE T. Evolut. Comput.* 13.2, pp. 333–349. DOI: 10.1109/TEVC.2008.926486.

Vlist, Eric van der (2002). *XML Schema*. Sebastopol, CA: O'Reilly.

Voros, J. (1999). "Iterative algorithm for parameter identification of Hammerstein systems with two-segment nonlinearities". In: *IEEE Transactions on Automatic Control* 44.11, pp. 2145–2149. ISSN: 0018-9286. DOI: 10.1109/9.802933.

Vyas, Renu, Purva Goel, and Sanjeev S. Tambe (2015). "Genetic Programming Applications in Chemical Sciences and Engineering". In: ed. by Amir H. Gandomi, Amir H. Alavi, and Conor Ryan. Cham: Springer International Publishing, pp. 99–140. ISBN: 978-3-319-20883-1.

Wahba, G. and S. Wold (1975). "A completely automatic French curve: fitting spline functions by cross validation". In: *Comm. Stat.* 4.1, pp. 1–17. DOI: 10.1080/03610927508827223.

Wang, P. et al. (2011). "A memetic genetic programming with decision tree-based local search for classification problems". In: *IEEE Congress of Evolutionary Computation (CEC)*, pp. 917–924. DOI: 10.1109/CEC.2011.5949716.

Washington Post (2016). *An alarming number of scientific papers contain Excel errors*. [Accessed 27 March 2017]. URL: https://www.washingtonpost.com/news/wonk/wp/2016/08/26/an-alarming-number-of-scientific-papers-contain-excel-errors/.

WHATWG (2018). *Document Object Model (DOM)*. [Accessed 25 June 2018]. URL: https://dom.spec.whatwg.org/.

White, D. R. et al. (2013). "Better GP benchmarks: Community survey results and proposals". In: *Genetic Programming and Evolvable Machines* 14.1, pp. 3–29.

World Wide Web Consortium (2008). *Extensible Markup Language (XML) 1.0*. [Accessed 17 July 2018]. URL: https://www.w3.org/TR/2008/REC-xml-20081126/.

— (2012). *XML Schemas*. [Accessed 17 July 2018]. URL: https://www.w3.org/standards/xml/schema.

Xiong, Zhihua and Jie Zhang (2005). "A batch-to-batch iterative optimal control strategy based on recurrent neural network models". In: *Journal of Process Control* 15.1, pp. 11–21. ISSN: 0959-1524. DOI: https://doi.org/10.1016/j.jprocont.2004.04.005. URL: http://www.sciencedirect.com/science/article/pii/S0959152404000526.

Yu, Li et al. (2008). "Parameter estimation error bounds for Hammerstein nonlinear finite impulsive response models". In: *Applied Mathematics and Computation* 202.2, pp. 472–480. ISSN: 0096-3003. DOI: https://doi.org/10.1016/j.amc.2008.01.002. URL: http://www.sciencedirect.com/science/article/pii/S0096300308000143.

Yuzgec, U., Y. Becerikli, and M. Turker (2008). "Dynamic Neural-Network-Based Model-Predictive Control of an Industrial Baker's Yeast Drying Process". In: *IEEE Transactions*

on Neural Networks 19.7, pp. 1231–1242. ISSN: 1045-9227. DOI: 10.1109/TNN.2008.2000205.

Z-Flores, Emigdio et al. (2014). "Evaluating the effects of local search in genetic programming". In: *EVOLVE - A Bridge between Probability, Set Oriented Numerics, and Evolutionary Computation V*. Cham, pp. 213–228. DOI: 10.1007/978-3-319-07494-8_15.

Zavoianu, Alexandru-Ciprian (2010). "Towards solution parsimony in an enhanced genetic programming process". In: *Master's thesis, International School Informatics: Engineering & Management, ISI-Hagenberg, Johannes Kepler University, Linz*.

Zhang, Jian, Kwai-Sang Chin, and Maciej Ławryńczuk (2018). "Nonlinear model predictive control based on piecewise linear Hammerstein models". In: *Nonlinear Dynamics* 92.3, pp. 1001–1021. ISSN: 1573-269X. DOI: 10.1007/s11071-018-4105-5. URL: https://doi.org/10.1007/s11071-018-4105-5.

Zhang, M., X. Gao, and W. Lou (2007). "A new crossover operator in genetic programming for object classification". In: *IEEE T. Syst. Man. Cy. B* 37.5, pp. 1332–1343.

Zhang, Mengjie and Will Smart (2004). "Genetic programming with gradient descent search for multiclass object classification". In: *7th European Conference on Genetic Programming (EuroGP 2004)*. Coimbra, Portugal, pp. 399–408. DOI: 10.1007/978-3-540-24650-3_38.

Zhang, Qian, Qunjing Wang, and Guoli Li (2016). "Nonlinear modeling and predictive functional control of Hammerstein system with application to the turntable servo system". In: *Mechanical Systems and Signal Processing* 72-73, pp. 383 –394. ISSN: 0888-3270. DOI: https://doi.org/10.1016/j.ymssp.2015.09.011. URL: http://www.sciencedirect.com/science/article/pii/S0888327015004069.

Zitzler, E. and L. Thiele (1999). "Multiobjective evolutionary algorithms: a comparative case study and the strength Pareto approach". In: *IEEE Transactions on Evolutionary Computation* 3.4, pp. 257–271. ISSN: 1089-778X. DOI: 10.1109/4235.797969.

Ławryńczuk, Maciej (2013). "Practical nonlinear predictive control algorithms for neural Wiener models". In: *Journal of Process Control* 23.5, pp. 696 –714. ISSN: 0959-1524. DOI: https://doi.org/10.1016/j.jprocont.2013.02.004. URL: http://www.sciencedirect.com/science/article/pii/S0959152413000322.

— (2014). *Computationally Efficient Model Predictive Control Algorithms*. Springer, Cham.