

**Bias and Variance Reduction Strategies for
Improving Generalisation Performance of Genetic
Programming on Binary Classification Tasks**

by

Jeannie Fitzgerald B.Sc.

Supervisor: Prof. Conor Ryan

External Examiner: Dr. Anna I. Esparcia-Alcázar

Internal Examiner: Dr. Michael English

A thesis for the PhD Degree

Submitted to the University of Limerick

May 2014



UNIVERSITY of LIMERICK

OLLSCOIL LUIMNIGH

Declaration

I hereby declare that the work presented in this thesis is original except where an acknowledgement is made or a reference is given to other work. I confirm that I have read and accept the Handbook of Academic Regulations and Procedures.

Student: Jeannie Fitzgerald

Signed:

Date: May 2014

Supervisor: Professor Conor Ryan

Signed:

Date: May 2014

Abstract

The central hypothesis of this thesis is that the reduction of *variance* and *inappropriate bias* in Genetic Programming (GP) will lead to the evolution of more generalisable and robust numerical binary classifiers. A secondary, supporting, hypothesis is that dynamic, individualised approaches may have a role to play in reducing the magnitude of error due to bias and variance, as such approaches can introduce diversity and change into the learning system. We expect that, where an influencing parameter is applied identically to each member of the population, and remains unchanged throughout evolution, that any (undesirable) effects on bias and variance error are likely to be stronger than if individuals in the population apply the same parameter differently, and where the application of any such parameter can change in response to system behaviour. In other words, a monolithic system may suffer from *monolithic bias*, and we believe that the introduction of individualised, dynamic approaches may have a beneficial effect in diluting this, leading to improved generalisation in the GP learner.

We explore the concepts of bias and variance as components of generalisation error for binary classification tasks, and investigate aspects of the GP paradigm which may influence these error components. Specifically, we identify sources of *variance*, *language bias*, *search bias* and *selection bias* inherent in standard GP for binary classification and pose several core questions relating to these sources. If the research can be shown to affirmatively answer these core questions, then our hypotheses will have been proven.

In responding to the core questions we carry out several empirical studies with the objective of gaining a deeper understanding of the impacts of these sources of bias and variance on generalisation and we propose several novel approaches which may be used to reduce variance, or to replace inappropriate inductive biases with more appropriate ones, with a view to improving generalisation performance.

Ultimately we combine several techniques, developed to address our fundamental questions, into a single, optimised GP (OGP) configuration. This is evaluated on nine different binary classification tasks and compared with the performance of several well known and respected machine learning algorithms on the same datasets. Results of these experiments demonstrate that a GP learner which has been optimised to reduce variance and bias error through individualised, dynamic and population based adaptations can deliver classification performance which is competitive with other machine learning algorithms.

The empirical studies and proposed techniques described in this theses provide answers to the core questions which we believe validate our central and supporting hypotheses.

I dedicate this thesis to the three most important men in my life: Eoin, whose philosophy during his short life inspired me to begin this journey - because sometimes life really is too short not to spend it doing things we feel passionate about; Luke, for encouraging me to reach for the finish-line by (frequently) pointing out that the reason I had not already done so was that the work was so enjoyable that I did not want it to end at all; and Richard, for supporting me wholeheartedly from beginning to end, in every way that mattered.

Acknowledgements

The writing of this thesis has been one of the most difficult yet rewarding experiences of my life. Without the support, patience and encouragement of the following people, this work could not have been completed. I owe them all my sincere thanks.

Richard, my husband, who has cheerfully put up with all of the sacrifices, late nights and lost weekends without complaint. He was always there to motivate and re-invigorate when I was running out of steam. This thesis would not have been possible without his emotional and practical support.

Professor Conor Ryan who provided me with the opportunity to begin with, and who gave so generously of his time throughout, despite his many other academic and professional commitments. His generosity of spirit, expert knowledge, and commitment to the highest standards both inspired and motivated me. Conor encouraged me to follow my own path, but was always standing by, ready to advise and encourage if I faltered on the way.

My second mentor, friend and colleague, Dr. R.M. Atif Azad for the many stimulating conversations we shared about life, family, politics, and of course - work. His positive attitude, wonderful smile and tremendous work ethic were a constant inspiration. I am deeply grateful for his help, encouragement and advice since I joined the BDS research group.

Members of the Bio-computing and Developmental Systems group with whom I have been lucky to work side-by-side with: thanks to Miriam, Fiacc, Darwin, John, Tom, Gopinath,

Conor, Fergal and David for their good company and helpful feedback over the years. I am grateful also to colleague Dr. Damien T. Hogan for generously sharing his thesis writing and LaTeX tips. Special thanks to my cubicle neighbour James Patten for his friendship and sense of humour, and for the lively arguments which were a welcome distraction and source of amusement.

Dr. Norah Power who really helped me to maximise my research time by offering help and practical support with some of my teaching responsibilities. I am very grateful for her guidance and practical assistance.

BDS and Department of Computer Science and Information Systems administrative staff, for their practical help and consummate professionalism. Thanks to Nuala, Anne and Gemma for all the support.

My family and friends, whom I hope believe in the adage that “absence makes the heart grow fonder” rather than “seldom seen soon forgotten”.

Science Foundation Ireland, for financial support which I gratefully acknowledge: Grant No. 10/IN.1/I3031.

Many thanks also, to external and internal examiners Dr. Anna I. Esparcia-Alcázar and Dr. Michael English for taking the time to read the work.

Last, but definitely not least, I would like to express my deep gratitude and give sincere thanks to Annette McElligott: without her kindness and generosity I might have missed out on one of the most rewarding experiences of my life.

Publications

- J. Fitzgerald and C. Ryan. Drawing boundaries: using individual evolved class boundaries for binary classification problems. In N. Krasnogor and P. L. Lanzi, editors, *GECCO*, pages 1347–1354. ACM, 2011
- J. Fitzgerald and C. Ryan. Validation sets, Genetic Programming and Generalisation. In *Research and Development in Intelligent Systems XXVIII. Proceedings of AI-2011 the 31st SGAI international conference on Innovative Techniques and Applications of Artificial Intelligence*, SGAI'11, pages 79–93. Springer, 2011b.
- J. Fitzgerald and C. Ryan. Validation sets for Evolutionary Curtailment with Improved Generalisation. In *ICHIT (1)*, pages 282–289, 2011a.
- J. Fitzgerald and C. Ryan. Exploring Boundaries: Optimising Individual Class Boundaries for Binary Classification Problems. In *Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference, GECCO '12*, pages 743–750, New York, NY, USA, 2012. ACM.
- J. Fitzgerald and C. Ryan. A Hybrid Approach to the Problem of Class Imbalance. In *International Conference on Soft Computing*, Brno, Czech Republic, June 2013.
- J. Fitzgerald, R. M. A. Azad, and C. Ryan. Bootstrapping to Reduce Bloat and Improve Generalisation in Genetic Programming. In *GECCO Companion*, Amsterdam, The Netherlands, July 2013. ACM.
- J. Fitzgerald, R. M. A. Azad, and C. Ryan. A Bootstrapping Approach to Reduce Over-fitting in Genetic Programming. In *GECCO Companion*, Amsterdam, The Netherlands, July 2013. ACM.
- J. Fitzgerald and C. Ryan. Individualized Self-Adaptive Genetic Operators with Adaptive Selection in Genetic Programming. In 5th World Congress on Nature and Biologically Inspired Computing, NaBIC 2013. August 2013.

Conor Ryan, Krzysztof Krawiec, Una-May O'Reilly, Jeannie Fitzgerald and David Medernach. Building a Stage 1 Computer Aided Detector for Breast Cancer using Genetic Programming. Proceedings of the 17th European Conference on Genetic Programming, EuroGP 2014, EuroGP, Granada, Spain. To appear.

J. Fitzgerald and C. Ryan. On Size, Complexity and Generalisation Error in GP. In *Proceedings of the sixteenth international conference on Genetic and evolutionary computation conference*, GECCO '14, Vancouver, BC, Canada, 2014. ACM. To appear.

Contents

List of Figures	xix
List of Tables	xxiii
1 Introduction	1
1.1 Motivation	2
1.1.1 Disadvantages of GP	3
1.1.2 Bias and Variance	5
1.2 Central Hypothesis	10
1.3 Core Research Questions	13
1.4 Contribution of the Thesis	14
1.4.1 Core Question 1: Variance	14
1.4.2 Core Question 2: Search Bias	15
1.4.3 Core Question 3: Language Bias	16
1.4.4 Core Question 4: Selection Bias	16
1.4.5 Evaluation	17
1.5 Scope and Limitations of the Thesis	18
1.5.1 Scope	18
1.5.2 Limitations	18
1.6 Outline of the Thesis	19
1.7 Tools and Datasets	22
2 Background	23
2.1 The Evolutionary Paradigm	23
2.1.1 Genetic Programming	25

2.1.1.1	Exploration and Exploitation	26
2.1.1.2	Fitness Function	26
2.1.1.3	Genetic Operators	27
2.1.1.4	Replacement Strategy	32
2.1.1.5	Functions and Terminals	34
2.1.1.6	Initialisation	36
2.1.1.7	Parameters	37
2.1.1.8	Advantages of GP	37
2.1.1.9	Alternative Implementations	38
2.1.2	Related Approaches	39
2.2	Classification	41
2.2.1	Classification Terminology	42
2.2.1.1	Precision	43
2.2.1.2	Recall	43
2.2.1.3	Specificity	43
2.2.1.4	Accuracy	43
2.2.1.5	Area Under the ROC Curve (AUC)	43
2.2.2	Numerical Binary Classifiers in GP	45
2.2.3	Selected ML Classifiers	48
2.2.3.1	K Nearest Neighbours	48
2.2.3.2	Naive Bayes	50
2.2.3.3	Decision Trees	52
2.2.3.4	Logistic Regression	54
2.2.3.5	Multi-Layer Perceptron	56
2.2.3.6	Support Vector Machines	59
3	A Review of Generalisation in Genetic Programming	63
3.1	Generalisation in Machine Learning	63
3.2	Generalisation in Genetic Programming	67
3.2.1	Regularisation and MDL Strategies - Size and Complexity	69
3.2.2	Validation Sets and Early Stopping	72
3.2.3	Sampling Approaches	74
3.2.4	Approaches specific to an EC methodology	75

3.3	Computational Learning Theory	78
4	Variance	81
4.1	Bootstrapping	82
4.1.1	Research Objective	82
4.1.2	Previous work on Bootstrapping	83
4.1.3	Details of Proposed Technique	85
4.1.4	Experiments	87
4.1.4.1	Bootstrap Configurations	87
4.1.5	Results and Discussion	88
4.1.5.1	Program Growth	91
4.1.5.2	Size Fitness Correlation	98
4.1.5.3	Comparison with other Methods	99
4.1.6	Summary	100
4.2	Validation Sets with Static Evolutionary Curtailment	101
4.2.1	Research Objective	102
4.2.2	Related Work	103
4.2.3	Details of Proposed Technique	104
4.2.4	Experiments	105
4.2.4.1	Configurations	105
4.2.5	Results and Discussion	106
4.2.5.1	Training and Test Fitness Scores	106
4.2.5.2	Validation Elitism	108
4.2.5.3	Comparison with other work	109
4.2.6	Summary	112
4.3	Validation Sets with Dynamic Evolutionary Curtailment	113
4.3.1	Details of Proposed Techniques	114
4.3.2	Experiments	116
4.3.2.1	Experimental Configurations	116
4.3.3	Results and Discussion	117
4.3.3.1	When to Stop?	118
4.3.3.2	Main Experiments	118
4.3.4	Summary	120
4.3.5	Research Outcomes	123

5	Search Bias	125
5.1	Introduction	125
5.2	Research Objectives	128
5.3	Previous Work on Boundary Determination	128
5.4	Exploring Boundaries	130
5.4.1	Details of Proposed Technique	130
5.4.2	Experiments	131
5.4.2.1	Function Sets	131
5.4.2.2	Experimental Set-up and GP Parameters	132
5.4.2.3	Results and Discussion	134
5.4.2.4	Comparison with other work	142
5.4.2.5	Summary	145
5.5	Optimising Boundaries	147
5.5.1	Boundary determination techniques	147
5.5.1.1	ECB	147
5.5.1.2	AUCN	147
5.5.1.3	AUCP	148
5.5.1.4	ECBS	148
5.5.1.5	OICB	149
5.5.1.6	OICBP	149
5.5.1.7	OICBU	149
5.5.1.8	OICB+	150
5.5.2	Experiments	150
5.5.2.1	Data Sets	150
5.5.2.2	Experimental Set-up and GP Parameters	150
5.5.2.3	Results and Discussion	151
5.5.2.4	Comparison with Other Methods	158
5.5.2.5	Program Analysis	159
5.5.2.6	Generation of Best Individual	163
5.5.3	Summary	163
5.6	The Role of Self-adaptation	165
5.6.1	Previous Work on Self-Adaptive Operators	165
5.6.2	Proposed Method	168
5.6.2.1	Individualised Adaptive Genetic Operators	168

5.6.2.2	Adaptive Tournament Selection	169
5.6.3	Experiments	170
5.6.3.1	Experimental Set-up and GP Parameters	170
5.6.3.2	Results and Discussion	172
5.6.4	Summary	175
5.7	Research Outcomes	176
6	Language Bias	179
6.1	Size Versus Complexity	179
6.1.1	Background	179
6.1.2	Research Objectives	180
6.1.3	Experiments	181
6.1.4	Experimental Configurations	181
6.1.5	Results	183
6.1.5.1	Training and Test Accuracy, AUC and Over-fitting	183
6.1.5.2	Size	190
6.2	Research Outcomes	196
6.3	Conclusions	200
7	Selection Bias	203
7.1	The Class Imbalance Problem	203
7.1.1	Research Objective	204
7.1.2	Previous Work on Class Imbalance	205
7.1.2.1	GP	206
7.1.3	A Hybrid Approach	207
7.1.4	Experimental Configurations	211
7.1.5	GP Parameters	213
7.1.6	Results and Discussion	214
7.2	Replacement Strategy, Tournament Size and Elitism . . .	219
7.2.1	Research Objective	219
7.2.2	Previous work	219
7.2.3	Experimental Configurations	221
7.2.4	Results and Discussion	222

7.2.4.1	Tournament Size	222
7.2.4.2	Replacement Strategy	222
7.2.4.3	Elitism	223
7.2.4.4	Further Remarks	223
7.2.5	Summary	225
7.3	Research Outcomes	226
8	Optimised GP	229
8.1	Optimising GP	229
8.2	ML Classification Algorithms	232
8.3	Experiments	232
8.3.1	Data Sets	232
8.3.2	Parameter Settings	232
8.4	Results and Discussion	234
8.5	Summary	242
8.6	Research Outcomes	242
9	Conclusions and Future Directions	245
9.1	Conclusions	245
9.1.1	Research Objectives	245
9.1.1.1	CQ1: Variance	246
9.1.1.2	CQ2: Search Bias	247
9.1.1.3	CQ4: Language Bias	249
9.1.1.4	CQ3: Selection Bias	250
9.1.2	Competitiveness	250
9.1.3	Conclusion	251
9.2	Future Directions	251
9.2.1	ECB/OICB+	252
9.2.2	The Class Imbalance Problem	252
9.2.3	Bootstrapping	252
9.2.4	Self Adaptive Operators	254
9.2.5	Size and Functional Complexity	254
9.3	Further Remarks	255
	Appendices	257

A Datasets	259
A.1 Data Sets	260
A.1.0.1 Biodegradation (BIO)	260
A.1.0.2 Blood Transfusion (BT)	261
A.1.0.3 Bupa Liver Disorders (BUPA)	261
A.1.0.4 Caravan (CAR)	261
A.1.0.5 Ecoli (EC)	262
A.1.0.6 German Credit (GC)	262
A.1.0.7 Habermans' Survival (HS)	262
A.1.0.8 Ionosphere (ION)	262
A.1.0.9 Pima Indians Diabetes data set	262
A.1.0.10 Wisconsin Breast Cancer (WBC)	263
A.1.0.11 Yeast	263
A.1.0.12 Brodtz Texture (BR)	263
B OICB Algorithm	265
C ECB2 Additional Experiments	267
D Texture Experiments: ECB and Reduced Function Set	271
E Selection Bias Results	275

List of Figures

1.1	The bias variance trade-off [Fortmann-Roe, 2012].	7
1.2	Bias and variance in dart throwing.[Domingos, 2012]	8
2.1	Example Numerical Expression GP Tree	26
2.2	Flowchart shows a typical GP run for a population of size N, using a generational replacement strategy and selection based on fitness.	28
2.3	Example of Two Parent Crossover	30
2.4	Example of Point Mutation	31
2.5	Example of Sub-tree Mutation	32
2.6	Comparison of ROC Curves	44
2.7	Example GP Numerical Classifier	46
2.8	Classification of first instance	47
2.9	Classification of second instance	47
2.10	Naive Bayes for classification	51
2.11	Entropy for a two class problem	53
2.12	Schematic of Single Perceptron. [Sayed, 2012]	57
2.13	Perceptron Activation. [Sayed, 2012]	57
2.14	Multi-Layer Perceptron	58
2.15	Illustration of support vectors for maximal margin.	60
3.1	Example Validation Error on the BUPA dataset from a Chap- ter 4 experimental run. An ideal stopping point would be generation 60, however there are several points before that where a naive early stopping heuristic may have terminated evolution.	73

4.1	Bootstrap Illustration	86
4.2	Bootstrap Experiments: BT Average Training and Test Error Rates	89
4.3	Bootstrap Experiments: Bupa Average Training and Test Error Rates	89
4.4	Bootstrap Experiments: HS Average Training and Test Error Rates	90
4.5	Bootstrap Experiments: WBC Average Training and Test Error Rates	90
4.6	Average Tree Size in Nodes: BT and BUPA Data	92
4.7	Average Tree Size in Nodes: HS and WBC Data	92
4.8	Size Frequency Distribution of Best of Run Individuals GP Configuration: BT and BUPA Data.	94
4.9	Size Frequency Distribution of Best of Run Individuals GP Configuration: HS and WBC Data.	95
4.10	Size Frequency Distribution of Best of Run Individuals Bootstrap Configuration: BT and BUPA Data.	95
4.11	Size Frequency Distribution of Best of Run Individuals Bootstrap Configuration: HS and WBC Data.	96
4.12	Phenotypic Diversity: BT and BUPA Data.	97
4.13	Phenotypic Diversity: HS and WBC Data.	97
4.14	Overall ranking of algorithms across problems with Friedman test, where 1 ranks highest.	122
5.1	Illustration of undesirable/inappropriate bias introduced when using a standard approach: zero boundary	131
5.2	Evolution of ECB Median and Median Absolute Deviation values for typical BUPA runs, Median and MAD on left axis.	140
5.3	Evolution of ECB Median and Median Absolute Deviation values for typical PID runs, Median and MAD on left axis	140
5.4	Evolution of ECB Median and Median Absolute Deviation values for typical WBC runs, Median and MAD on left axis	141
5.5	Relative ranking of Boundary Determination Algorithms, where 1 represents the best performance and 9 the worst.	156

5.6	Comparison of function usage for OICB+ and Static configurations.	160
5.7	Comparison of attribute usage for OICB+ and Static configurations.	162
5.8	Methods ranked from 1 to 5 based on average Test Accuracy, where 1 is best and 5 is worst.	176
7.1	Proportional Individualised Random Sampling example for a source distribution with 20% minority instances. P1-Pn are programs in the population trained on potentially different input samples where the percentage of majority instances for that individual is in the range 20-80%	209
7.2	Class Imbalance Experiments: Methods ranked from 1 to 7 based on average AUC, where 1 is best and 7 is worst.	215
8.1	Ranked comparison of all methods across the 9 classification datasets.	241
9.1	Example Bootstrap Tree Lattices.	253
9.2	Corresponding Standard GP Tree Lattices where identical seeds to the Bootstrap runs were used, i.e. run 0 of Standard GP used the same random seed as run 0 of the Bootstrap configuration.	254
D.1	Texture Images	272

List of Tables

2.1	Confusion Matrix	42
2.2	Named attributes and their values for two example instances.	45
4.1	Bootstrap Experiments: GP Parameters	85
4.2	Fitness Configurations for Bootstrap Experiments	88
4.3	Bootstrap Best of Run Individuals: All values are averaged over 200 Best-of-run trained Individuals for each task, for each configuration.	93
4.4	Average training, test error correlated with average tree size, GP and BootStrap configurations. Values closer to -1 indicate a stronger relationship between increasing program size and decreasing error.	98
4.5	Bootstrap Experiments: Comparative ML Algorithms	100
4.6	Performance comparison of BS and GP with ML algorithms listed in Table 4.5	101
4.7	Static Early Stopping Experiments: GP Parameters	106
4.8	Static Early Stopping Training & Test Accuracy. For Training, Average is the average of the best-of-run trained individuals, in all cases Best is the best individual overall. Test results in bold font are significant at the 95% confidence level with both Student's t-test and Mann Whitney U Test	107
4.9	Test Fitness VP with Elitism: average fitness on test data over 100 runs at each setting	108
4.10	Static Early Stopping Experiments: BUPA Comparative Data	110
4.11	Static Early Stopping Experiments: HS Comparative Data [Cetisli, 2010]	111

4.12	Static Early Stopping: WBC Comparative Data	112
4.13	Chronos configurations, where C# is the Chronos version, VE is Validation Elitism, VP is Validation Pressure, VS is primary selection on Validation and trigger gen is the generation at which the early stopping mechanism is applied.	116
4.14	Chronos Specific Parameters	117
4.15	PID Training and Test Results for a range of “Trigger” Generations. Best Gen is the average generation at which the best so far individual was found. Stop Gen is the average generation at which evolution was terminated.	119
4.16	Dynamic Early Stopping Experiments: Training & Test Results. Size is average tree size in nodes. Gen is the average generation at which the best so far individual was found. Best test results are in bold font. Best average test result for PIMA was achieved with Chronos1 as shown in table 4.15	121
5.1	GP Parameters for initial Class Boundary experiments.	132
5.2	ECB Experimental Configurations.	133
5.3	Training Validation & Test Fitness: Static, CDCBD and ECB Configurations. Best results are in bold font.	135
5.4	Average Tree Size: Static, CDCBD and ECB Configurations.	137
5.5	Average Nodes Processed: Static, ECB and CDCBD Configurations.	138
5.6	WBC Comparative Data from [Polat and Güneş, 2008]	143
5.7	Statlog[Michie et al., 1994] Training and Test error rates on PID Domain.	144
5.8	BUPA Comparative Data from Polat and Güneş [2008]	145
5.9	GP Parameters for Boundary Determination Experiments.	148
5.10	Boundary Determination Experimental Configurations	150
5.11	Training & Test Results for Selected Boundary Determination Approaches. Best results in each category are shown in bold font.	154
5.12	Boundary Determination Experiments Average Tree Size	156
5.13	Avg. Nodes Processed (1000s)	157

5.14	Comparison of OICB and other Machine Learning Algorithms % Accuracy, ranked from top to bottom according to average accuracy. AdaBoostMI is a Weka implementation of Adaboost [Freund and Schapire, 1996] and the implementations of SVM and RandForest are those of Chang and Lin [2011] and Breiman [2001] respectively.	158
5.15	Generation of best-of-run Individual	163
5.16	GP Parameters for Self-adaptive Individualised Genetic Operator Experiments.	171
5.17	Method Configurations for Individualised Self-adaptive Genetic Operator Experiments.	172
5.18	Best Training & Test Results for Self-adaptive Experiments.	174
6.1	GP Parameters for Size, Complexity Experiments	181
6.2	Size Complexity Experiments: Size Constraints	182
6.3	Size Complexity Experiments: Operator Complexity Configurations	182
6.4	Size Complexity Experiments: BUPA Training & Test Accuracy	185
6.5	Size Complexity Experiments: HS Training & Test Accuracy. Best results are in bold font.	186
6.6	Size Complexity Experiments: ECO Training & Test Accuracy. Best results are in bold font.	188
6.7	Size Complexity Experiments: ION Training & Test Accuracy. Best results are in bold font.	189
6.8	Size Complexity Experiments: CAR Training & Test Accuracy. Best results are in bold font.	191
6.9	Size Complexity Experiments: BUPA Size Data	193
6.10	Size Complexity Experiments: ECO Size Data	194
6.11	Size Complexity Experiments: HS Size Data	195
6.12	Size Complexity Experiments: ION Size Data	197
6.13	Size Complexity Experiments: CAR Size Data	198
7.1	GP Parameters for Class Imbalance Experiments	206
7.2	Class Imbalance Data Sets	214

7.3	Class Imbalance Experiments: Performance of best-of-run Trained Individuals. Majority and minority class results are expressed as %Error Rate, Gen is the generation at which the best-of-run individual was discovered. Best result in each category is in bold font.	218
7.4	Selection Bias Experiments: Results by Tournament Size . . .	222
7.5	Selection Bias Experiments: Results by Replacement Strategy	223
7.6	Selection Bias Experiments: Elitism	224
7.7	Optimised GP: Replacement Strategy Size Comparison . . .	225
8.1	ML Algorithms	232
8.2	Optimised GP Experiments: Comparison of Optimised GP with ML Algorithms, BIO to BUPA	235
8.3	Optimised GP Experiments: Comparison of Optimised GP with ML Algorithms, CAR to HS	237
8.4	Optimised GP Experiments: Comparison of Optimised GP with ML Algorithms, ION to WBC	238
A.1	Data Sets [Frank and Asuncion, 2010]	260
C.1	Comparison of Replacement Strategies	268
C.2	Comparison of tournament sizes 4 and 5.	269
C.3	Comparison of run durations 50 and 60 generations.	269
C.4	Comparison with and without ERCs.	270
D.1	Brodatz Texture Training and Test Fitness	274
E.1	Replacement Strategy Experiments: Performance of best-of-run Trained Individuals, BIO	276
E.2	Replacement Strategy Experiments: Performance of best-of-run Trained Individuals, BT	277
E.3	Replacement Strategy Experiments: Performance of best-of-run Trained Individuals, BUPA	278
E.4	Replacement Strategy Experiments: Performance of best-of-run Trained Individuals, CAR	279

E.5	Replacement Strategy Experiments: Performance of best-of-run Trained Individuals, GC	280
E.6	Replacement Strategy Experiments: Performance of best-of-run Trained Individuals, HS	281
E.7	Replacement Strategy Experiments: Performance of best-of-run Trained Individuals, ION	282
E.8	Replacement Strategy Experiments: Performance of best-of-run Trained Individuals, PIMA	283
E.9	Replacement Strategy Experiments: Performance of best-of-run Trained Individuals, WBC	284

Chapter 1

Introduction

The central hypothesis of this thesis is that the reduction of variance and inappropriate bias in GP will lead to the evolution of more generalisable and robust numerical binary classifiers. We test this hypothesis and the effectiveness of our approach by comparing a modified GP system with state of the art machine learning algorithms.

Generalisation is probably the most important criterion for measuring the performance of any Machine Learning (ML) algorithm [Mitchell, 1997]. However, many researchers, including Costelloe and Ryan [2009]; Kushchu [2002b] and O’Neill et al. [2010] amongst others, have expressed the view that the GP field has in the past, perhaps not paid sufficient attention to the evolution of generalisable solutions.

In this thesis, we demonstrate through several empirical studies, that non-optimised, “out of the box” GP does not generalise well when applied to a range of binary classification tasks, whereas GP that has been optimised through various novel adaptations can deliver test results on the same datasets, which are competitive with many state of the art ML algorithms.

In this first chapter we set out in detail what this thesis attempts to achieve, and why. We explain our motivation for the research and refine the central hypothesis into several core questions which will be addressed in the remainder of the thesis. Also in this chapter contributions that the work offers to the field of GP research are presented, and the scope

and limitations of the research are defined.

1.1 Motivation

Although GP has been around for some time, it is still a relative newcomer to the wider field of Machine Learning. It is probably fair to say that when GP, as we know it today, was initially mooted in the early nineties, the method was not taken as seriously by the scientific community as its proponents might have hoped. Indeed, its Champion, John Koza [1990], had occasion to issue a robust response [Koza, 1995] to a perceived anti-GP bias within a segment of the ML community at the time.

Today, the GP paradigm has matured, and its many advocates and practitioners might argue that it has established a substantial niche in areas of classification, regression and control, achieving numerous human competitive results on a variety of problems [Koza, 2010]. In addition, significant progress has been made in both investigating and formulating theoretical aspects of the approach [Poli et al., 2010]. However, in spite of this steady progress, and although the combative exchanges of the early days are very much in the past, GP has still not achieved widespread acceptance as a “trusted mainstream member of the computational problem solving toolkit” [ONeill et al., 2010].

A significant proportion of papers presented at GP conferences, and at GP tracks in EC conferences, are concerned with classification problems, and in a recent survey of work in which GP was applied to classification problems, Espejo et al. [2010] reviewed 66 papers where GP was compared with other methods with regard to classification accuracy. In 54.72% of cases GP was the best performing method.

However, if we look at a range of well cited research outside of the GP community; [Lim et al., 2000] [Lotte et al., 2007] [Harper, 2005] [Williams et al., 2006] [Kotsiantis et al., 2006b] [Kotsiantis et al., 2007], from 1999 to 2007, each of which aims to review the performance of various classification algorithms, not even one of them makes any reference to Genetic Programming!

Also, the well known and popular data mining software packages Weka [Hall et al., 2009] and Rapidminer [Mierswa et al., 2006] which provide experimental frameworks containing a large number of classification algorithms of various types. At the time of writing neither offer a GP implementation, although Rapidminer provides GA functionality.

Part of the problem may be to do with how the research is positioned: although several GP practitioners including Alfaro-Cid et al. [2007]; Archetti et al. [2007]; Bot and Langdon [2000]; Eggermont et al. [2004a]; Francone et al. [1996] and Hunt et al. [2012], to name a few, do compare their results with other Machine Learning classification algorithms, it is certainly not standard practice to do so. This is understandable, as several of the popular algorithms have a large number of parameter settings which the GP practitioner would seek to optimise in order to make a fair comparison, and the learning curve needed to acquire the required expertise may be quite steep.

In summary, although the GP community has expended considerable effort over many years tackling classification problems, and in so doing, has provided exciting and novel ideas while delivering many competitive results, it is probably true to say that the approach would not be mentioned or even considered, by practitioners outside of the GP research community, in the same way that methods such as Support Vector Machines, Decision Trees, Logistic Regression or Bayesian approaches would be.

1.1.1 Disadvantages of GP

On the face of it GP should be ideally positioned to solve classification tasks: it is very flexible and expressive, domain independent and facilitates automated feature selection. All of which means that it can be used to develop fully automated classifiers. Most importantly, GP has the ability to *explain* any classification decisions: the evolved solution is available for inspection, which can be used to provide confidence in the decision. However, GP has some perceived disadvantages which may be limiting its potential for wider acceptance as a tool for classification tasks:

1. **Poor Generalisation** [Keijzer and Babovic, 2000b]
Training results may not be a good predictor of test results;
2. **Complex Bias** [Kushchu, 2002a]
As with other classification algorithms, GP has various biases both representational and implementational. There are various parameter settings and algorithmic choices available, each of which may introduce some bias, and the combinations and permutations of these may result in complex biases which are difficult to predict and understand in advance;
3. **Lack of Comprehensibility** [Espejo et al., 2010]
Although the solution provides an explanation of the classification decision, this solution may be very large and difficult to interpret;
4. **Non-Determinism**
GP is non-deterministic which means that different runs with the same parameter settings (excluding random seed) may generate solutions of varying accuracy. Thus many runs may be necessary to deliver results which are statistically significant;
5. **Long Training** [Jabeen and Baig, 2010b; Schmutter, 2002]
Long training times result from the fact that new, potentially large, individuals need to be evaluated at each iteration, combined with the necessity to conduct many runs due to the non-deterministic nature of the paradigm;
6. **Parameterisation** [Hunt et al., 2012]
GP has a number of parameters which may affect algorithm behaviour. Determining the most appropriate settings, and combinations thereof, may require experience.

Of these disadvantages, the first is perhaps the most crucial. After all, if the objective is to develop a classifier that is to be used in the real world over a long period of time, the most important feature of such a classifier is its ability to provide provably *accurate* and *reliable* results. This is analogous to a piece of software: if that software offers the superior performance that a user requires, he will likely choose it over an inferior product even if it takes longer to develop or has a complex configuration protocol.

1.1.2 Bias and Variance

Shavlik and Dietterich [1990] described the field of supervised learning as the study of biases. The term *bias* is often used in ML, with a variety of meanings. Mitchell first defined the term in an ML context as meaning “any basis for choosing one generalisation over another other than strict consistency with the observed training instances”.

The decomposition of generalisation error into *bias* and *variance* components is well known in ML literature [Bishop, 2006; Friedman, 1997; Geman et al., 1992], as shown in Equation 1.1:

$$\textit{generalisation error} = \textit{bias} + \textit{variance} + \textit{noise} \quad (1.1)$$

Where bias error represents the extent to which the *average* prediction over all data sets differs from the desired target, and variance error measures the extent to which the solutions for individual data sets *vary* around their average, and hence variance measures the extent to which the model is sensitive to the particular choice of data set [Bishop, 2006]. Noise is the unavoidable component of the error, incurred independently of the learning algorithm [Domingos, 2000]. Friedman [1997] refers to bias and variance respectively as “the systematic and random components of classification error”.

Arguably the most widely accepted theoretical view of the bias and variance of a learner in ML is the unified approach put forward by Domingos [2000]. In that work, Domingos outlined a decomposition of bias and variance that could be applied to both regression and classification problems, which catered for both variable cost and zero-one loss functions for classification. The underpinnings of the theoretical conclusions arrived at in this and other related works, depend on having a large (possibly infinite) number of datasets of the same size, where a potentially different classifier is trained on each dataset. For zero-one loss functions, the researchers approach was to define the bias of a learner (for an example x) as the loss incurred by the *main prediction* relative to the *optimal prediction*, where the main prediction is the class most frequently predicted by the various classifiers for that example x . In this approach variance is

the *average* loss incurred by predictions relative to *the main prediction*. Bias and variance could be averaged over all examples to produce average bias and average variance.

In practice we do not have an infinite number of datasets available for experimentation. Nor do we have an infinite number of models from which we can extract prediction statistics for comparison purposes. In GP, we generally work with a training set and a test set and want to compare performance of our best models or the average performance across a population of models, over many runs.

We propose to apply the decomposition outlined by Domingos [2000] to our practical scenario by defining the average bias of a single model to be the zero-one loss (classification accuracy) on *the training set* divided by the number of training instances, and the average bias of a set of models to be the average of these average biases.

Generally, in GP literature, variance is *estimated* by evaluating error on a test data set not used during model training [Azad and Ryan, 2011b]. Thus, we define the average variance of a single model to be the difference between the average bias of the model, and the zero-one loss on the test set divided by the number of test instances. Where a positive value for average variance indicates that the loss on the test set is greater than the loss on the training set. Similarly, the average variance of a set of models is the average of the average variances.

In taking this vastly simplified approach we acknowledge that we cannot claim, or rely on, any of the theoretical guarantees or conclusions outlined in Domingos [2000]. However, such simplifications may prove useful in understanding generalisation error in GP and other learning algorithms.

It is typical of GP experiments that as training performance improves (decreasing bias), testing performance deteriorates (increasing variance). As the purpose of training is to gain information concerning the target function from the training data, sensitivity to the training data is essential, and generally more sensitivity results in lower bias [Friedman, 1997]. However, this in turn increases variance, and so there is a natural “bias-variance trade-off” [Geman et al., 1992] associated with function

approximation and classification tasks.

Figure 1.1 illustrates this bias variance trade-off, where increasing model complexity has the effect of reducing bias in the model while at the same time increasing variance. There is an optimal “sweet spot” where bias and variance are both balanced and relatively low, and where generalisation error (in terms of bias and variance components) is minimised. This trade-off is not entirely unavoidable: Breiman [1996b] demonstrated that by perturbing the input dataset and training classifiers on different samples of the data and then combining the various classifiers with a voting mechanism, it is possible to reduce variance without increasing bias. Hence, the wide popularity of ensemble methods for classification problems.

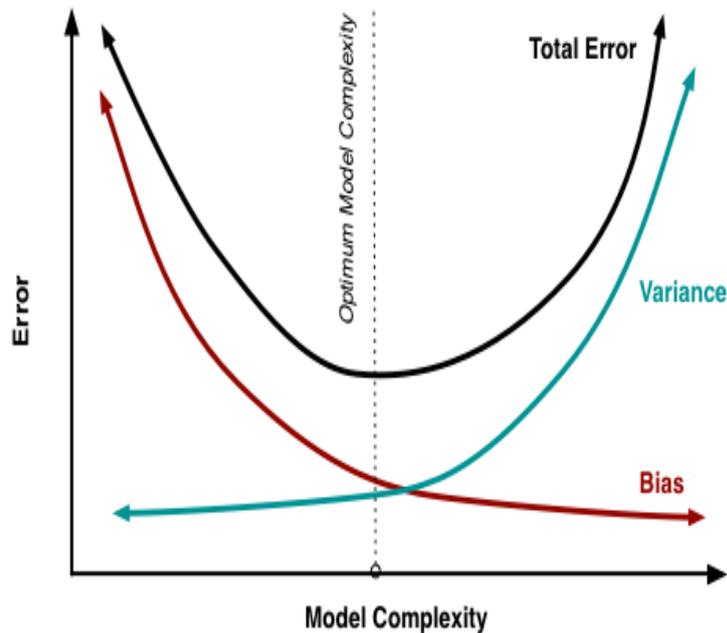


Figure 1.1: The bias variance trade-off [Fortmann-Roe, 2012].

Figure 1.2 illustrates the concepts of bias and variance, as understood in ML, in the context of a set of darts thrown at a dart board.

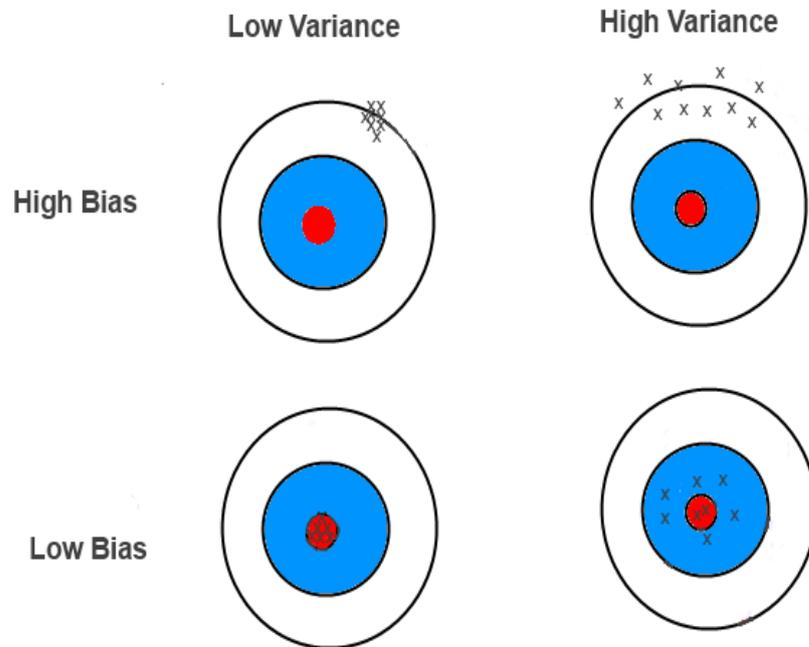


Figure 1.2: Bias and variance in dart throwing.[Domingos, 2012]

Breiman [1996b] put forward ideas about bias and variance as they related specifically to classification. He also introduced the idea of *stable* and *unstable* classifiers, where unstable classifiers are those where small changes in the training set, or in the construction of the classifier, may result in large changes in the constructed predictor. Breiman observed that unstable classifiers are characterised by high variance, and this observation is consistent with the earlier mentioned explanation of Bishop [2006] who also noted that very flexible models have low bias and high variance, whereas relatively rigid models have high bias and low variance. Under these assumptions, we can equate GP classifiers with low bias and high variance.

Up to this point, we have considered bias and variance as components of generalisation error and without focusing on any specific learning algorithm. Now, we continue by considering the term bias in the context of the *inherent* bias of a learning algorithm, with particular reference to GP.

Definition 1.1.1. Inductive Bias [Mitchell, 1980] The set of assumptions that the learner uses to predict outputs given inputs that it has not encountered.

With regard to the inherent bias of a learning system we adopt the term *inductive bias* as defined by Mitchell in definition 1.1.1, and, for the remainder of this section where we refine or expand on the term “bias” we are referring to inductive bias rather than error attributable to bias, unless otherwise stated

The following working definitions are also useful to clarify the particular use of the term “hypothesis” in the next few paragraphs:

Definition 1.1.2. Classification A procedure for selecting a hypothesis from a set of alternatives that best fits a set of observations.

Definition 1.1.3. Hypothesis A particular mapping from feature vectors to class labels: the output of the learning algorithm (model/classifier); *one* of many possible patterns observed in the data.

Definition 1.1.4. Hypothesis Space A set of hypotheses: the space of all *possible* hypotheses considered by a learning algorithm.

Dietterich and Kong [1995] distinguished between *relative* and *absolute* bias, where relative bias means that certain hypotheses are preferred over others, and absolute bias means that certain hypotheses are entirely excluded from the hypotheses space. These terms are sometimes also referred to as *preference bias* and *restrictive bias* respectively.

Those researchers also introduced the concepts of *appropriate* or *inappropriate* bias as further refinements of absolute bias; and *strong* or *weak* bias as further descriptions of relative bias. An appropriate absolute bias has good approximations of the target function in its hypothesis

spaces, whereas an inappropriate absolute bias does not. A weak relative bias allows the learning algorithm to consider too many hypothesis instead of focusing on appropriate hypotheses, whereas a strong relative bias, while it does not rule out the consideration of hypotheses which may be good approximations to the target function, prefers to consider those hypotheses which may be less good.

In other work Whigham [1996] defined three types of inductive bias and mapped those to particular aspects of GP:

1. **Search Bias:** Search bias refers to the factors that control the way in which the learning system transforms one hypothesis into another. In terms of GP this bias is represented by the crossover and mutation operators and the maximum depth of any created program tree;
2. **Selection Bias:** Refers to the method used to select one hypothesis over another. In GP this relates to the chosen fitness function, together with any other selection biases such as tournament size, elitism and replacement strategy;
3. **Language Bias:** The representation language implicitly defines the space of all hypotheses that a learner can represent and therefore can ever learn, as it restricts the forms that those hypotheses may represent. In GP this relates to the function and terminal sets.

For the remainder of the thesis we adopt Whigham’s GP-specific terms of *selection bias*, *language bias* and *search bias*. In addition we make use of the terms *appropriate bias* and *inappropriate bias*, but in a more generic way than that suggested by Dietterich and Kong [1995]: for our purposes, appropriate bias is bias which may make a **positive** contribution towards the evolvability of generalisable solutions, whereas inappropriate bias is bias which may be counter-productive to learning hypotheses which generalise well.

1.2 Central Hypothesis

Considering the second perceived disadvantage of GP as outlined in section 1.1.1 in light of the previous discussion on bias and variance, we

hypothesise that in the context of GP for numerical binary classification¹, there are inappropriate selection, language and search biases which may increase the value of the bias component of generalisation error, which together with high variance associated with the paradigm, inhibit generalisation performance, and that removing or mitigating these inappropriate aspects may facilitate the evolution of generalisable classifiers which are competitive with other ML classification algorithms.

In summary, the central hypothesis of this thesis is that **the reduction of variance and inappropriate bias in GP will lead to the evolution of more generalisable and robust numerical binary classifiers.**

A secondary hypothesis is that dynamic, individualised approaches may have a role to play in reducing the magnitude of error due to bias and variance, thus improving generalisation in GP: natural evolution is a dynamic (if slow) process, yet we commonly implement GP with many static parameters. Secondly, while EC algorithms, such as GP, acknowledge the contribution of the individual through elitism and selection mechanisms - all members of a population are essentially treated the same, i.e. operate under identical conditions. Again, this runs counter to the way things work in the natural world, where varying conditions, of various sorts, may have a strong influence on survivability. In the study of genetics and evolutionary biology the term “genotype by environment (*GXE*) interaction” has been extensively studied and refers to the fact that the best genotype in one environment may not be the best genotype in a another environment and that genotypes differ in their response to environmental factors [Mulder et al., 2013].

Also, Esparcia-Alcázar and Sharman [1999] mentioned the *Baldwin effect* in relation to the influence of learning on evolution in GP. With the Baldwin effect “individuals can be said to pretest the efficacy of different individual designs by phenotypic (individual) exploration of the space of nearby possibilities. If a specific winning strategy is thereby discovered, this will create a new selection pressure: organisms that are closer in the adaptive landscape to that discovery will have a clear advantage over

¹Numerical binary classification is explained in 2.2.2

those more distant” [Dennett, 2003].

Similarly, in Sociology, the term *monolithic bias* introduced by Eichler [1981], refers to possible research bias introduced through “a tendency to treat the family as a monolithic structure, by emphasising uniformity of experience and universality of structure and function over diversity of experiences, structures and functions”. We believe this idea to be strongly analogous with traditional GP approaches, and that the term “monolithic bias” is meaningful in GP as a way of describing bias introduced or magnified through use of a monolithic strategy.

In this work therefore, we have chosen to focus on improving the generalisation performance of GP by tackling variance and inappropriate bias which may arise in various components of the GP methodology, where the bias component may be distributed among the various biases as previously defined by Whigham [1996].

At the same time, where it is feasible to do so, we incorporate dynamic and/or individualised approaches.

In adopting this strategy, we would expect that improving generalisation may also confer benefits on some of the other perceived GP disadvantages as outlined in 1.1.1. For example, it may not be quite the same thing to say that smaller solutions may generalise better as it is to say that programs which generalise well tend to be smaller. In the first instance, a logical strategy, which has been widely adopted in the research, would be to influence program size by encouraging small programs or penalising larger ones. In the latter case, it may be that a necessary condition to produce (small) programs which generalise well may be the existence of larger programs in the system. So, rather than trying to generate smaller solutions in order to achieve better generalisation, one may hope that by trying to improve generalisation, by other means, smaller solutions will naturally emerge. All other things being equal (e.g. solution complexity) smaller solutions will tend to be easier to interpret.

The approach that we have elected to pursue in this thesis is to tackle issues of variance and inappropriate bias by devising novel strategies which can be applied either at the level of the individual program or to

the population as a unit, which can ultimately be combined to produce an optimised GP configuration which offers improved generalisation over standard GP. We develop techniques, each of which aims to either address some *inappropriate bias* or reduce *variance* in binary classification tasks, in the belief that solving or mitigating these challenges will improve performance on unseen data. Some of the challenges undertaken are common to other ML algorithms, whereas others may be particular to GP and may also have application to a wider set of problems.

For the remainder of this chapter, when GP is mentioned, it is assumed that we are talking about GP for numerical binary classification unless otherwise stated.

1.3 Core Research Questions

In considering our central hypothesis, we have decomposed and refined it into several core research questions, each of which addresses selected aspects of generalisation error due to either variance or bias:

- CQ.1 **Variance:** Can novel applications of *bootstrapping*, *validation* and *early stopping* strategies be employed to reduce variance (overfitting) in evolved classifiers?
- CQ.2 **Search Bias:** Can new *individualised* approaches to selection of *boundary values* and *genetic operator probabilities* lead to the evolution of more generalisable solutions?
- CQ.3 **Language Bias:** Is it possible to improve generalisation performance by implementing heuristics derived from a deeper understanding of the contributions of *program size* and *operator complexity* to language bias?
- CQ.4 **Selection Bias:** Can new sampling and algorithmic methods designed to mitigate the *class imbalance problem*, which is influenced by *inappropriate selection bias*, deliver improvements in generalisation?

Where these questions relate to bias, they are *primarily* concerned with the specific type of bias mentioned, but may also deal with other biases

that are deemed to be relevant in the circumstances. Thus, for example, in addressing Core Question 2 the research was primarily concerned with search bias, but those empirical studies also looked at tournament size which may also be considered a source of selection bias.

The main criterion we will use in evaluating the central hypothesis is whether or not the research delivers an affirmative response to one final question: Can an optimised GP classifier deliver results on unseen data which are competitive with other ‘state of the art’ classification algorithms?

1.4 Contribution of the Thesis

In this section, we briefly outline contributions that the thesis makes to the study of GP for numerical binary classification tasks.

The thesis is comprised of five distinct segments of research in which we first address core questions 1 – 4. In the final research segment, promising techniques and approaches arising out of the first four segments are combined and the fruits of the central hypothesis are evaluated. Here we outline separately contributions arising out of each research segment:

1.4.1 Core Question 1: Variance

A novel application of Bootstrapping is applied to GP in Chapter 4 with the aim of evolving individuals whose behaviour on training data is reliable indicator of likely performance on test data. This study produced several interesting research outcomes leading to the following contributions:

- The research demonstrates that the application of bootstrapping, in the manner proposed, produces dramatically smaller programs;
- Results show that the best trained individual is discovered earlier in the evolutionary process when bootstrapping is applied than is the case with standard GP, and that in the majority of cases training performance is a reliable indicator of test performance;
- We establish that there is a stronger negative size fitness correlation

when bootstrapping is used, implying that the suggested application of bootstrapping is bloat resistant;

- When bootstrapping is used, performance of the best of run trained individuals on test data is superior to when standard GP is employed.

Also in Chapter 4 of this thesis we evaluated the over-fitting avoidance capability of an original use of validation sets together with an early stopping mechanism. The completion of this investigation yielded the following contribution:

- The research shows that previously undocumented use of a validation set, to both influence the learning process and serve as an over-fitting reduction measure, can deliver better performance on test data in many cases.

1.4.2 Core Question 2: Search Bias

Two empirical studies investigate the use of various techniques that could be used to determine an effective decision threshold, with the research objective of replacing the observed inappropriate bias with an *appropriate bias*. The research outcomes for this core question provide the following contributions:

- The research highlights previously unreported inappropriate bias resulting from application of traditional zero boundary and fixed polarity settings;
- We provide empirical evidence of improved generalisation when individualised boundaries are preferred to population based settings;
- The research provides experimental evidence that individualised boundaries may encourage more discriminative attribute selection and result in smaller programs;
- We make the previously undocumented observation that ephemeral random constants are not required for good performance on binary classification problems in GP.

Also in Chapter 5 we evaluate a novel application of dynamic individualised crossover and mutation operators. The results of those experiments give rise to the following contributions:

- We highlight the advantages of self-adaptive methods over static ones;
- The research finds that varying selection pressure during evolution may have a beneficial effect on generalisation.

1.4.3 Core Question 3: Language Bias

Chapter 6 of the thesis evaluates some effects of language bias on generalisation. Although the research is far from mature, results arising tentatively suggest the following interesting contributions:

- Previously unreported evidence that variability in GP runs is related to operator complexity: simple operators result in significantly lower variance across runs.
- Previously unreported evidence that a function set consisting of only *+* and *-* produces better *average* results than more complex function sets, tends not to over-fit and may actually produce test results which are *better* than training results. Whereas more complex operators have a tendency to over-fit and that this over-fitting tends to *increase as program size increases*.

1.4.4 Core Question 4: Selection Bias

In Chapter 7 of this thesis we investigate a novel approach to tackling *the class imbalance problem*, and the results of this empirical evaluation leads to the following contributions:

- The research demonstrates a previously unreported effect whereby the performance of GP on binary classification problems *significantly* degrades when the distribution of instances is even mildly unbalanced.
- We provide evidence that a hybrid method which combines sampling techniques with an algorithmic method outperforms many approaches which are solely algorithmic.

- We make the previously undocumented observation that *individualised, proportional* sampling may be superior to previous sampling techniques applied to the class imbalance problem with GP where equal sample sizes were employed.

Also in Chapter 7 we study the effect of several other selection biases on generalisation performance: we compare outcomes for classification performance and over-fitting for both a generational and steady-state replacement strategy across a range of tournament sizes, with and without elitism. The results of the investigation highlight several interesting results:

- The research shows that average solution size is *universally smaller* when a generational replacement strategy is used;
- We show that a non-elitist generational strategy is the least prone to over-fitting;
- Results indicate that a tournament size of 2 produced least over-fitting and that variance error tends to increase with tournament size;

Overall, the results of this particular study suggest that larger tournaments are more prone to over-fitting than smaller ones, and that a small tournament with a non-elitist generational algorithm is least likely to over-fit.

1.4.5 Evaluation

In Chapter 8 several of the more promising approaches which emerged from previous chapters are combined to construct an optimised GP configuration which is compared with several state of the art algorithms. The main observations from this final chapter are:

- Confirmation that an optimised GP configuration can deliver results, in terms of AUC measure ¹ which are extremely competitive with state of the art ML methods.

¹Details of the AUC measure are explained in 2.2.1.5.

- The observation that, at least for binary classification problems, the practitioner may need to leverage knowledge of the *problem type* in order to optimise a GP system. Failure to do so, i.e. trying to use GP “out of the box” is likely to result in vastly inferior solutions.

1.5 Scope and Limitations of the Thesis

The topics of generalisation and classification in GP offer a great deal of scope for academic research, both in terms of breadth and depth. Although this thesis is quite broad in its purview, it is nevertheless necessary to limit the scope of the work in order to properly focus on the key aspects identified in the core research questions outlined in 1.3.

1.5.1 Scope

The scope of this work is restricted to improving the generalisation of Genetic Programming *numerical binary classifiers*. We consider this an appropriate approach which allows us to focus on addressing the fundamental core issues. However, as it has been well established that any n class multi-class problem may be tackled by binary decomposition using n “one versus all” binary classifiers [Loveard and Ciesielski, 2001], it is likely that techniques which improve performance in the binary case may also offer benefits in the multi-class situation where a one versus all strategy is employed. With regard to nominal/categorical attributes, these can quite easily be converted to numerical values if required [Loveard and Ciesielski, 2002].

1.5.2 Limitations

Scalability is an important aspect of any methodology, particularly in today’s world of big data. By its nature, GP is highly suited to parallelism, as in essence every GP run involves a parallel exploration of the search space by the individuals in the population. However, these aspects are outside of the scope of this work.

In recent years Multi-objective GP (MOGP) has also become popular. In contrast to standard GP which typically has either a single objective

or composite objective function, MOGP usually has several *separate* objectives. Instead of a strict linear performance ranking, MOGP solutions are ranked according to the principle of *pareto dominance*. Although the investigation and evaluation of selected methods for reducing variance and bias contained in this thesis may apply to MOGP, explicit consideration of MOGP is outside the scope of this thesis.

Some of the more successful modern classification systems, found in GP and ML involve *ensembles* of classifiers, where individual classifiers are combined with the aim of producing a superior predictive performance than any of the constituent classifiers are capable of. Some of the best known algorithms are *bagging and boosting* [Quinlan, 1996] which are ensemble strategies which can be applied with a wide range of base classifiers.

Ensemble learning is beyond the scope of this thesis, except that we have on occasion compared results with ensemble methods, and also noted that several of our proposed techniques may naturally transfer well to an ensemble approach.

1.6 Outline of the Thesis

The remainder of the thesis is organised as followed:

Chapter 2

Some necessary background information on the topics of classification and Evolutionary Computation is provided in this chapter, where general background information on EC is discussed with a more detailed description of Genetic Programming

Chapter 3

This Chapter provides an overview of some of the important previous work concerning generalisation with a particular emphasis on generalisation as it applies to GP.

Chapter 4

This chapter contains details of empirical studies undertaken which address CQ.1 which is concerned with the mitigation of variance in GP for binary classification tasks: can novel applications of *bootstrapping*, *validation* and *early stopping* strategies be employed to reduce variance (over-fitting) in evolved classifiers?

Here, a system trained with one third training data and one third validation data performed at least as well as a standard configuration using two thirds training data, even though experimental runs using the first configuration terminated after far fewer generations.

Chapter 5

The second of the core questions is covered in Chapter 5 which investigates the search bias component of GP's inductive bias. In this chapter we highlight inappropriate search bias in the standard choice of a population based, zero valued decision threshold ¹ and undertake three empirical studies: two of which are concerned with bias in class boundary determination and one with the application of the crossover and mutation genetic operators.

In the latter we investigate differences between static population based and self-adaptive individualised settings for crossover and mutation parameters. We also evaluate static versus dynamic settings for tournament size. We carry out an empirical investigation of a self-adaptive individualised approach for controlling crossover and mutation probabilities together with a technique which adaptively increases the tournament size during evolution. These two methods are compared with a. a standard GP approach tuned to use mutation and crossover values which were empirically selected, b. a method from the recent literature proposed by [Yin et al., 2007], and c. an approach which employed random selection to set individual mutation and crossover probabilities.

Chapter 6

This chapter is concerned with language bias with a focus on under-

¹This traditional approach is explained in detail in 5.1.

standing the effects of program size and complexity on the generalisation ability of GP. The effects of varying permissible program size and operator complexity are investigated in this study.

It might be argued that maximum allowable depth should be investigated under the heading of search bias. The justification for taking this approach is that questions concerning the effects of program size and program complexity have for a long time been considered together in the research, to the extent that at times the distinction is somewhat blurred. When asking the question “is it program size or complexity which causes over-fitting” it makes sense to consider these aspects side by side.

Chapter 7

In this chapter the empirical research investigates an aspect of selection bias as it applies to the class imbalance problem under a zero-one loss function, where an inappropriate selection bias, when combined with bias in the training data, has a strong adverse effect on classification performance using standard GP. We undertake an empirical study to evaluate several novel approaches to mitigating both of these biases.

In addition we study the effect on generalisation performance of several other selection biases including tournament size and replacement strategy.

Chapter 8

The final empirical investigation is detailed in this chapter, where research outcomes and lessons learned from all of the previous studies are brought together and the main hypothesis evaluation question is responded to.

Chapter 9

The final chapter of the thesis concludes the research with a discussion of the research outcomes. Conclusions are presented together with some ideas for possible future research directions.

1.7 Tools and Datasets

All datasets used in this thesis are taken from the University of California, Irvine (UCI) Machine Learning repository [Bache and Lichman, 2013] unless otherwise stated. Details of these datasets can be found in Appendix A.

The GP experiments undertaken as part of this thesis were programmed using the Open Beagle Framework for Evolutionary Computation [Gagné and Parizeau, 2002].

For those experiments where the research was compared with ML algorithms, and we constructed the ML experiments as part of this research, those experiments were implemented using the Weka data mining software package Hall et al. [2009].

Statistical tests used in this thesis were for the most part conducted using the popular R statistical package [R Development Core Team, 2008].

Chapter 2

Background

In this chapter we provide background information on evolutionary computation with a particular focus on GP, together with an overview of binary classification.

2.1 The Evolutionary Paradigm

GP is a sub-field of Evolutionary Computation (EC) which may be considered a branch of Machine Learning (ML) which can, in turn, be thought of as a sub-field of Artificial Intelligence (AI), more specifically, Computational Intelligence. EC is distinguished from other ML methods by its biological inspiration and adoption of Darwinian principles of evolution. Essentially, EC involves parallel, guided random search of the problem space, where the “guided” aspect is implemented using algorithms which are influenced by ideas from biological evolution.

Although the origins of EC can be traced back to the late 1950s, including the influential research of Friedberg [1958], Box [1957], Bremermann [1962], and others, the field remained relatively unknown to the wider ML community for several decades [Back et al., 1997].

Historically, the cornerstone evolutionary algorithms that we know today, which were eventually unified under the umbrella of Evolutionary Computation, arose between the 1960 and 2000. These were: *Evolutionary Programming* as pioneered by Owens et al. [1966] and Fogel [2009]; Holland’s *Genetic Algorithms* [Holland, 1975]; *Evolution Strategies* orig-

inally investigated by Rechenberg [1973] and Schwefel [1977]; (*Learning Classifier Systems* also put forward by John Holland; Differential Evolution (DE) proposed by Storn and Price [1997]; and *Genetic Programming* which was first proposed by Cramer [1985], and subsequently developed, expanded and popularised by Koza [1990].

Evolutionary Programming, in its original format, used finite state machine representation for candidate solutions, which were recombined or mutated depending on some “pay off” function. However, in more recent years the method has evolved to use arbitrary data representations and has been applied as a general optimisation paradigm. ES and EP differ in their selection processes in that EP typically uses stochastic tournament selection while ES usually employs deterministic selection [Abraham et al., 2006].

Both Genetic Algorithms and Evolution Strategies utilise a linear representation, although in early implementations the data types of the representation vector were different: GAs used a bit string representation whereas ES used vectors of real numbers. The primary search operator in GAs is recombination, whereas in ES, mutation plays a more important role and is usually self-adaptive. In many other respects the two approaches are very similar and have to a great extent converged in recent years. Currently both methodologies are more likely to use whatever linear representation is considered suitable for the problem at hand.

Learning Classifier Systems are re-inforcement learning methods where the initial concept was based around the genetic algorithm, with various, fairly complex rule learning algorithms built on top. Since their inception, the algorithms have evolved and the research has branched off in several distinct directions. A detailed description, review and roadmap for Learning Classifier Systems can be found in Urbanowicz and Moore [2009].

In Differential Evolution candidate solutions are referred to as *agents*. These agents are moved around in the problem search space through the application of various mathematical formulae, and each new agent position is evaluated. If a new location is an improvement on previous ones it is accepted, otherwise the new position is discarded.

2.1.1 Genetic Programming

In common with other evolutionary algorithms, the GP paradigm is based around the Darwinian model of human evolution, where the fittest survive and have a greater opportunity to reproduce, and so pass their genetic material to future generations. From time to time, mutations occur which, while often fatal, occasionally introduce useful new genetic material which leads to some measurable improvement. Thus, many of the terms and concepts outlined in this section are common to other evolutionary algorithms.

The aspect which differentiates GP from most other EC methods is that the population consists of individual *executable programs* historically implemented as Lisp s-expressions. Once executed, each program represents a candidate solution to the given problem.

Typically a GP implementation involves randomly generating a set of computer programs, comprised of functions and terminals, which may be appropriate to the particular problem and where each individual program represents a possible solution to the problem at hand. Traditionally, GP programs are represented using tree based structures like the example shown in Figure 2.1, but various other representations have been used including linear representations. In tree-based GP, each internal node contains an operator and each terminal or leaf node is a place holder for an operand. Tree structures are evaluated substituting in operands *specific to each problem instance*.

In the terminology of GP, this set of computer programs is referred to as a *population*, and the programs themselves are called *individuals*. When *recombination* is discussed, the programs which are selected for recombination are referred to as the *parents* whereas the programs resulting from the recombination process are called *offspring* or *children*.

The example GP tree shown in Figure 2.1 represents a numerical expression. When such trees are evaluated the result is a numerical value, often referred to as the program's *semantics*. GP trees can also be used to represent many other types of expressions such as Boolean expressions.

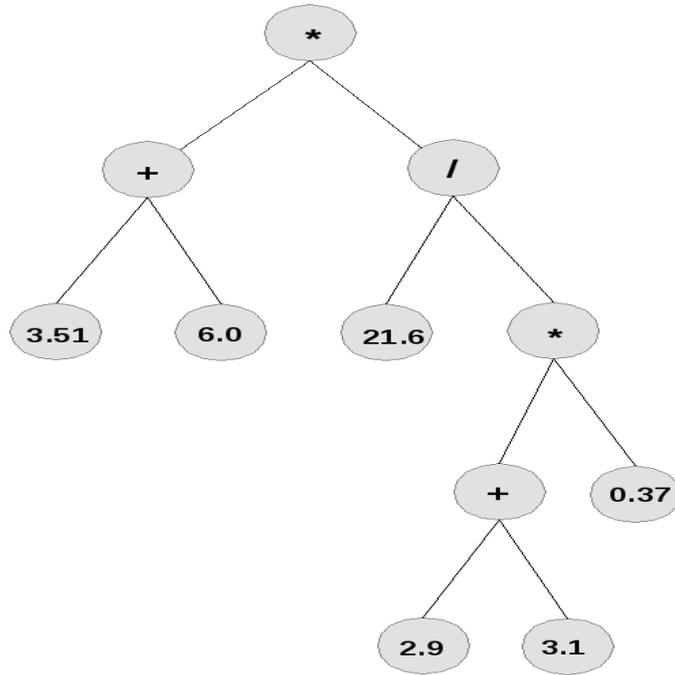


Figure 2.1: Example Numerical Expression GP Tree

2.1.1.1 Exploration and Exploitation

Exploration and exploitation are the two pillars of problem search. In GP as with many other variants of EC, the general view is that exploration is achieved through evolutionary operators: primarily *crossover* and *mutation*, whereas *selection* is the main agent of exploitation - where individuals are selected based on their fitness in order to exploit their evolutionary advantage. Although some[Eiben and Schippers, 1998] have argued that mutation could be viewed as exploitative as it is conservative in its operation: preserving much of the original genetic material.

2.1.1.2 Fitness Function

The fitness function is used to determine how well a particular individual performs on a given task. At each generation, the performance of individuals is evaluated according to some predefined criteria, whereupon the individual is assigned a fitness score, which influences its survivability, chance of mating or being mutated. The fitness function, together with the selection process, guide evolutionary search. From a design perspec-

tive, the choice of fitness function is an important consideration, and how accurately it is aligned with the problem to be solved has a huge influence on the quality of solutions which may be evolved. However, it is this alignment of fitness function to problem which makes GP and evolutionary computation generally a suitable paradigm for tackling a wide variety of different problems.

In the case of binary classification, for example, a possible fitness function would be the percentage of total problem instances correctly classified. GP, and other EC algorithms, are considered particularly suitable for optimisation tasks [Mitchell, 1997]. For classification problems this optimisation may take the form of either maximising the percentage of instances correctly classified, or minimising the percentage of instances incorrectly classified.

2.1.1.3 Genetic Operators

The Darwinian evolutionary model is realised in GP through the use of a fitness function combined with the application of a variety of genetic operators. Given a population of randomly generated individual candidate solutions, this population evolves in an iterative fashion, where at each iteration the fitness of the individuals is evaluated and individuals are probabilistically selected to participate in the next iteration through the application of various genetic operators.

Asexual Reproduction

The simplest GP operator involves copying an existing individual (probabilistically) into the next generation without modification of that individual. *Elitism* can be viewed as a form of asexual reproduction whereby individuals with high fitness from the previous generation are copied unchanged to the next generation. Rather than choosing the elite probabilistically as with standard reproduction, the number of these elite individuals to be copied is usually controlled by a program parameter expressed as a percentage of the population. Without elitism, it is quite possible for the best individual to be lost from one generation to the next when a generational algorithm is employed. Depending on the spe-

Flowchart of a GP Run

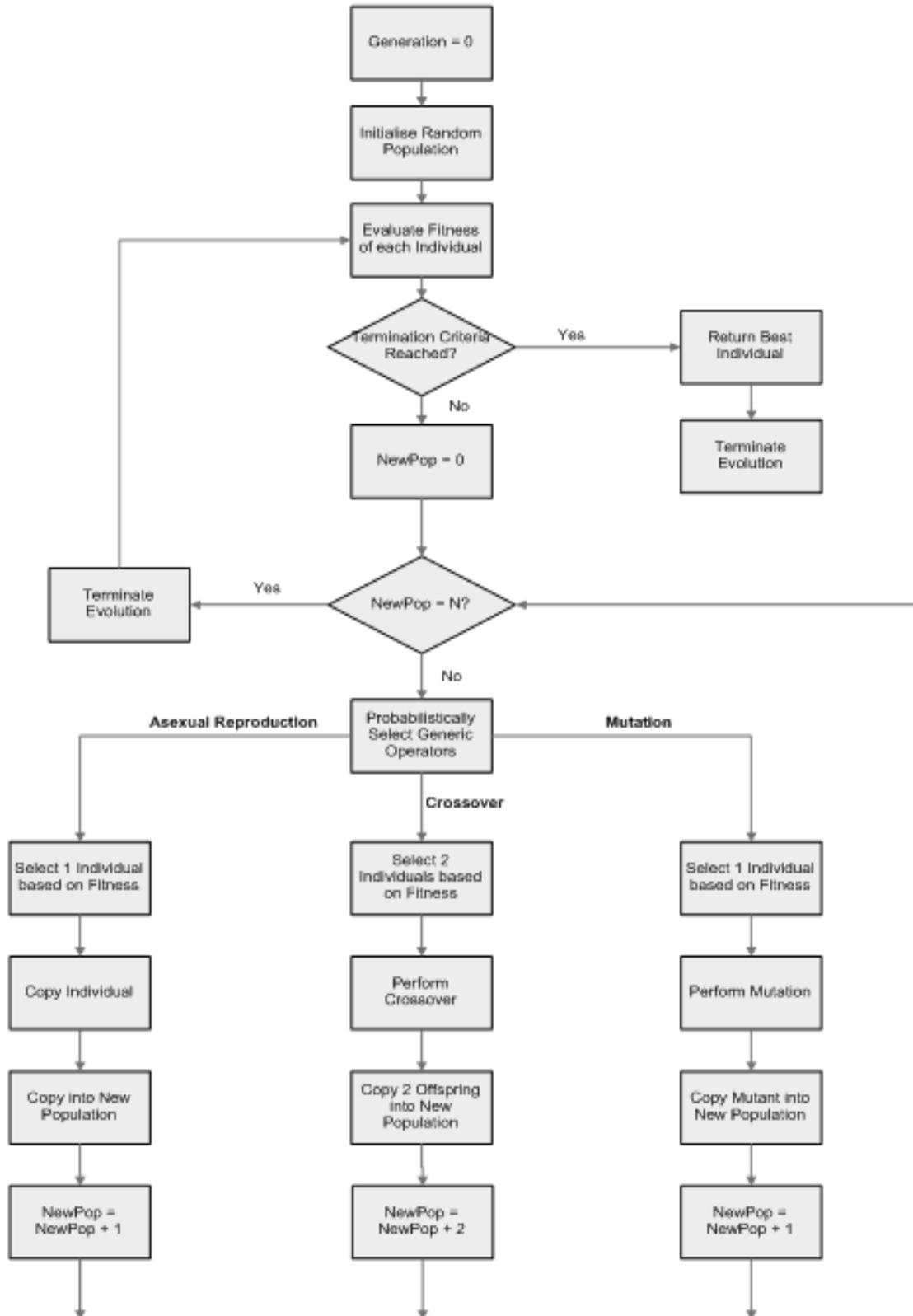


Figure 2.2: Flowchart shows a typical GP run for a population of size N , using a generational replacement strategy and selection based on fitness.

cific implementation, this may not occur when a steady state approach is used, as the best so far individual will not usually be discarded from the population unless a better one has been discovered first.

Crossover

New individuals are created by combining genetic material (program parts) from *parent* programs from the previous generation. When the crossover operator is applied, two individuals are selected from the population to act as parents who provide *offspring* for the next generation. The most commonly used form of crossover is sub-tree crossover as illustrated in Figure 2.3. Using this algorithm, a crossover point (tree node) is independently selected at random from each parent and a new individual is created from a copy of the first parent where the node at the crossover point in that parent is replaced with a copy of the sub-tree rooted at the crossover point of the second parent. This non-destructive form of crossover allows the same individuals to be selected to pass on genetic material multiple times. Some implementations involve the generation of two offspring each time, however, the creation of a single offspring is more common. Also, the concept of *forced improvement* is sometimes applied, whereby offspring are only allowed into the new population if there are fitter than their parents by a predefined margin.

Mutation

New individuals are generated by altering a copy of an individual from a previous generation. The mutation operator can be applied in various ways. Common approaches include *sub-tree* mutation where a node with its sub-tree is replaced with a randomly generated sub-tree, or *point mutation* where the information at a single node is simply replaced with new information. The application of the mutation operator needs to take care that the integrity of the underlying logic is maintained, e.g. that the resulting sub-tree or node value is type safe and has the correct arity. Figures 2.5 and 2.4 illustrate sub-tree and point mutation.

Crossover Versus Mutation

Various studies including those of Spears et al. [1992], Luke and Spector

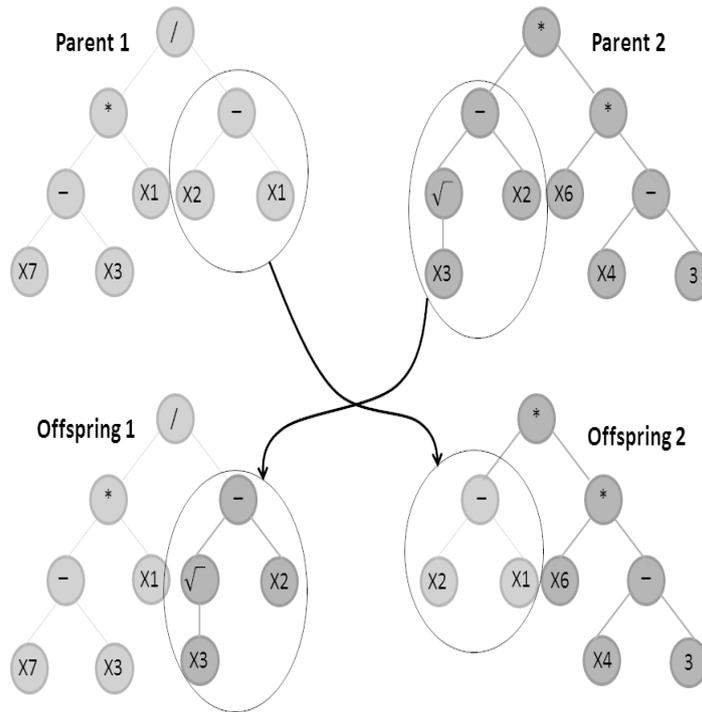


Figure 2.3: Example of Two Parent Crossover

[1997], and White and Poulding [2009] suggest that contrary to popular belief and practice, crossover does not offer any statistically significant advantage over mutation in the search process when studied over a wide range of problems. Spears concluded that “standard mutation and crossover are simply two forms of a more general exploration operator, which can perturb alleles based on any available information”.

Although the overall performance of both operators may be very similar, they behave differently and each has its own strengths: crossover is seen to play a construction role, building new structures from lower level building blocks, whereas mutation plays a stronger role in disruption and exploration [Spears et al., 1992]. Because of these differences, it is usual to use both operators together, and in their experiments, [Luke and Spector, 1997] noted a slight trend towards better results when crossover values in the range 60-70% were used. However, it is still common practice for GP researchers to choose a very small value for mutation when setting experimental parameters.

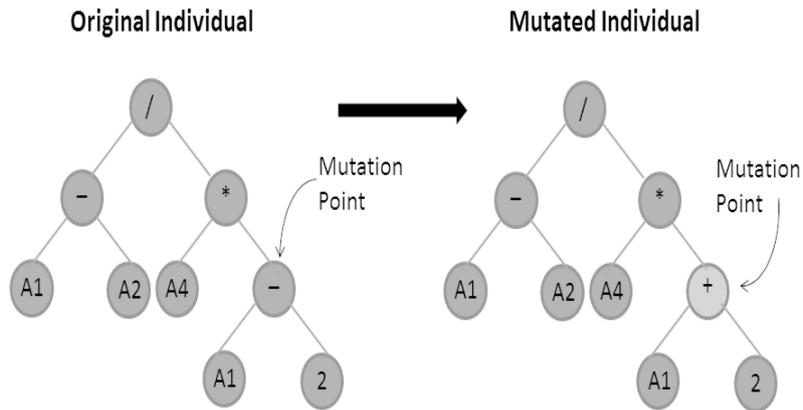


Figure 2.4: Example of Point Mutation

Selection

The selection operator is used to select which individuals from the population will be used for crossover and mutation. Selection methods are usually fitness based. Two examples of selection operators are fitness-proportional and tournament selection. Using fitness proportional selection, an individual has an opportunity of being selected which is proportional to its fitness. Tournament selection is more commonly used. Using this method a pre-set number of individuals (tournament size) are randomly selected to participate in a tournament. The fittest individual wins the tournament and goes on to participate in mutation or crossover. Tournament selection may also be used to select candidates for replacement. In this case, the tournament candidate with lowest fitness is deemed the winner and will be replaced in the population by a newly created individual.

In GP terminology these processes of selection, asexual reproduction,

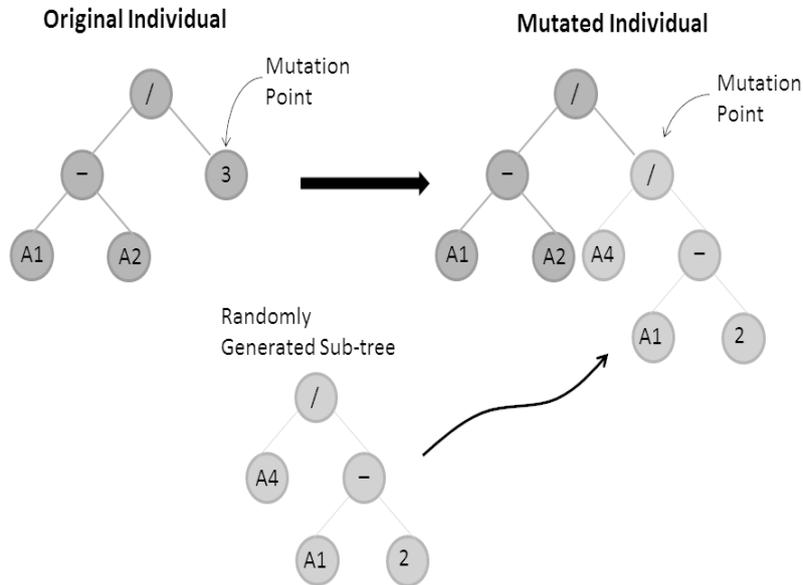


Figure 2.5: Example of Sub-tree Mutation

crossover and mutation roughly correspond to the Darwinian notions of *natural selection*, *survival of the fittest*, *sexual reproduction* and *genetic mutation*. Through these mechanisms, the population evolves increasingly better solutions through the generations until an optimum solution is found, until the individuals in the population are no longer showing significant improvement or until a pre-set maximum number of generations has been reached.

In addition to the genetic operators as they are described here, numerous other variations of mutation and crossover have been proposed in the literature over the years, together with a variety of selection schemes: see, for example, Poli et al. [2008a].

2.1.1.4 Replacement Strategy

There are different replacement strategies which may be employed in Genetic Programming to determine how a population is managed between

one iteration and the next. Two of the most popular are *generational* and *steady state*. Using a generational approach, at each iteration, known as a generation, as evolutionary operators such as reproduction, crossover, and mutation are applied, new individuals are *copied* into a *new* population. Genetic operators are applied to *a copy* of the selected individual, which means that the original is available to be selected multiple times if required.

A steady state strategy involves replacing members of the *existing* population with new individuals. Using this approach, a single population is maintained, and new individuals or offspring are available for immediate selection. New additions to the population replace existing individuals according to some pre-defined scheme: offspring may replace their parents, or weaker members of the population may be selected for replacement. The choice of evolutionary algorithm may have significant impact on the expected outcome of evolution, as different individuals may survive, die, be mutated or allowed to reproduce depending on which algorithm is applied.

In a study comparing genetic robustness of steady state versus generational approaches Jones and Soule [2006] investigated the behaviour of both algorithms on a simple “two peaks” problem. In this type of problem the *fitness landscape* [Wright, 1932] consists of two peaks: a broad low peak and a high narrow peak, where the objective is for the population to discover the tall narrow peak. In their investigation Jones and Soule determined that, when a generational algorithm was used, once the low broad peak was discovered the system required code growth in order to move off that peak onto the taller peak. On the other hand, when a population was encoded using the steady state approach, it naturally made a smooth transition from one peak to the other without requiring code growth. In that context, the term *robust* refers to a feature of individual programs, such that they are relatively impervious to small changes introduced by crossover or mutation, i.e. small changes in their structure will not lead to big changes in their behaviour. In this way, a robust individual, once on the slopes of the high narrow peak, will be less likely to produce offspring prone to “falling off” in the event of a small

change to its structure.

Figure 2.2 illustrates the evolutionary process of a typical GP run, where a generational replacement strategy is used.

2.1.1.5 Functions and Terminals

In GP the term *primitive set* encapsulates the potential expressive power for a given GP *run*. The primitive set consists of the functions, terminals and constants that are available to the GP system for the construction of the individual programs that form a population.

In the literature it is common to describe GP as evolving *programs*, however, in practice, it is not typical to use GP to evolve programs in languages people normally use for software development. It is instead more common to evolve expressions or formulae in a more constrained and often domain-specific language [Poli et al., 2008a]. The definition of the *primitive set* specifies such a language. The primitive set consists of a *function set*, a *terminal set* and optionally either random or *ephemeral random* constants. These components of the primitive set are explained in more detail in the following paragraphs:

Terminal Set

The terminal set consists of the external inputs to the program, each of which is usually represented by a named variable. For example, in a particular classification problem, the inputs may be the results of four blood tests for each of 100 patients. In this case the test results could be mapped to variables named R1, R2, R3 and R4, and when the program is run the actual values for the test results would be substituted in, and the program evaluated with these values. This process of substituting in the four values and evaluating the individual program is done a hundred times, once for each individual.

Function Set

The function set includes any functions which the practitioner feels may be necessary or useful for solving a particular problem. For example, these could be arithmetic, trigonometric or Boolean. However, all sorts

of other functions can be used, such as those found in typical computer programs, such as, “if then else” or looping constructs. Sometimes very specialised functions may be required to tackle a particular problem. For example, if the goal is to navigate a maze, then the function set might include such functions as “turn-left”, “turn-right” or “continue”.

Constants

Depending on the problem it may be desirable to have numerical values in the solution other than those which may exist in the external inputs. This can be achieved by using a primitive which generates a random number. Using this approach the individual program may generate a different random number for that primitive each time it is run, which while it might suit some applications, will generally not be what is required. An alternative is to use what are called *ephemeral random constants*, whereby a constant is initially randomly generated (within a pre-defined range of possible values) and is associated with a particular terminal, once that terminal has been introduced into an individual program tree its value remains fixed, for that program, until the end of evolution.

Closure

Koza [1992] described a concept called *closure* which is necessary for a GP implementation to function correctly. Closure is assured if the two properties of *type consistency* and *evaluation safety* are maintained. Type consistency is necessary to ensure that a valid program tree will always result from the application of the mutation or crossover genetic operators as these may have the effect of disrupting, swapping or re-arranging part/s of a GP tree. It is quite possible to implement GP with mixed types, so long as the system provides machinery for either casting or converting from one type to another, however in practice, when using standard GP, it is more usual to achieve type consistency by enforcing the rule that all of the functions in the function set accept and return the same data type. Strongly Typed GP (STGP) [Montana, 1995] and Grammatical Evolution (GE) [Ryan et al., 1998] are two examples of methodologies which implicitly support mixed types.

Evaluation safety involves taking care to *protect* the behaviour of any operator which has the potential to throw an exception at runtime, such as division by zero for example. A common approach is to return 1 as the result of the operation any time division by zero occurs. An alternative strategy is to catch this type of exception and penalise the guilty individual by giving them a bad fitness score, but this can be problematic if the probability of such exceptions occurring is high, as it may result in many individuals in the population having poor fitness. This will hamper the normal selection process [Poli et al., 2008a] and may introduce an undesirable bias into the system.

Sufficiency

Sufficiency means that the functions, terminals and constants, i.e., the elements of the primitive set are *sufficient* to represent a solution to the problem at hand. This is a necessary, property of any GP system, however without previous knowledge gained through theory and/or experience, it is not always possible to guarantee sufficiency in advance.

2.1.1.6 Initialisation

GP trees can be constructed in a number of ways. Using the *full algorithm*, all trees generated are balanced with a uniform depth, and operators are selected at random, from a predefined primitive set, to populate the internal nodes, terminals are selected, also at random, for the leaf nodes where each leaf node is at the same depth. Alternatively, the *grow method* allows the selection of either functions or terminals for each node of the tree until a specified maximum depth is reached, and at that point only terminals may be added. This results in trees of varying depth as some branches will terminate before the maximum depth is reached.

The *ramped half and half method* is the most commonly used, as it tends to generate a greater variety of tree sizes and shapes. Unsurprisingly, given its name, with this method half of the population are generated using the full method and the other half are generated using the grow method.

2.1.1.7 Parameters

Some important parameters for a GP run are the size of the population, the maximum number of generations that the population will be given to evolve a solution, the initial depth of the GP trees and the maximum depth that the trees are permitted to evolve to. Values for these parameters are chosen by the practitioner to suit the problem at hand.

Another important parameter is the size of the tournament used for selection purposes (in the case where tournament selection is employed). Larger tournaments are said to be more *elitist* as they are more competitive than smaller tournaments: once selected to compete in a tournament, an individual with average fitness has a lower probability of winning a larger tournament than it has a smaller one.

2.1.1.8 Advantages of GP

GP has been successfully applied to a wide variety of problems and problem types [Koza, 2010], however, the method has several attributes which make it (and other EC methodologies) very suitable for certain types of problem situations, which may be less tractable using alternative approaches:

Domain independence

GP is domain independent which means that it can be used to tackle problems for which there is no human expert available, a similar framework can be applied to a wide variety of problem domains without major adjustments, and the GP practitioner does not need to know anything about the domain in question in order to tackle a task effectively [Koza et al., 2004].

Intrinsic Parallelism

GP is suitable for very large search spaces as the algorithm involves a parallel investigation of the search space, where each individual may be potentially searching a different area simultaneously. Being able to tackle problems that have large search spaces is a very desirable attribute, but just as important is the ability to *escape* from sub-optimal solutions or

local minima (or maxima). As GP performs a parallel search, at any given time, there are likely to be some individuals searching what may be considered less promising areas of the search space, i.e., GP does not use hill climbing or conventional greedy search. This means that if some individuals become trapped in a locally good solution (where there is a better, global one available in the fitness landscape) there are likely to be others who are free to continue to find the better solution.

Considering parallelism in the broader computing sense, in comparison with other ML methodologies, GP is inherently parallel [Hoffmeister, 1991] as programs can be modified and evaluated individually. Although selection is carried out on the entire population, which may create a bottleneck that would need to be managed in order to achieve true parallelism.

Flexibility and Expressiveness

Sometimes there are problems which are difficult to formulate mathematically, or for which an exact solution is not well understood. Given a suitable primitive set, GP has the capability to evolve interesting, complex or unusual solutions which human experts would not have considered in advance. According to Angeline [1994]: “genetic programs often acquire elegant solutions with a degree of subtlety not anticipated by the programmer”.

2.1.1.9 Alternative Implementations

Since the emergence of ES, EP, GAs, GP and LCS, new developments have occurred in each of these methodologies, and novel new areas of research have come under the EC umbrella. This can be seen in the development of one of the most important EC conferences: The annual Genetic and Evolutionary Computation (GECCO) conference, which in 1999 had seven different category tracks, by 2013 it had eighteen.

As already mentioned, traditional GP usually uses a tree-based representation, however there are many other representations possible including Linear GP [Brameier, 2005] and Cartesian GP [Miller, 1999]. Canonical GP is generally implemented using a single objective fitness

function, but in recent years Multi-objective GP (MOGP) has also become popular.

Philosophically, one might consider a composite fitness function where two or more objectives are combined into one scalar fitness value [Langdon and Poli, 1998; O'Reilly and Hemberg, 2007] to be multi-objective. However, in practice, when multi-objective GP is discussed, it is in the context of *separate objectives*. In contrast to standard GP which typically delivers an ordered ranked set of solutions, MOGP returns a set of partially ordered solutions, where this partial ordering is achieved through the evaluation of solutions according to the principle of *pareto dominance*. In this case, there may not be one overall best solution, but each of the *set of best solutions* will be no worse than some of the others with regard to some objective and will be better than at least one other solution with regard to one objective. Essentially, this technique allows one to trade off various objectives against each other.

Multi-objective methods using separate objectives were first proposed for genetic algorithms by Fonseca et al. [1993], subsequently the technique was applied to GP by Rodriguez-Vazquez et al. [1997] and others. Details of the concept of pareto dominance can be found in Fonseca et al. [1993].

2.1.2 Related Approaches

In this section we provide a brief description of a few fields of research which are closely related to GP.

Grammatical Evolution (GE)

GE [Ryan et al., 1998] was first proposed in 1998, and has been gaining popularity ever since. This approach facilitates the evolution of programs in any language and is characterised by a separation of the phenotype from the genotype. In GE, the genotype is typically an ordered list of integers which is coded using the modulus operator to select rules from a Backus-Naur form grammar, these grammar rules are, in turn, mapped onto the corresponding phenotype that, similar to GP is realised as a tree structure. The important aspect of GE is this genotype phenotype mapping, where it is possible to encode constraints and desirable bias

into the grammar. The method is independent of the search process, and this function is often carried out using a GA.

Gene Expression Programming (GEP)

Proposed by Ferreira [2001], GEP is related to GP, but like GE it has a genotype/phenotype mapping process. A GEP system is *multigenic* in that each genotype is used to encode multiple parse trees. Because these parse trees are the result of gene expression, in GEP they are called expression trees. Each chromosome is comprised of a sequence of genes which may be mapped to expression trees, where all genes are not necessarily mapped or *expressed*. Although genes are of fixed length, they can encode expression trees of different sizes and shapes.

Swarm Intelligence

Swarm Intelligence is an interesting field of study which has attracted a lot of research interest. Swarm intelligence is a biologically inspired paradigm influenced by swarm and foraging type behaviours in nature, such as bird flocking, bee and ant colonies and animal herding. Methods which adhere to this model include: Ant Colony Optimisation [Dorigo, 1992], Particle Swarm Optimisation [Eberhart and Kennedy, 1995], Bees Algorithm [Pham et al., 2005], Cuckoo Search [Yang and Deb, 2009] and Eagle Strategy [Yang and Deb, 2010] among others.

Artificial Life (ALife)

ALife named by Langton [1986], involves the study of “living” systems and their associated processes and evolution. Such systems may not necessarily be living in the sense that we normally understand it, instead they may be modelled, simulated or synthesised. There are three main branches of ALife: *soft*, *hard* and *wet* which relate to software, hardware and bio-chemistry respectively. ALife may have a useful role to play in understanding relationships between natural and artificial evolution [Wilke et al., 2001].

2.2 Classification

Classification problems are ubiquitous [Weihs and Gaul, 2005], arising in a wide range of situations in everyday life. Indeed, the average person probably encounters more classification situations than they realise every day, particularly ones of a binary nature. For example:

- Email may be classified as spam or not spam depending on the content or source of the message;
- Vehicles may be determined road worthy or not depending on various test results;
- Products on offer in the supermarket have undergone various classification processes to determine if they meet pre-defined standards of quality and appearance;

Taking a higher level view, classification problems arise in many application domains, including Internet search engines, document classification, credit scoring, image analysis, bio-metrical identification and security, manufacturing quality control, medical diagnosis and computer vision [Weihs and Gaul, 2005]. Accurate, fully automated classification offers potential for significant benefits, and has long been an important area of study in Computer Science.

Classification involves systematically assigning each of a given number of data items to a particular category. In machine learning terms, the data item is an *instance* represented by a *feature* vector describing attributes of that instance, the category to which the instance is assigned is referred to as a class, and the attribute that identifies to which class the instance belongs is referred to as the class label. In binary classification, a class boundary is a decision point that determines the class label.

In supervised learning, classification involves learning from a set of labelled instances where the class assignment is *known*, in order to predict the class label of new unlabelled instances of the same type.

2.2.1 Classification Terminology

There are two classes in binary classification tasks, and by convention, those are called the *positive class* and the *negative class*, where for example, the positive class may represent patients that have a particular disease whereas the negative class represents those patients who are free from that disease. In other domains, the positive class usually refers to the *class of interest* which is often the minority class, i.e. the class that has fewer instances. For the remainder of this thesis we will adopt the convention that where we are discussing minority and majority classes we may refer to these as the positive and negative classes respectively.

A classifier is essentially a predictive system, and in this context, a *confusion matrix*, as shown in Table 2.1, is a representation of the behaviour of that predictive system on a given classification task.

Table 2.1: Confusion Matrix

		Prediction	
		Positive	Negative
Truth	Positive	TP	FN
	Negative	FP	TN

Where TP (true positive) is the number of positive instances which have been classified as positive, TN (true negative) is the number of negative instances classified as belonging to the negative class, FP (false positive) is the number of instances actually belonging to the negative class that have been misclassified as positive, and FN (false negative) is the number of positive instances that have been misclassified as negative. The true positive rate is also commonly referred to as the *hit rate*, false positives are also referred to as *false alarms* or *type I errors*, false negatives are known as *misses* or *type II errors*.

There are various measures which are commonly used to describe aspects of classifier performance with regard to a confusion matrix:

2.2.1.1 Precision

Precision is the percentage of instances which the classifier predicted as positive that are actually positive, and is calculated with as per equation 2.1.

$$Precision = \frac{TP}{TP + FP} \quad (2.1)$$

2.2.1.2 Recall

Recall is the percentage of positive instances that were correctly predicted as positive by the classifier as shown in equation 2.2. Recall is also known as *Sensitivity*.

$$Recall = \frac{TP}{TP + FN} \quad (2.2)$$

2.2.1.3 Specificity

Specificity is the percentage of negative instances that the classifier correctly predicted as negative as described in equation 2.3.

$$Specificity = \frac{TN}{TN + FP} \quad (2.3)$$

2.2.1.4 Accuracy

Classification accuracy is the percentage of overall correct predictions as shown in equation 2.4.

$$Accuracy = \frac{TP + TN}{(TP + FN + TN + FP)} \quad (2.4)$$

2.2.1.5 Area Under the ROC Curve (AUC)

Depending on the particular classification task at hand, higher importance may be assigned to the positive or negative class. In the case of spam filtering for example, it is probably preferable to mis-classify a spam email as non-spam (or “may be spam”) than to misclassify an important

email as spam, where in this example it is assumed that spam emails are in the minority.

In the area of medical diagnosis it is usually the case that we want to maximize the true positive rate (sensitivity) and minimize the false positive rate (1-specificity). Originating from World War II where it was used to evaluate the performance of radio personnel at accurately reading radar images, a tool which may be used to measure the performance of medical tests, radiographers, and classifiers and also to examine the balance between the true positive and false positive rates as the decision threshold is varied, is the *Receiver Operating Characteristic* (ROC).

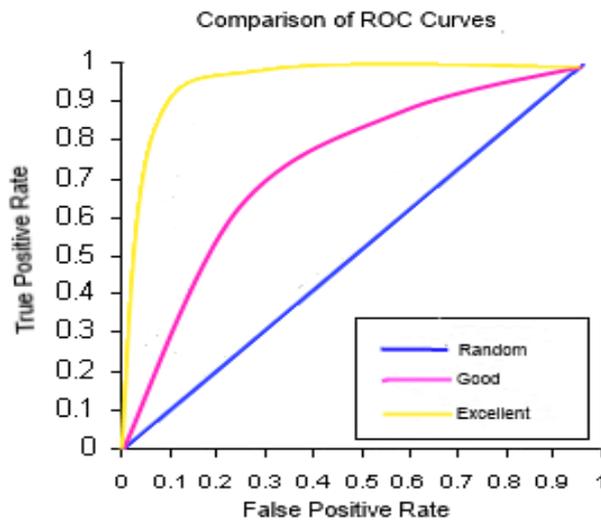


Figure 2.6: Comparison of ROC Curves

In a ROC curve, the true positive rate is plotted against the false positive rate for different cut-off points. Each point on the resulting plot represents a sensitivity/specificity pair corresponding to a particular decision threshold. A “perfect” classifier will have a ROC curve which passes through the upper left corner of the plot which represents 100% sensitivity and 100% specificity. Therefore the closer the ROC curve is to the upper left corner, the higher the overall accuracy of the classifier [Zweig and Campbell, 1993], whereas a curve that splits the plot across the diagonal is no better than random. This is explained in Figure 2.6.

The area under the ROC curve, known as the *AUC* is a scalar value which captures the accuracy of the entity under scrutiny. The AUC is a non-parametric measure representing ROC performance independent of any threshold [Brown and Davis, 2006]. A perfect ROC will have an AUC of 1, whereas the ROC plot of a random classifier will result in an AUC of approximately 0.5.

There are various ways to calculate the AUC including trapezoidal integration ¹ and the Wilcoxon Mann Whitney test.

The latter, also known as the Mann Whitney U test or the Wilcoxon Rank Sum test is simpler to calculate and has the advantage that it facilitates the estimation of confidence intervals [Stober and Yeh, 2007]. Where AUC is used as an evaluation measure in this thesis, it has been calculated using the Wilcoxon Mann Whitney approximation.

2.2.2 Numerical Binary Classifiers in GP

In this section, we provide a brief example of the mechanics of binary classification for numerical data using GP.

Table 2.2: Named attributes and their values for two example instances.

A0	A1	A2	A3	A4	A5	A6	A7	A8	A9
12	6	3.4	9	0.75	2	5.6	13.1	0.34	26
2.04	0.5	9.65	7.89	1.5	19.1	3.56	2.22	1.4	16

Table 2.2 shows a table containing a list of attribute values for two problem instances. Typically, for a particular problem, there will be many such instances, each of which belongs to one of two distinct classes and each individual attribute is associated with a named variable (A0,A1,...). Figure 2.7 shows a possible GP program (classifier) which comprises a set of function and terminal values. Note that not all of the available terminals (attributes) are used in the program: highlighting the ability of

¹Using trapezoidal rule to approximate the area under a curve involves slicing up the area to be found into a number of strips of equal width approximating the area of each strip by the area of the trapezium formed when the upper end is replaced by a chord; the sum of these approximations then gives the final numerical result of the area under the curve [Stober and Yeh, 2007]

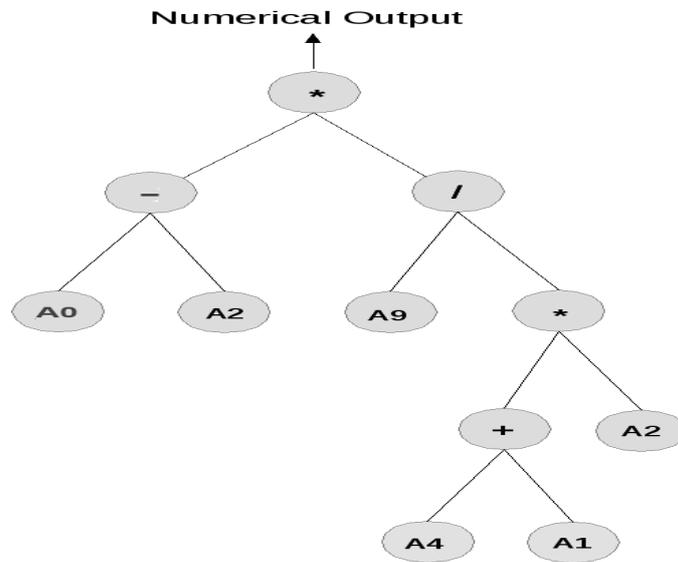


Figure 2.7: Example GP Numerical Classifier

GP to perform feature selection. This program is essentially a *blueprint* that can be applied to any instance from the same task, where the actual values of the attributes will vary from instance to instance.

In Figure 2.8 we can see that the attribute values for the first instance have been substituted in and the parse tree has been evaluated, yielding a value of 9.74. In order for the classifier to determine to which of the two available classes this particular instance belongs, it needs to have a decision threshold available. It is common in GP research to use a threshold of zero: where a program value greater than zero is classified as positive and a program value less than or equal to zero is classified as negative. In this example, the example classifier classifies the example instance as positive.

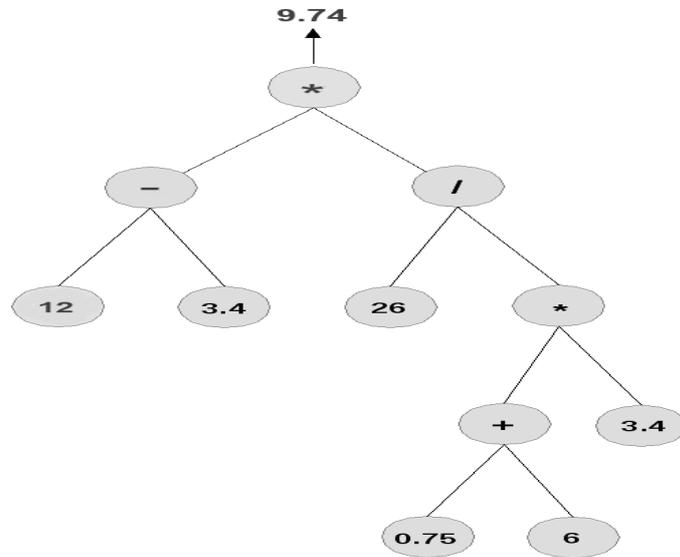


Figure 2.8: Classification of first instance

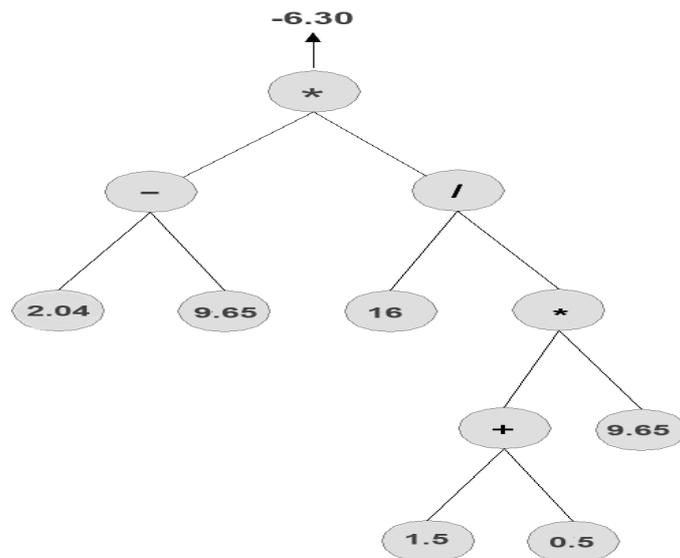


Figure 2.9: Classification of second instance

Figure 2.9 demonstrates that when the example classifier is applied to the second instance, different values for the same attributes are substituted in, resulting in a negative value when the GP tree is evaluated. Using a zero boundary as before, this second instance is classified as negative by the classifier. Note that if the subtraction operator is not part of

the genotype of the classifier, it would not be capable of classifying any instances as negative in the absence of an attribute having a negative value.

2.2.3 Selected ML Classifiers

In Chapter 1 we stated that the principal criterion for evaluating the central hypothesis would be the ability of a GP system which has been modified to reduce variance and inappropriate bias to compete with state of the art ML classification algorithms. In this section we provide details of the ML algorithms selected for this purpose. These particular ML algorithms have been chosen due to their effectiveness documented in the literature, see [Williams et al., 2006] and [Kotsiantis et al., 2006b] for example, and to represent various *types* of ML classification algorithms [Kotsiantis et al., 2007].

- K Nearest Neighbours (kNN);
- Naive Bayes (NB);
- Decision Trees (DT);
- Logistic Regression (LR);
- Multi-Layer Perceptron (MLP);
- Support Vector Machine (SVM).

2.2.3.1 K Nearest Neighbours

A method originally suggested for pattern classification, now known as the k-nearest neighbours algorithm was first introduced by Fix and Hodges [Fix and Hodges, 1989] in an unpublished US Air Force School of Aviation Medicine report from 1951. KNN is one of the simplest, most thoroughly studied and most popular of ML classification algorithms. It is an *instance based learning* method which means that processing of training examples is delayed until a new test instance needs to be labelled. This property has led to instance-based learners sometimes being referred to as *lazy* learners. An effect of this behaviour is that the learner

need not form a global hypothesis over the entire instance space - instead, instance based learners may form a different local approximation to the target function for each test instance.

The inductive bias in kNN is the assumption that the classification of an instance x will be most similar to the classification of other instances that are either *nearby* or *similar in some way* to x . There are numerous distance and similarity measures proposed in the research, but the principle is essentially the same regardless. For the purpose of explanation, we can assume that Euclidean distance is the measure of “nearness” used.

The algorithm works by classifying each new test instances by how *close* it is to instances in the training data. In the special case of $k = 1$ the nearest neighbour algorithm classifies the test instance as being in the same class as the training instance closest to it in terms of Euclidean distance. For $k > 1$ a voting mechanism may be employed, whereby the test instance is classified according to majority vote of its k nearest neighbours.

For kNN, the bias variance trade-off is controlled by the value of k : a low value will result in low bias and high variance, whereas as k approaches n (the number of training instances) bias will be high and variance low, as the new instance will be classified with the same label as the majority class. Although *distance weighted voting* is an enhancement which has the effect of weighting the votes of the closest neighbours heavier than those of more distant neighbours and can mitigate this problem. Breiman [1996a] described kNN as a “stable” learner, by which he means that the algorithm is less sensitive to small changes in the training data than other methods such as Decision Tree learners. Stable classifiers typically exhibit low variance.

Advantages

- Simple implementation with only two parameters to tune: k and distance metric;
- Handles classes which may not be linearly separable;
- Effective when supplied with ample training data.

Disadvantages

- Sensitive to noisy or irrelevant attributes: Distance between instances is calculated based on *all attributes of the instance*, so if there are a lot of attributes with little predictive capacity they may overwhelm the prediction. This problem can be mitigated by selecting or discarding a subset of features;
- Computationally expensive: kNN can be computationally expensive as all processing is deferred until a new instance needs to be classified. This problem has been somewhat mitigated by the use of indexing techniques [Bentley, 1975];
- Sensitive to very unbalanced datasets.

2.2.3.2 Naive Bayes

Naive Bayes classifiers are statistical learners based on Bayesian reasoning which rely on “the assumption that the quantities of interest are governed by probability distributions and that optimal decisions can be made by reasoning about these probabilities together with observed data” [Mitchell, 1997].

Bayes’ Theorem (also known as Bayes’ Rule or Bayes’ Law) forms the foundation of Naive Bayes classifiers. This simply states that given probabilities for A and B; $P(A)$ and $P(B)$ and the conditional probability B given A; $P(B|A)$, the conditional probability of A given B is as described in Figure 2.5

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \quad (2.5)$$

Naive Bayes classifiers assume that the effect of the value of a predictor (attribute) on a given class is independent of the values of other predictors on that class. This assumption is called *class conditional independence* and is why the method is called *naive*. Bayes Networks do not have this assumption. Although Naive Bayes classifiers are relatively simple they are surprisingly effective, and often outperform more sophisticated methods [Rish, 2001]. They are particularly suited to high dimensional problems.

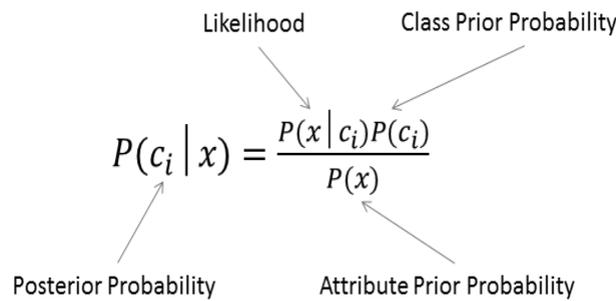


Figure 2.10: Naive Bayes for classification

Naive Bayes classifiers operate by firstly calculating the prior probabilities for each class based on the class label. Prior probabilities for each predictor (attribute) are calculated using the frequency distribution of the attribute from the data available in the training set. Before constructing frequency tables for *numerical attributes*, the attributes can either be converted to categorical values using binning, where values may be grouped together within a specified range, or the distribution of the numerical attributes can be used to estimate the frequency. When using the latter approach, it is common practice to assume a normal distribution.

The decision of how to classify new, unseen instance is made by separately calculating the probability of the instance belonging to each class, given the prior probabilities together with the attribute values of the new instance. Whichever results in the highest probability determines the classification. A useful aspect of this method is that the probabilities may be used as a confidence measure.

Advantages

- Suitable for large datasets;
- Learns from a small number of examples;
- Final probabilities can be used as confidence indicators;
- Fast and Simple.

Disadvantages

- Assumes independence of attributes: cannot learn interactions / relationships between attributes. This can be mitigated by the construction of new features to represent some relationship between existing features.

With regard to the bias variance trade-off, Van Der Putten and Van Someren [2004] categorised Naive Bayes as a low variance learner, with potentially high bias error depending on the configuration: e.g. if feature construction or selection are employed the bias error can be reduced.

2.2.3.3 Decision Trees

Decision tree learning is a very popular method often employed in statistics, data mining and machine learning. This approach uses a decision tree as a predictive model which maps observations about an item to conclusions about that item's target value. There are two basic types of decision tree which differ in the type of predicted outcome: classification trees predict the class label for a classification instance, whereas regression trees predict a numerical value which may represent something like the price of a commodity. Using decision tree structures, leaf nodes represent predicted outcomes whereas internal *decision nodes* specify some test to be carried out on a single attribute value, with one branch and sub-tree for each possible outcome of the test.

For classification purposes, the goal is to determine which attributes are most useful for classifying the data instances, i.e. have the greatest

predictive power. C4.5, which is a popular decision tree learner, uses the concept of *information gain* as a measure of the value of an attribute which describes how well a given attribute separates the training examples according to their target class labels. This measure is used to select among the candidate attributes at each step while growing the tree. Information gain is also known as Kullback Leibler divergence [Kullback and Leibler, 1951], information divergence or relative entropy. Information gain employs the idea of *entropy* as used in information theory.

For binary classification problems, given a set S , containing positive and negative examples, the entropy of set S relative to this simple, binary classification is defined as:

$$Entropy(S) = -p_p \log_2 p_p - p_n \log_2 p_n \quad (2.6)$$

Figure 2.11: Entropy for a two class problem

Where p_p is the proportion of positive examples in S and p_n is the proportion of negative examples in S . Using this entropy value, information gain for a given attribute can be calculated as the expected reduction in entropy caused by segregating the training examples according to that attribute.

Advantages

- Easy to understand and interpret;
- Minimal data preparation required;
- Handles both numerical and categorical data;
- Facilitates statistical validation of the model;
- Can process large amounts of data with reasonable resource usage.

Disadvantages

- The problem of learning an optimal decision tree is known to be NP-complete even for some simple concepts [Hyafil and Rivest, 1976].

Consequently, practical decision-tree learning algorithms are based on heuristic algorithms such as the greedy algorithm where locally optimal decisions are made at each node. Such algorithms cannot guarantee to return the globally optimal decision tree;

- Decision-tree learners can create over-complex trees that tend to over-fit and not generalise the data well. This problem can be addressed by either stopping tree growth early or allowing the tree to be fully grown and then pruning it back. The first approach is problematical as it may be difficult to determine a good stopping point, and in practice, the pruning method is more common;
- There are concepts that are challenging for decision tree learners to learn, because they cannot easily express them, such as XOR, parity or multiplexer problems. Faced with this type of task, the decision tree can become prohibitively large;
- For data including categorical variables with different numbers of levels, information gain in decision trees may be biased in favour of those attributes with more levels [Deng et al., 2011].

C4.5 is a well-known decision tree algorithm which was originally proposed by Quinlan [1986]. Random Forest is an ensemble classifier consisting of many decision trees. This approach has been shown to produce highly accurate classifiers for many problem types. However, one of the main advantages to using decision trees is lost when using this approach, as unlike standard decision trees, the output of random forest may be difficult to understand or interpret.

From the bias-variance perspective Breiman [1996a] categorised Decision Tree learners as “unstable” as they are sensitive to small changes in the training data - small changes in the training data can result in large changes in prediction - high variance. As mentioned previously 2.2.3.3 high variance/over-fitting is a feature of Decision Trees which can be partially overcome by pruning or early stopping, for example.

2.2.3.4 Logistic Regression

Regression is another popular predictive technique. In simple terms, regression allows us to model how the value of one variable, known as the dependent variable (in this case a class label) changes when one or more

independent variables (attributes) are varied. There are various types of regression including *linear regression* and logistic regression. In linear regression, the dependent or target variable is continuous, whereas with logistic regression it is discrete, and usually dichotomous. Linear regression assumes a linear relationship between the dependent and independent variables whereas logistic regression does not. The “logistic” distribution is an S-shaped distribution function which constrains the estimated probabilities to lie between 0 and 1. These factors make logistic regression suitable for probabilistic binary classification.

The result of a simple logistic regression model with one independent variable is an equation shown in Equation 2.7.

$$\text{Log}[p/(1 - p)] = a + bx \quad (2.7)$$

Where p is the probability that an instance belongs to a particular class, a and b are constants which are generated by the model and x is the independent variable. $[p/1 - p]$ is an *odds ratio* which represents the “odds” of a particular class. The fixed constant a is the *intercept* which is the point on the Y-axis (log odds) where the regression line crosses at $x=0$ and the second constant b is the slope of the log odds regression line which is the rate at which the predicted log odds increases or decreases with each successive unit of x , also referred to as the *co-efficient of x* .

In practice where there is more than one independent variable, logistic regression uses the *Maximum Likelihood Estimation* technique in which it tries to estimate the odds that the dependent variable values can be predicted using the values of independent variables. This is done by starting out with a random set of co-efficients, and then iteratively improving them based on improvements to the log likelihood measure. The process terminates after some number of iterations have elapsed or there is minimal improvement, based on some predetermined criteria.

Advantages

- Can provide information on the predictive power of attributes or groups of attributes;

- Outperforms Naive Bayes when training data is plentiful;
- Results are easy to interpret;
- Provides a probabilistic interpretation.

Disadvantages

- Effective performance requires plentiful training examples; the more attributes the greater the sample size required [Hosmer Jr et al., 2013];
- Estimation may be unreliable if attributes are highly correlated.

The behaviour of logistic regression with regard to the bias variance trade-off is dependent on the training data: the method is known to over-fit as the number of independent parameters increases beyond a certain point, whereas insufficient attributes may lead to under-fitting. Typically, if the model is over-fitting a complexity penalty is employed, whereas if the bias error is high new features are added.

2.2.3.5 Multi-Layer Perceptron

The perceptron was invented by Rosenblatt [1957]. A broad definition found in Parker [2003] defines a perceptron as “a pattern recognition machine, based on an analogy to the human nervous system, capable of learning by means of a feedback system which reinforces correct answers and discourages wrong ones”.

The single perceptron has binary inputs, an input bias θ (not shown) and a binary output. The bias is a constant value which essentially functions as a decision boundary. The system operates by applying a weight to each input and then summing these. The output of the perceptron depends on whether the sum of the inputs/weights falls above or below the threshold. If the weighted sum falls above the threshold, the perceptron is said to be *activated*.

The perceptron “learns” by adjusting its input weights depending on how well the achieved output matches the desired/target output.

Single Layer Perceptron

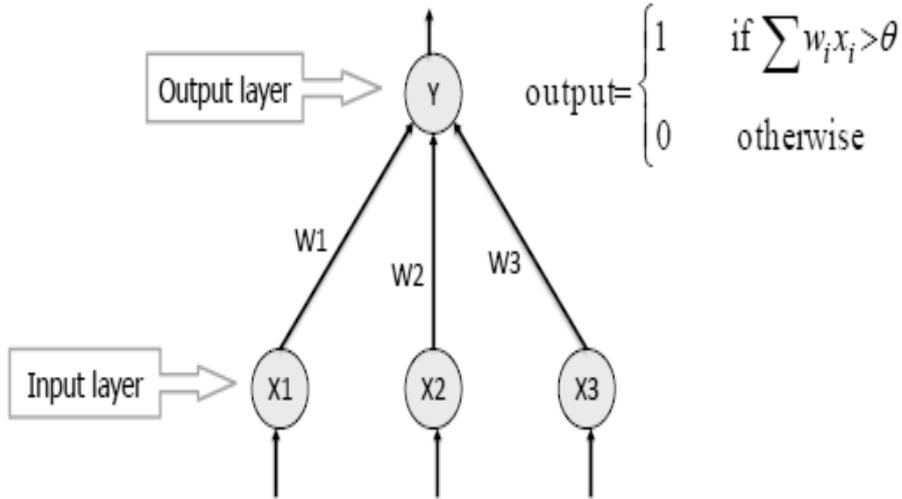


Figure 2.12: Schematic of Single Perceptron. [Sayed, 2012]

$$\begin{aligned}
 w_1 x_1 + w_2 x_2 + \dots + w_n x_n > \theta & \Rightarrow \text{Output } 1 \\
 w_1 x_1 + w_2 x_2 + \dots + w_n x_n \leq \theta & \Rightarrow \text{Output } 0
 \end{aligned}$$

Figure 2.13: Perceptron Activation. [Sayed, 2012]

A *Multi-Layer* perceptron (MLP) is an extension of the simple single perceptron. An MLP is described as a *feedforward neural network* with one or more layers between input and output layer. The term “feedforward” refers to the fact the direction of the data flow is forwards from input to output layer. A single layer neural network is unable to solve problems which are not linearly separable whereas a multi-layer one is. MLPs are widely used for classification, pattern recognition, prediction and approximation.

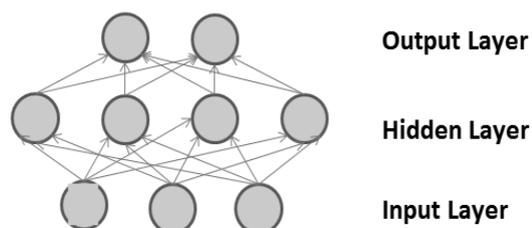


Figure 2.14: Multi-Layer Perceptron

Advantages

- Can solve problems where classes are not linearly separable;
- Can solve problems which have a large number of inputs;
- General and versatile.

Disadvantages

- Relatively slow in both training and test phases;
- Difficult to interpret reasons for classification decisions, and consequently cannot determine relative predictive power of attributes.

Breiman [1996a] categorised neural networks as being unstable, and thus susceptible to high variance. Strategies for reducing variance include early stopping and reducing the size of the network. In the event that the MLP exhibits high bias error, this may be mitigated by growing the network or combining networks.

2.2.3.6 Support Vector Machines

Support Vector Machines (SVM) [Boser et al., 1992; Cortes and Vapnik, 1995] were, in the first incarnation, described by Boser et al. [1992] as simply “a training algorithm that maximizes the margin between the training patterns and the decision boundary”. The algorithm worked by optionally removing what were referred to as *atypical or meaningless* examples from the training data before maximizing the margin. The researchers demonstrated that the resulting classification function depended only on what were originally referred to as *supporting patterns* which are those training examples that lie closest to the decision boundary - these later became known as *support vectors*.

SVMs are suitable for classification problems where the class instances may not be linearly separable in two dimensions: training instances (vectors of attributes) are mapped into higher dimensional space where SVM learns a linear separating hyperplane with the maximal margin in this higher dimensional space by orientating this hyperplane such that it is as far as possible from the closest members of both classes (the support vectors). Part of the major appeal of the method is that even for very high dimensional space, support vectors can be constructed from a very small proportion of the training data which leads to an important conclusion of Cortes and Vapnik [1995] “that if the optimal hyperplane can be constructed from a small number of support vectors relative to the training set size the generalisation ability will be high even in an infinite dimensional space”.

Advantages

- Computationally efficient;
- Good performance on balanced data;
- Good generalisation with balanced data sets: theoretical guarantees regarding over-fitting.

Disadvantages

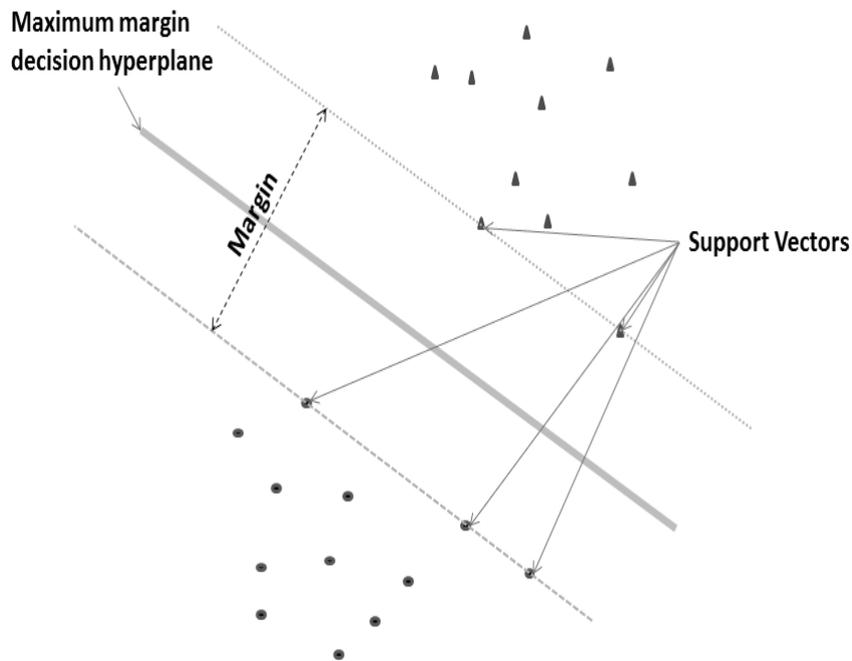


Figure 2.15: Illustration of support vectors for maximal margin.

- May require significant and complex parameter tuning to achieve optimal results [Hsu et al., 2003];
- Poor performance on imbalanced data [Akbari et al., 2004].

In an empirical study investigating error due to bias and variance in SVMs for a variety of kernel functions Valentini and Dietterich [2004] established: for gaussian kernels bias and variance are largely controlled by the *sigma* parameter (width of the gaussian), where small values of *sigma* resulted in high bias and very high values resulted in high variance; for dot product kernels bias was high for small values of the *C* parameter (regularisation penalty) and variance increased with larger values of *C*; and for polynomial kernels, bias and variance depend on the degree of

the polynomial and the value of the C parameter, this kernel behaves similarly to the dot product kernel, but increasing the degree of the polynomial reduced the variance caused by increasing the value of C .

In this section we have briefly described the ML algorithms which will be used in Chapter 8 for comparison with our modified GP implementation, but there are many other ML algorithms that may be used for classification purposes, see for example Lim et al. [2000] and Harper [2005] for details of many other popular methods.

Chapter 3

A Review of Generalisation in Genetic Programming

In this chapter we review previous work on the topic of generalisation. Firstly we explain what generalisation means in the broader context of Machine Learning (ML), before going on to detail some of the significant work that has been published in the field of Genetic Programming (GP) that is either concerned with, or sheds some light on, the evolvability of robust, generalisable solutions, with a particular emphasis on classification problems.

Much of the work mentioned here does not make reference to generalisation as it relates to classification specifically, although a minority have used classification examples in their experiments. In reviewing the research, it can be assumed that the various techniques and strategies mentioned may be suitable for classification, unless otherwise stated.

Details of other work which may be specifically relevant to the core questions outlined in Chapter 1 are discussed in the various chapters which explore those core questions: Chapters 5 to 6.

3.1 Generalisation in Machine Learning

Generalisation is one of the most important evaluation criteria for an ML system [Mitchell, 1997] and a vital concern of ML practitioners is how to develop learners which generalise well [Kovacs and Wills, 2012].

As previously outlined in Chapter 1 under-fitting and over-fitting can be understood to correspond to bias error and variance error respectively, where generalisation error can be decomposed into bias, variance and irreducible error components Breiman [1996b]. In this view of generalisation error, bias is the tendency of a learner to consistently learn the same wrong thing (under-fitting) and variance is the tendency to learn random things irrespective of the true signal (over-fitting) Domingos [2012]. Or as Keijzer and Babovic [2000b] expressed it “the bias component (of generalisation error) reflects the generic ability of a method to fit a particular data set, while the variance error reflects the intrinsic variability of the method in arriving at a solution”. For practical purposes we have adopted the simplification, explained in Chapter 1 where the bias error of a model is the difference between the training performance of the model and the optimal performance (perfect performance less irreducible error), and error due to variance is the difference in model performance between training and test data.

Another way of looking at generalisation with respect to classification is that effective learning for classification tasks is essentially a balancing act between generalisation and discrimination [Wah, 1999] both of which represent different degrees of *restrictive bias*. If the balance is swayed too far in favour of generalisation, under-fitting (bias error) will result, whereas a bias in favour of discrimination will usually result in over-fitting (variance error). Simply put; a model which is *over general* is likely to under-fit, whereas one which is *over specific* is likely to over-fit. Under-fitting will manifest itself in poor performance on training data, while over-fitting will deliver good results on training data that do not translate into good test performance.

For example, supposing we have a training set which consists of various items of information about animals, where the animals in question are cats and dogs of different types, and we want a model that will learn to classify new instances as either cats or dogs. A model with weak restrictive bias, high generalisation and low discrimination, might choose the hypothesis that all animals with four legs and a tail are dogs. This will result in most animals in the training set being categorised as dogs

- a clear case of under-fitting. Now, suppose that a high proportion of the dog instances in the training set are cocker spaniels, a model with strong restrictive bias, i.e. low generalisation and high discrimination might propose the hypothesis that any animal with four legs, a short tail, who barks or whines, is bronze in colour, has long ears, a wavy coat, 42 or 44 teeth and is 20 inches in height is a dog. This model will do very well on the training data as it contains a lot of cocker spaniel instances. However, if the test data is more representative of the dog population as a whole, this model will fail to correctly classify any dog that is not a cocker spaniel - over-fitting. Finally, a model which balances generalisation and discrimination may choose a hypothesis which states that an animal which barks or whines, drinks water, and has more than 40 teeth is a dog.

The term *generalisation* is used in various subtly different ways in ML. For example, in the previous paragraph the intended meaning was the *opposite of specific*. In the case of supervised learning for classification tasks, we want a learner that can learn from a set of labelled examples without significant under-fitting and can use this acquired knowledge to make predictions about future unseen instances of the same general type. Except for the case where a new unseen instance is identical to a training instance, *all classification involves generalisation* [Kovacs and Wills, 2012].

Our goal in ML is to design learners which perform this task *well* such that the model learns the true underlying patterns in the training data, and provides *accurate* predictions on new unseen instances, where any difference between the training performance and test performance is small. Thus, the concepts of under-fitting over-fitting and generalisation are tightly coupled; important strategies in improving generalisation performance are the reduction of both under-fitting (bias error) and over-fitting (variance error).

For the remainder of this thesis the following definitions apply:

Definition 3.1.1. Over-fitting [Mitchell, 1997]: Given a hypothesis space H , a hypothesis h in H is said to over-fit the training data if there exists some alternative hypothesis h' in H , such that h has smaller error

than h' over the training examples, but h' has a smaller error than h over the entire distribution of instances.

Definition 3.1.2. Generalisation [Mitchell, 1997]: A process that takes into account a (large) number of specific observations (inductive bias), and that extracts and retains the important features that characterise classes of these observations.

The issue of generalisation has been an active area of study in the wider field of ML for many years, and several well-established techniques have been shown to be successful. The most important of these are:

1. Regularisation [Bishop, 1995]
Regularisation is concerned with minimising some notion of complexity while at the same time optimising fitting of the data. Regularisation generally involves the application of a complexity penalty;
2. Minimum Description Length (MDL) Strategies [Grünwald, 1997; Rissanen, 1978]
The MDL principle says that the best hypothesis for a given set of data is the one that leads to the largest compression of the data. This may be achieved by setting the inductive bias in a system to favour smaller models;
3. Feature Reduction [Kohavi and John, 1997]
It is generally true that over-fitting increases as the number of features increases as training data as a percentage of the entire input space is decreased. Also, each new feature offers the potential for adding noise to the system. This is particularly true of *irrelevant* features, which do not in fact have any predictive power but may appear to have for some training instances. Identification and removal of such features should reduce over-fitting and consequently improve generalisation;
4. Validation Sets [Dietterich, 1998]
This technique involves using a three data set methodology where a model is trained on a training set and then refined using a validation

set, before it is finally tested on a hold-out test set. In selecting the “best” solution, instead of choosing the model that performs best on the training set only, one which performs well on both the training and validation sets is chosen;

5. Early Stopping [Morgan and Bourlard, 1989]

Early stopping is an approach that aims to avoid over-fitting by curtailing learning according to some predefined heuristic, such as dis-improvement in performance on a validation set;

6. Pruning [Breiman et al., 1984]

This is a technique which may be applied to Decision Tree learners, or other similar methods, which perform greedy search and are known to over-fit. One of several popular solutions to this over-fitting is to allow the model to perfectly fit the training data and then to prune back the tree with reference to a validation set, for example, in order to produce a more general solution;

7. K Fold Cross Validation [Breiman et al., 1984]

In k -fold cross-validation the data is partitioned into k equally sized folds. Subsequently k iterations of training and validation are performed such that in each iteration a different fold of the data is held back for validation while the remaining folds are used for learning. The technique may also be helpful if data is scarce and/or to overcome anomalies in data partitioning.

3.2 Generalisation in Genetic Programming

Despite the well established importance of generalisation in the wider ML paradigm, the focus of a considerable amount of GP research has been on achieving consistency in training results, (where *consistency* means getting predictably repeatable results that have low variance across different runs), possibly in the belief that a high level of consistency would automatically translate to good generalisation. However, some of the evidence would suggest that that this belief is mistaken: GP is a flexible learner and thus likely to suffer from high variance error [Bishop, 2006;

Breiman, 1996b] and a study of the bias variance trade-off effect on GP [Keijzer and Babovic, 2000b] established this empirically. Thus, in the absence of re-mediation, good training results in GP will not map to equally good test results as a general rule.

In recent years several researchers [Costelloe and Ryan, 2009; Eiben and Jelasity, 2002; Gagné et al., 2006; Kushchu, 2002b; Naik and Dabhi, 2013; O’Neill et al., 2010] have expressed the opinion, that in the past, traditional GP research effort may not have put sufficient emphasis on the generalisation potential of a given proposed method. The authors point out that it was not uncommon for experiments to be carried out using a single data set: the training set. In the early days of GP, a great deal of effort was focused on finding new and better ways of tackling benchmark problems such as symbolic regression, parity, Boolean multiplexer and artificial ant. Research effort was focused on establishing that the new paradigm had the capability to solve various problems of different types, and on understanding and developing strategies for doing so, and it is perhaps reasonable that a single data set was used in the early development of the field [Gagné et al., 2006].

Kushchu [2002b], in a review of generalisation in GP, expressed the opinion that until such time as GP researchers adopt the notion of performance evaluation based on *generalisation ability*, genetic based learners and GP in particular would be limited in their scope.

Perhaps this type of criticism plays a necessary part in the natural evolution and development of any new method. Indeed similar practice was common in the early days of ML [Domingos, 2012]. In any case, it is perhaps not entirely fair to chide the early GP research community for lack of focus on generalisation, as there *is* a body of work on the topic from those early days, and one would not expect there to be a large volume of same, as there were far fewer researchers working in the field at the time.

More recently, there has been a strong resurgence of interest in improving the generalisation ability of evolved solutions in GP [Azad and Ryan, 2011b; Castelli et al., 2010; Quang et al., 2010], and to this end many have adopted a strategy of training the learner on a labelled ran-

domly generated subset of data and then evaluating its performance on a test set of unlabelled instances. Many of the methods investigated for improving or promoting generalisation can be aligned with the various ML techniques outlined previously, such as *regularisation*, *MDL strategies*, *early stopping* and *validation sets*.

For clarity, for the remainder of this review previous work is collated under similar broad headings where several have been combined to accommodate significant overlap in some of the cited works. New categories are used to cover sampling strategies and also approaches which may be particular to an EC methodology.

3.2.1 Regularisation and MDL Strategies - Size and Complexity

Rosca [1996] in an analysis of generality versus size in GP, with reference to the well-known Pac-Man game, concluded that smaller programs do tend to generalise better but that methods which control the *effective size* are more likely to be beneficial in promoting generalisation than approaches which apply a generalised size penalty. The results indicated that small, generalisable solutions may be difficult to find when a simple size penalty is applied. Rosca defines effective code as *any code that is executed at least once*, thus constraining the effective size of a given program to be the size of the effective code in nodes.

In other early work Zhang and Mühlenbein [1995] proposed an adaptive fitness function designed to balance parsimony and diversity for the synthesis of neural networks and reported a significant improvement in generalisation performance.

Nikolaev and Iba [2001] proposed an adapted “STROGANOFF” inductive GP system which encouraged small solutions and applied a regularisation penalty to reduce variance in the semantics. They applied their method to data mining and time series data and reported that their regularisation approach eliminated complex solutions from the population and resulted in programs that were “parsimonious, accurate and predictive”. More recently, Vladislavleva et al. [2009a] et al. proposed *order*

of *non-linearity* as a complexity measure, which they used to encourage simpler model *behaviour* without limiting the complexity of the model *structure*. They applied the method on a number of synthetic symbolic regression problems and reported that the generated models exhibited significantly better extrapolative abilities.

Paris et al. [2003] carried out an empirical study comparing the overfitting behaviour of three versions of GP: standard GP, GP using *size fair crossover* [Langdon, 1999], and GP that was boosted [Freund and Schapire, 1996]. They concluded that, all other things being equal, smaller populations of smaller programs were less likely to over fit than larger populations of larger programs. Overall, they found that boosted GP was the least likely to over fit and noted a general rule of thumb: increased computational effort increases the likelihood that the system will over fit the training data.

Gomez et al. [2009] investigated *behavioural complexity* in reinforcement learning, where the concept of behavioural complexity recognised that programs with very similar structure may exhibit quite different behaviour when they interact with the problem environment. The researchers noted that, in general, when training fitness was low - high behavioural complexity had a beneficial effect on generalisation, but as training fitness improved, behavioural complexity was negatively correlated with generalisation.

Program growth without (significant) return in terms of fitness [Poli et al., 2008a] is a widely accepted definition of the term *bloat* as used in the GP community. The phenomenon of bloat is inconvenient for several reasons, including computational expense, increased run times, deterioration in program comprehensibility and a likely dilution of program semantics. Naturally, the topic has been an area of intense and prolific research since the behaviour came to the notice of the GP research community, see [Luke and Panait, 2006], [Silva and Costa, 2009] and [Alfaro-Cid et al., 2010] for reviews of current bloat theories and methods of controlling or preventing it.

For some time, the accepted wisdom in the GP community was that smaller programs may generalise better. This *appeared* consistent with

the general principle of Occam’s razor, and its formalisation in the minimum description length (MDL) principle proposed by Grünwald [1997]. However, in the last few years, the link between bloat and over-fitting has become controversial. Recent work by Vanneschi et al. [2010a] which set out to define the concepts of bloat, over fitting and complexity, and to investigate relationships that may exist between them, suggests that somewhat contrary to popular belief, these phenomena may be independent to some degree. Azad and Ryan [2010] explained that a small GP tree is not necessarily simple: for example, $\sin(x)$ is more complex than a larger GP tree encoding $x+x+x+x$. Related work of Castelli et al. [2011] tentatively suggested that functional complexity, or the lack thereof, may play a more important role in generalisation than bloat does.

In the wider ML context Domingos [1999b] talks about two different interpretations of the principle of Occam’s razor: the first which suggests preferring the simpler of two models with the same generalisation error because simplicity is a goal in itself, and the second favouring the simpler of two models with the same training-set error because this will lead to lower generalisation error. Domingos [1999b] empirically demonstrated that the latter interpretation is false, and in doing so cited numerous instances where simple solutions underperformed and many others where complex methods were superior. In that research Domingos suggested that a more realistic technique for reducing over-fitting is to incorporate domain knowledge in the learning algorithm. Another suggested strategy also proposed by Jensen and Cohen [2000] is to reduce the number of hypotheses considered, which in the case of GP implies that smaller populations may result in solutions which are less likely to over-fit than those originating from larger ones.

Mahler et al. [2005] investigated the effects of *tarpeian bloat control* [Poli, 2003] on generalisation performance, where the tarpeian strategy involves *penalising a random subset* of size constraint breaching offspring. Experimental results of Mahler et al. [2005] indicated that application of the tarpeian method yielded improved generalisation on some tasks but had a negative effect on others.

While it is important to solve the problem of bloat for reasons already

outlined, the close coupling of bloat and program size for the purpose of analysing variance error/over-fitting may be something of a “red herring”. Given the definition of bloat as previously provided by Poli et al., we can imagine that it is quite possible to have small programs which exhibit bloat and large ones which do not. It might be beneficial to focus on bloat, size and complexity separately as sources of over-fitting.

3.2.2 Validation Sets and Early Stopping

Schmiedle et al. [2001] experimented with static, pre-defined early stopping generations and reported that without early stopping, good generalising solutions were lost in later generations as the system tended to over-fit the training data.

An early stopping strategy was also investigated by Tuite et al. [2011]. Tuite concluded that while early stopping may improve generalisation, the use of a “naive early stopping heuristic” can result in terminating evolution *too soon*. This supports the earlier view of Prechelt [1998] who described the difficulty in choosing an appropriate stopping point as “the ugliness of reality” where the classical, idealised, steadily increasing over-fitted curve is absent, instead we may see many “hills” and “valleys” where local minima can occur at various points. Figure 3.1 illustrates this “reality”.

Recent work by Nguyen et al. [2012] investigated the effects of early stopping on generalisation performance and computational effort. They investigated a large number of problems and found the method was very beneficial in reducing run times and delivered some improvement in generalisation.

Thomas and Sycara [1999] experimented with using a validation set to improve performance of financial trading rules. Firstly, they tested the best of run trading rule against the validation set and kept it if it produced positive returns. Next, they also tested all of the trading rules of the final training generation against the validation set and kept any that produced a positive return. The experiments did not yield any noticeable improvement and in several cases the results were actually worse.

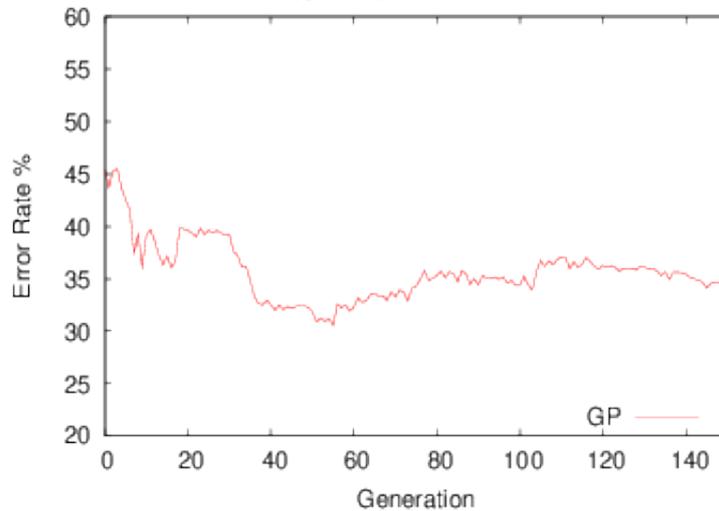


Figure 3.1: Example Validation Error on the BUPA dataset from a Chapter 4 experimental run. An ideal stopping point would be generation 60, however there are several points before that where a naive early stopping heuristic may have terminated evolution.

Robilliard and Fonlupt [2001] proposed *backwarding* where whenever a new best individual with respect to the training set was discovered, this program was also compared to the last best one with respect to an independent validation set. A solution that improved on both sets was then stored and the last such solution was returned at the end of the GP process. The authors tested their approach on a regression task and on a real world remote sensing problem and reported that the system delivered robust solutions.

Foreman and Evett [2005], and later, Vanneschi et al. [2010b] looked at over-fitting measures based on some relationship between training and validation fitness. The latter noted that in the absence of cross-validation, the success of such measures may be dependent on how the data is partitioned. Foreman and Evett [2005] proposed what they called *canary functions* which could be used to detect over-fitting, where a canary function is “distinct from the fitness function but also related toward meeting the same goal as the fitness function”. The authors developed a canary function with encapsulated some measures for detecting over-

fitting previously proposed by Prechelt [1998] for neural networks.

Using Binary Classification for their experiments, Gagné et al. [2006] employed a three data set methodology where the individual program to be evaluated against the test data was selected based on performance on a validation set. They combined this approach with the *lexicographic parsimony pressure* technique of Luke and Panait [2002] which selects the smaller of two solutions in the case where both have the same fitness. Gagne et al. reported that their combined approach drastically reduced the size of best-of-run solutions and decreased the variance in test results. The latter is a useful result for GP due to its non-deterministic nature.

Vanneschi and Gustafson [2009] introduced *repGP* which uses a three data set methodology, but instead of explicitly using good performance on the validation set to select individuals to evaluate against the unseen test data, they look at those individuals who perform *badly* on the validation set. These individuals, together with other “similar” individuals are marked as *repulsors* and are effectively excluded from selection: various dissimilarity measures are used in the selection process in order to ensure that individuals which are different to the repulsors are selected. In other work looking at diversity, Chong et al. [2009] investigated the relationship between diversity and generalisation in co-evolutionary learning in the games domain. They concluded that managing and/or introducing diversity had the potential to improve generalisation.

3.2.3 Sampling Approaches

The work of Gathercole and Ross [1994] has been hugely influential in recent years. In this seminal paper they proposed, (among other related techniques), *dynamic subset selection (DSS)*, whereby different subsets of training instances were presented to the population. They compared their method with a neural network implementation and reported superior generalisation performance using DSS. This approach is both elegant and makes sense intuitively: the GP system would seem to have fewer opportunities to memorise the training instances.

Gonçalves and Silva [2011] applied a similar technique which they have called *Random Sampling Technique (RST)* to a symbolic regression

problem and observed significant reduction in over-fitting when compared with a standard approach.

Langdon [2011] demonstrated that GP is capable of good generalisation using a very small subset of training cases (even a single case). He also noted that many of the successful evolved solutions were “far from parsimonious”. Similar work on sampling was recently proposed by Gonçalves and Silva [2013], where they suggested interleaved sampling of training instances as a generalisation promoting technique. Those researchers also achieved good results by randomly choosing a single training instance at each generation. In their approach they balanced the application of a single instance with periodically using all training data.

3.2.4 Approaches specific to an EC methodology

Early work by Banzhaf et al. [1996] on classification problems, demonstrated that aggressive use of mutation could produce significant improvements in generalisation while at the same time reducing the occurrence of introns in the programs. In further work, they benchmarked their *Compiling Genetic Programming System* (CGPS) against several ML algorithms and reported that their approach generalised almost as well as those other methods but used much sparser training data. Their system outperformed several of the ML approaches, when both were trained with sparse data. CGPS is quite different from GP as described by Koza. For instance, CGPS evolved linear sequences of machine code rather than tree structures. Interestingly, in this early work, the authors employed a three data set methodology, selecting the best performing individual on a validation set to evaluate on the unseen test data.

In other work from around the same time Tacket and Carmi [1994] tackled the “doughnut” problem, which is an artificial classification problem designed to be difficult: the classes are not linearly separable and there is considerable class overlap. They reported good generalisation performance on the problem using distributed evolution, where the population was distributed across a number of small demes (sub-populations) and a novel breeding policy was implemented whereby parents for new individuals in a given deme were selected from other demes in the locality.

In the area of co-evolutionary learning, Juille and Pollack [1998] reported that the existence of an “arms race” between the co-evolving entities lead to improved generalisation.

At a time when very large populations were the norm, Gathercole et al. [1997] demonstrated that GP run with very small populations (50) for a long number of generations could result in better training and test performance than GP with large populations (5000/10000), run for fewer generations. A similar result was reported by Harper [2010] in the aptly named paper: “Genetic Programming - Too much P and not enough G”.

Da Costa and Landry [2006] proposed a *Relaxed Genetic Programming* whereby they *relaxed* the success criteria: instead of requiring an individual to map inputs to a specific output value, success was achieved if the individual produced a target within a given range. This interesting approach is perhaps more suited to problems such as symbolic regression rather than classification tasks.

Another novel approach called *hereditary repulsion* was suggested by Murphy and Ryan [2008]. In this method selection was biased in favour of individuals which had the least amount of common ancestry. This was combined with a constraint which stated that new offspring would only be added to the population if they were fitter than both of their parents. The researchers reported that their system was immune to over-fitting, even late in the evolutionary process.

In an investigation into how best to engineer “robust” GP solutions Panait and Luke [2003] experimented with a range of popular techniques with which other researchers had achieved good generalisation including fitness sharing [McKay, 2000], co-evolution and a range of different approaches to sampling of training instances. They applied these methods to a variety of problem types and concluded that while each technique did improve generalisation capability on some subset of the problems undertaken, none of the methods were universally successful.

In other work Ryan et al. [2005] investigated the use of Run Transferable Libraries (RTLs) as a means of introducing “favourable bias” for learning good functions to solve Boolean problems, where once a run had completed, libraries of learned functions were made available for future

runs.

Jackson [2008] compared the generalisation behaviour of their alternative GP system *Selection Architecture* with standard GP and found that it performed no worse than the standard approach, remarking however that “neither exhibits generalisation powers that could be said to be impressive”.

Costelloe and Ryan [2009] examined the generalisation performance on symbolic regression problems of two seminal GP techniques: *linear scaling* [Keijzer, 2003] and a mating technique which prevents mating between two solutions with the same fitness, commonly known as *no same mates* [Gustafson et al, 2005]. While each of these methods have been shown to reduce bias error, delivering significant improvements on training fitness, the experimental results of Ryan et al. [2005] demonstrated that neither technique improved performance on test data *when employed separately* but that *used together* there seemed to be an harmonious effect whereby a statistically significant improvement in generalisation was observed in three of four problems studied.

Quang et al. [2010] investigated *Semantic Similarity based Crossover* (SSC) on several symbolic regression tasks and found that the method significantly outperformed standard GP with respect to generalisation ability and also improved training performance. The method involves sampling semantics from prospective parents and comparing these according to a semantic similarity measure in the expectation that crossover is most likely to be beneficial if the sub-trees selected from each individual are neither semantically identical nor too semantically dissimilar.

Although mentioned earlier by Zhang and Mühlenbein [1995], the notions of bias and variance error as understood in ML were introduced to the GP community in an in depth manner by Keijzer and Babovic [2000b]. These researchers undertook an empirical study of symbolic regression in GP with an emphasis on ensemble methods in order to learn if symbolic regression, as it is realised through GP, is unbiased and reliable. Their results confirmed empirically the intuition that GP for symbolic regression exhibits low inductive bias and high variance. Interestingly, they observed that small programs had both high bias and

high variance but that if program size was unconstrained and data was plentiful, then results with low bias and acceptable variance could be achieved. Overall, they confirmed the benefits of using ensemble methods for variance reduction. Perhaps disappointingly, there has been very little published work exploring the bias variance decomposition in GP since then. One notable exception is the work of Nikolaev et al. [2002] who proposed *pGP* (polynomial GP), in which they implemented several techniques for reducing both bias and variance error in GP for polynomial regression. Their techniques included a custom regularisation method and a modified fitness function that favoured parsimonious solutions.

3.3 Computational Learning Theory

Computational learning theory is a field of study pioneered by Leslie Valiant [Valiant, 1984] which relies on the concept of inductive bias. In the words of Mitchell : “If consistency with the training instances is taken as the sole determiner of appropriate generalizations, then a program can never make the inductive leap necessary to classify instances beyond those it has observed. Only if the program has other sources of information, or biases for choosing one generalization over the other, can it non-arbitrarily classify instances beyond those in the training set” [Mitchell, 1980].

Valiant developed Probably Approximately Correct (PAC) Learning Theory which, under certain defined circumstances may supply answers to important questions concerning generalisation such as how, why and under what circumstances learning can happen, how many training instances may be required to develop a model that will generalise well on a particular problem or class of problems? The approach is also concerned with providing *provable* guarantees regarding the generalisation performance of learning algorithms.

Kushchu [2002b] provides a detailed description of the PAC learning method and describes how it may be applied to GP. More recently, Kötzing et al. [2011] applied PAC learning to GP in order to analyse its computational complexity.

Recently, Valiant proposed a theory of “evolvability” [Valiant, 2009] which aimed to establish (among other things) which function types are *provably evolvable* and which are not. Computational learning is an important and potentially very useful resource for GP practitioners, which complements existing GP theory, and may help us to reinforce our research with a strong theoretical framework.

As John McCarthy said (of generality) in 1987 ”All this is unpleasantly vague, but it is a lot more than could be said in 1971” [McCarthy, 1987].

Chapter 4

Variance

In this chapter we address the first of the core questions outlined in Chapter 1: “Can novel applications of *bootstrapping*, *validation* and *early stopping* strategies be employed to reduce variance (over-fitting) in evolved classifiers?”.

In answering this question we investigate several novel approaches which explicitly aim at improving the generalisation performance of GP on binary classification tasks, by reducing, avoiding or preventing over-fitting, i.e. reducing the magnitude of the variance component of generalisation error. As previously discussed in Chapter 3, the use of a validation set, early stopping, or sampling techniques such as bootstrapping, are methods which have been used by the ML community to tackle the problem of over-fitting. In this chapter we propose several new techniques based on these ideas and apply them to GP for binary classification problems.

In the first section, we investigate a new application of an existing technique called *bootstrapping* with which we aim to encourage the evolution of models which are *less sensitive* to small changes in the training data. As already discussed in Chapter 1 *unstable* models are those which *are* sensitive to small changes in the training data and are characterised by high variance Breiman [1996b]. Bishop [2006] stated more explicitly that “variance measures the extent to which the model is sensitive to the particular choice of data set”. Thus, in the first section of this chapter, we investigate if a novel application of bootstrapping can help to evolve *stable* models that exhibit lower variance than those evolved

using standard GP.

As previously described in Chapter 3 two well known ML strategies for improving generalisation are the use of a validation set and/or an early stopping mechanism. Essentially, both of these strategies attempt to *avoid* over-fitting; the first by using a validation set so that final model selection is not based solely on training performance, and the second by terminating the learning process *before* the system has had too much opportunity to over-fit.

In sections 4.2 and 4.3 of this chapter we investigate a new application of a validation set as a means of helping to avoid over-fitting. In the evaluated approach, a validation set functions both as a method of providing the system with enhanced learning opportunities and as part of an early stopping heuristic.

In section 4.2 we study the effects of using the proposed approach with a *static* early stopping mechanism, and in section 4.3 we extend and develop these methods to realise a *dynamic* early stopping system.

In summary, we firstly evaluate bootstrapping as a mechanism for reducing variance in evolved solutions, and then in the remainder of the chapter we evaluate a new application of a validation set with a three data set methodology to avoid the potential increase in variance associated with either over-reliance on training data or increased size complexity due to longer evolution.

4.1 Bootstrapping

In this section we investigate a novel application of bootstrapping as a mechanism for reducing variance in evolved solutions.

4.1.1 Research Objective

In this chapter we examine the core question of whether there are reliable novel strategies which can be used to reduce variance in GP. In this section we investigate one such strategy: the leveraging of a well-known method from statistics, known as *bootstrapping*.

Historically, the quality of a solution in Genetic Programming (GP) was often assessed based on its performance on a given training sample. However, in ML we are more interested in achieving reliable estimates of the quality of the evolving individuals on *unseen data*. In this chapter, we propose to simulate the effect of unseen data during training without actually using any additional data. We do this by employing a technique called *bootstrapping* that repeatedly re-samples *with replacement* from the training data and helps estimate sensitivity of the individual in question to small variations across these re-sampled data sets. We minimise this sensitivity, as measured by the *Bootstrap Standard Error*, together with the training error, in an effort to evolve models that generalise better to the unseen data.

We evaluate the proposed technique on four binary classification problems and compare with a standard GP approach. The results show that for the problems undertaken, the proposed method not only generalises significantly better than standard GP while the training performance improves, but also demonstrates a *strong* side effect of containing the tree sizes.

4.1.2 Previous work on Bootstrapping

In statistics, bootstrapping [Efron and Tibshirani, 1994] or *the bootstrap* is a non-parametric technique which can provide a confidence interval for some statistic. In general, its purpose is to assist with non-normal or unknown distributions where the available sample can be used to indirectly assess some properties of the population from which the sample is taken.

Bootstrapping also estimates the sensitivity of a statistic of interest (such as mean) to the given data. This measure of sensitivity is called the *Bootstrap Standard Error* (BSE). For example, to estimate BSE of the mean of a data set of size m , we randomly re-sample from the data set with replacement to create n samples, each sample being of size m . Each of the n re-sampled data sets is called a *bootstrap replicate* or a *bootstrap sample*. Then, for each bootstrap replicate we compute its mean. The BSE, then, is the standard deviation of the means obtained thus from these n bootstrap replicates. Much like standard deviation depicts the

variability in any statistic of interest, the lower the BSE, the lower is the variability in the mean of the *original* data set and hence the higher is the confidence in that mean.

In ML *bootstrap aggregating* (bagging) proposed by Breiman [1996a] is an ensemble method which may be used to improve the performance of ML algorithms on regression or statistical classification tasks. Given an initial training set, bagging generates a number of bootstrap replicates (also known as *bags*) by sampling from the initial set uniformly with replacement. However, then, instead of generating a BSE of some statistic of interest on these bags, typically, the ML algorithms produce multiple solutions each trained on a different bag. These multiple solutions are then pooled: in the case of regression averaging is used, whereas for classification some committee voting mechanism is usually employed.

Inspired by the bootstrap, Oakley [1996] generated bootstrap replicates of GP programs, creating multiple populations from an initial base population, and applied these to chaotic time series data. In 1999 Iba [1999] developed *BagGP* which combined bagging with a co-evolutionary GP approach. Although the performance improvements reported on test data were modest, the results indicated that the method delivered smaller, more robust solutions than those obtained with standard GP when applied to a real world financial problem.

Several GP researchers including [Keijzer and Babovic, 2000a; Song et al., 2005] successfully applied bagging to a range of problems. In general, the bootstrapping component of this work involved applying resampling techniques to *training instances* to create ensembles from functions trained on different bootstrap replicates. Results show that the method was suitable for both small and very large datasets: it allows effective use of limited training data as well as facilitating the use of very large datasets by dividing them into manageable subsets. Similar work by Goncalves et al. [2012] demonstrated that random sampling of training data could reduce over-fitting. Doucette and Heywood [2008] demonstrated that bagging could be used to handle imbalanced data sets for classification tasks by eliminating the class imbalance in the bootstrap sampling phase.

Table 4.1: Bootstrap Experiments: GP Parameters

Parameter	Value
Strategy	Steady State
Initialisation	Ramped half-and-half
Selection	Tournament
Tournament Size	5
Crossover	90
Mutation	10
Initial Min Depth	1
Initial Max Depth	8
Max Depth	20
Function Set	+ - * /
ERC	-5 to +5
Population	2000
Max Gen	200
Bootstrap Replicates	50

This study uses bootstrapping in a very different way. Rather than working with ensembles, in this study we propose to produce reliable *individual* learners that generalise to unseen data. However, there is no reason that individuals thus evolved cannot then be combined in ensembles.

4.1.3 Details of Proposed Technique

In this study, we propose to use bootstrapping to estimate the sensitivity of the evolved models to the training data set. We use this sensitivity as a measure of predictive ability of the evolving solutions on the unseen test data. The proposed approach is different from previous approaches discussed in section 4.1.2 that formed ensembles from the GP individuals trained over *different* bootstrap replicates derived out of a given training set. Thus, in those approaches different GP runs were treated to different subsets of the training set.

In contrast, in this work, we use the *entire* training set for every individual in each run. However, after we score an individual on the training set, we also score the individual under consideration on n bootstrap repli-

centage classification error of the individual on the overall training set, whereas the BSE is the standard deviation of percentage classification errors on n bootstrap replicates.

For this preliminary study we have chosen to use 50 bootstrap replicates to get a reliable estimate of BSE [Efron and Tibshirani, 1994].

4.1.4 Experiments

GP parameters used for the experiments are detailed in tables 4.1. For these experiments we have used the HS, BUPA, BT and WBC datasets. Two hundred runs were completed for each configuration with identical random seeds for each set of experiments. For each task, at every generation, we record percentage error rate for both training and test data for reporting and comparison purposes. In addition, during each run, details of the performance of the best-so-far individual were captured.

4.1.4.1 Bootstrap Configurations

For this preliminary investigation, we have experimented with three different bootstrap configurations in addition to a standard GP set-up. Details of the fitness function for each are outlined in table 4.2. For the standard GP configuration, the fitness measure used to drive the evolutionary process was simply the percentage of misclassified instances, called error rate from now on, whereas for the proposed Bootstrap method the fitness of each individual was calculated by multiplying the error rate by the bootstrap standard error (BSE) of the bootstrap estimates for that individual. After noting the results from standard GP and for GP with bootstrap (BS), we conducted two additional experiments to further investigate the efficacy of incorporating standard error inside the fitness function. The first set-up, termed *BootRandom (BSR)*, generates a random value from the *entire* range of BSE values observed with original bootstrap based runs. Thus, we introduce a noise within the range of the previously observed BSE, but apply it randomly, and investigate whether this noise also has an effect similar to BSE.

Next, in the final set-up called *BootTight (BSRT)*, we generate the

Table 4.2: Fitness Configurations for Bootstrap Experiments

Acronym	Description	Fitness Function
GP	Standard GP	Error %
BS	BootStrap	Error % * BSE
BSR	BootRandom	Error % * random value: range 0.01-0.061
BSRT	BootTight	Error % * random value: range 0.01-BSE

BSE for each individual in the same way as for the BS configuration, but then we select a random value between 0.01 and the generated BSE to use in the fitness function, instead of the BSE itself, thus generating a tighter BSE value which will on average be smaller than the actual BSE. Using these two methods we verify if the original, *true* Bootstrap approach offers benefit over and above just introducing carefully crafted noise.

It may be informative to also compare with methods such as an ensemble based GP or multi-objective GP. However, in this preliminary, proof-of-concept, investigation we restrict ourselves to the methods described above.

4.1.5 Results and Discussion

Figures 4.2 to 4.5 illustrate average percentage error rate on training and test data for each of the problems studied. Looking at these we can see that although overall on training data, standard GP performed better than the various bootstrap methods, on the all-important test data the BS method performs as least as well. Observing the graphs for average training and test fitness side by side, we can see that in the case of standard GP, the gap between training and test fitness is larger than with the BS configuration, and this gap is tending to widen as evolution progresses. This suggests that there is some over-fitting occurring with the standard approach and a stronger correlation in training and test set performances with bootstrap methods.

In addition to the population average test scores, we also examined the test performance of the best-of-run *trained individuals* as shown in table 4.3. Here, we see that the BS method consistently outperformed

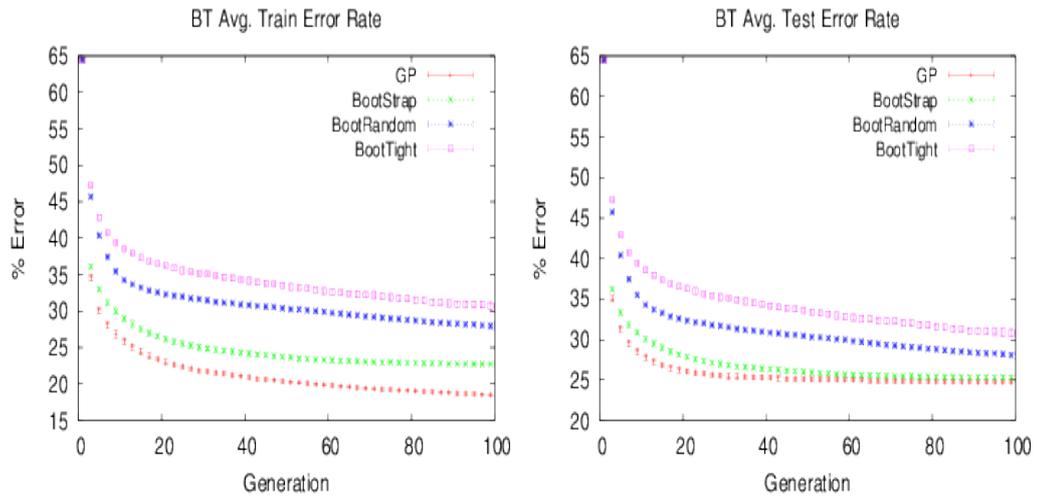


Figure 4.2: Bootstrap Experiments: BT Average Training and Test Error Rates

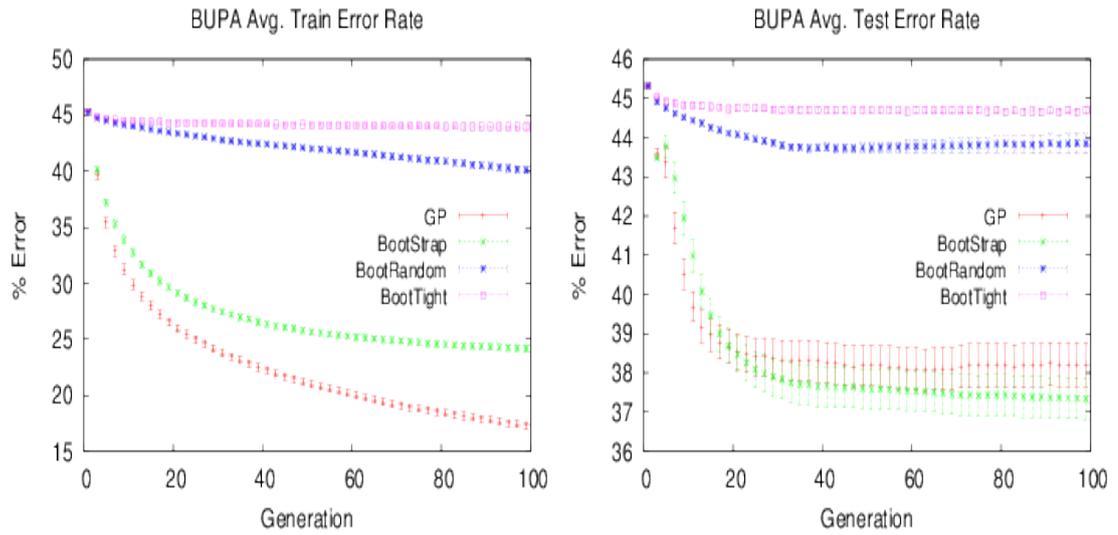


Figure 4.3: Bootstrap Experiments: Bupa Average Training and Test Error Rates

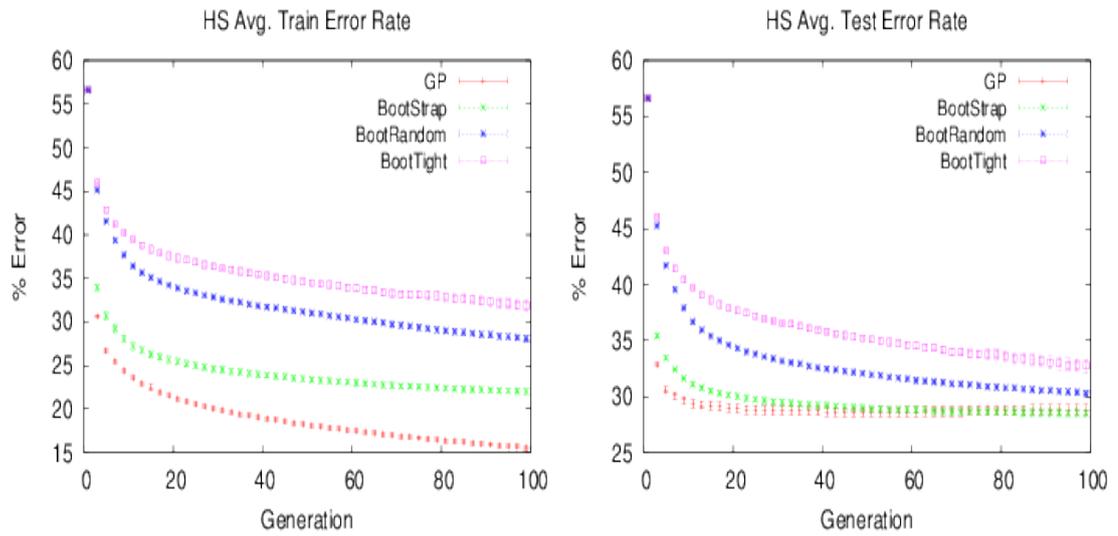


Figure 4.4: Bootstrap Experiments: HS Average Training and Test Error Rates

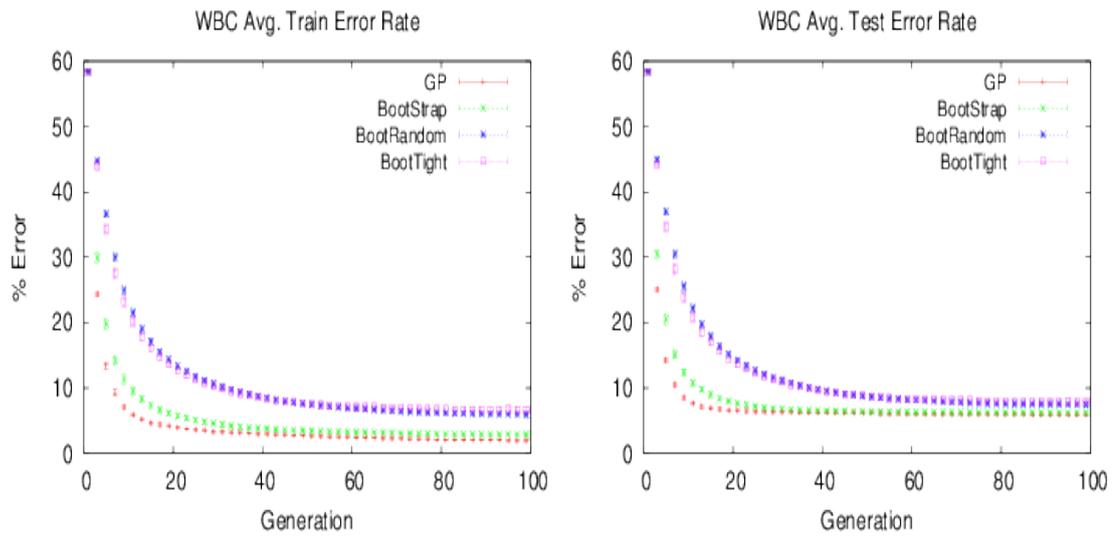


Figure 4.5: Bootstrap Experiments: WBC Average Training and Test Error Rates

GP on each of the four datasets. Tests for statistical significance revealed that the results were significant in all cases using a ninety-five percent confidence interval using a paired Student t-test and a non-parametric paired Wilcoxon signed rank test. Differences between the three different bootstrap methods were not statistically significant. In all cases, the size of the programs generated with standard GP was larger than that of those generated with other methods. Also, note that the best trained individual was on average discovered earlier in the evolutionary process using one of the bootstrap configurations.

4.1.5.1 Program Growth

Perhaps the most interesting (and unexpected) aspect of the results was a dramatic difference in average tree size, as illustrated in Figures 4.6 and 4.7. Even though the experiments were constructed without any explicit growth constraining mechanism, the average tree size obtained when various bootstrap configurations were used was significantly smaller than that with standard GP. In particular, the BSR and BSRT configurations generated very small trees. *Without losing accuracy*, the generation of smaller program trees is a very desirable outcome, offering substantial savings in run times and significant improvements in comprehensibility, particularly when a simple function set is used, as is the case here.

In many standard GP implementations, without explicit bloat curtailment measures, it is often the case that when the system stops learning before evolution ceases, the tree structures continue to grow. Using bootstrap, we see that for the BS configuration, learning tails off somewhere between generations 40 and 60 and that this is matched by a corresponding dramatic slowdown in program growth.

In the case of standard GP, the significant extra program growth does not translate into better test accuracy. Without a detailed knowledge of the reasons for the greatly curtailed program size associated with the BS approach, we cannot be sure if there is a causal relationship between program size and learning outcomes or vice versa.

A deeper understanding of the mechanisms involved is necessary, but one possible explanation of the reduced tree size is that the bootstrap

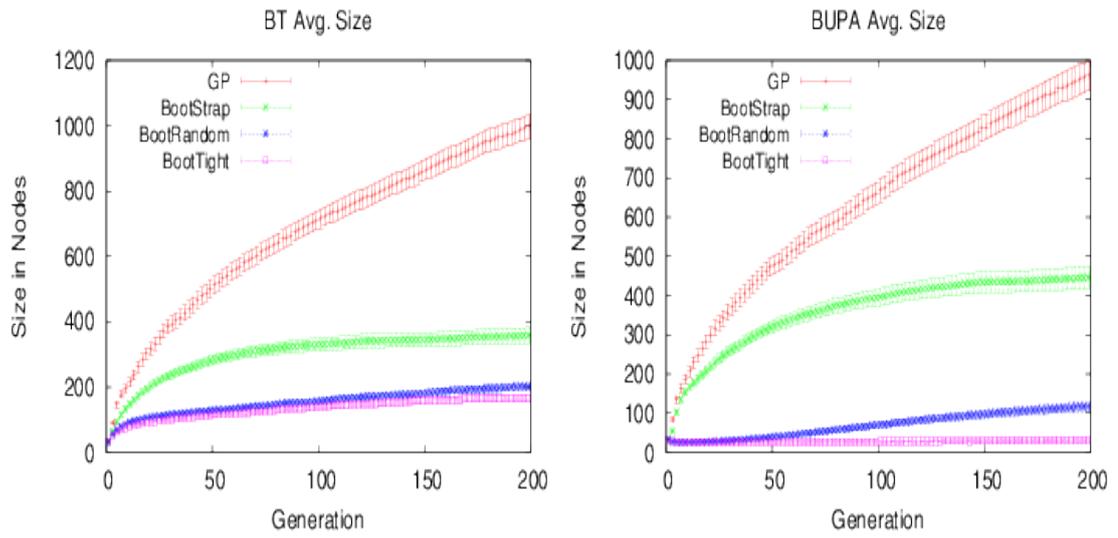


Figure 4.6: Average Tree Size in Nodes: BT and BUPA Data

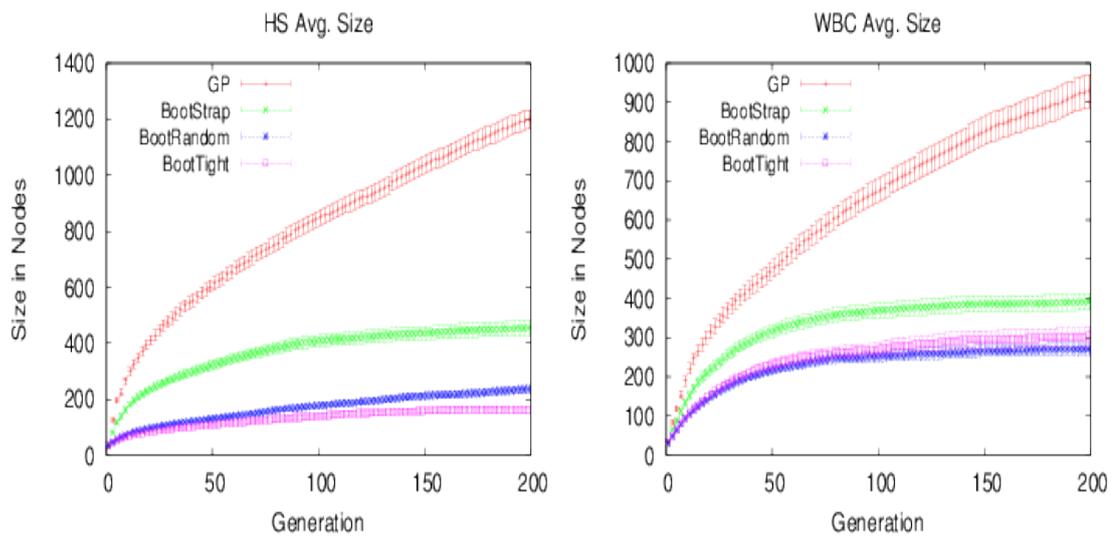


Figure 4.7: Average Tree Size in Nodes: HS and WBC Data

Data	Method	Size	Train Err%	Test Err%	Best Test Err. %	Gen.
BT	GP	941.82	14.38	23.21	19.50	173.18
	BS	342.05	18.42	22.18	19.21	98.20
	BSRT	136.32	18.63	22.00	19.31	73.47
	BSR	132.46	19.72	22.02	19.31	65.46
BUPA	GP	917.42	9.97	37.63	27.33	176.76
	BS	435.38	19.35	33.95	24.42	119.90
	BSRT	49.56	25.60	33.86	25.00	101.94
	BSR	99.83	24.91	35.49	26.75	129.84
HS	GP	1083.53	11.02	28.48	21.06	156.78
	BS	462.91	17.47	25.16	20.39	115.42
	BSRT	164.45	20.32	25.88	19.74	81.53
	BSR	205.40	20.23	25.35	19.74	97.99
WBC	GP	699.18	0.49	5.28	2.65	103.9
	BS	384.66	0.78	3.72	1.47	102.69
	BSRT	269.95	2.14	4.03	2.06	83.86
	BSR	237.46	2.14	4.26	2.06	120.73

Table 4.3: Bootstrap Best of Run Individuals: All values are averaged over 200 Best-of-run trained Individuals for each task, for each configuration.

standard error is likely to be small when an individual has either very high or very low classification accuracy, as it is consistently classifying or misclassifying. If it were also to be the case that both highly fit and highly unfit individuals would tend to have smaller programs, this may have the effect of disproportionately rewarding small programs at the expense of *middle of the road* performers that have larger programs, so that larger programs may be eliminated from the population over time. Note, that regardless of classification threshold, individuals with a very low accuracy are good at separating the classes: simply inverting the polarity of outputs converts a weak solution into an excellent one [Fitzgerald and Ryan, 2012]. Thus, both highly fit and unfit solutions may have similar size distributions.

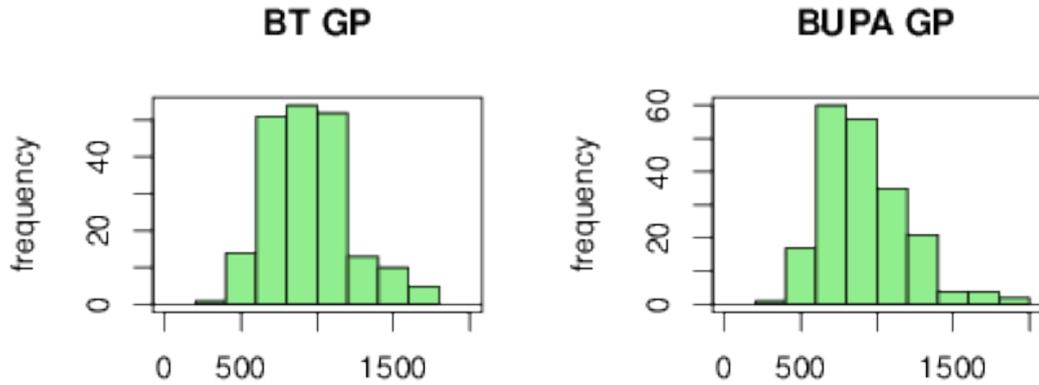


Figure 4.8: Size Frequency Distribution of Best of Run Individuals
GP Configuration: BT and BUPA Data.

This explanation is also possible for the very small trees produced by the BootTight configuration, as the noise value applied instead of the BSE will tend to be smaller than the corresponding BSE, and will thus lead to an even greater disparity in relative fitnesses.

Figures 4.8 to 4.11 illustrate the frequency distribution of program sizes for the 200 best-of-run individuals on each of the four problems, for the standard GP and the BS respectively. Here, we see that for the standard GP configuration, the distributions have roughly “normal” shape, whereas for the BS configuration, the general trend is skewed towards the left, with a higher frequency of programs in the lower range, with the exception of the WBC data. Therefore, not only the BS programs are smaller than GP programs on average but also a greater proportion of BS programs are clustered around the lower end of the distribution, showing a strong bias of BS towards small solutions.

If the suggested explanation for the smaller trees produced with bootstrap were accurate, we would expect to see a reduction in diversity over time as programs with average fitness are squeezed out of the population. Accordingly we captured various information on population diversity during evolution. We recorded measures of *genotypic*, *phenotypic* and *functional* diversity, roughly corresponding to the notions of *struc-*

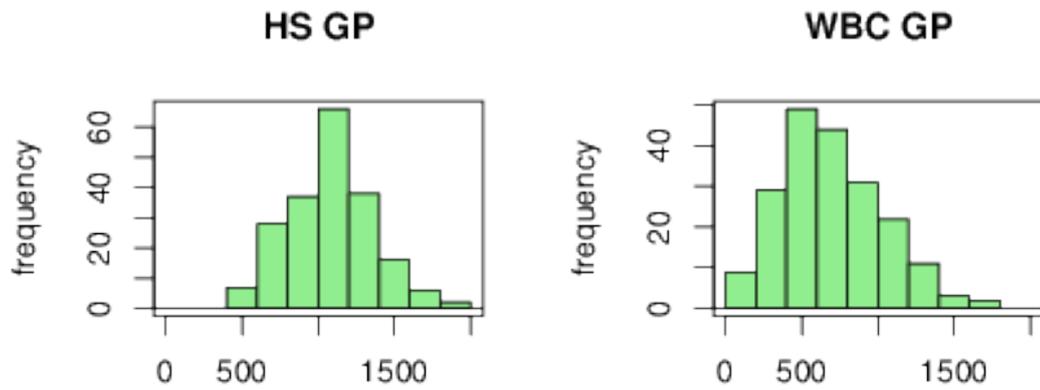


Figure 4.9: Size Frequency Distribution of Best of Run Individuals
GP Configuration: HS and WBC Data.

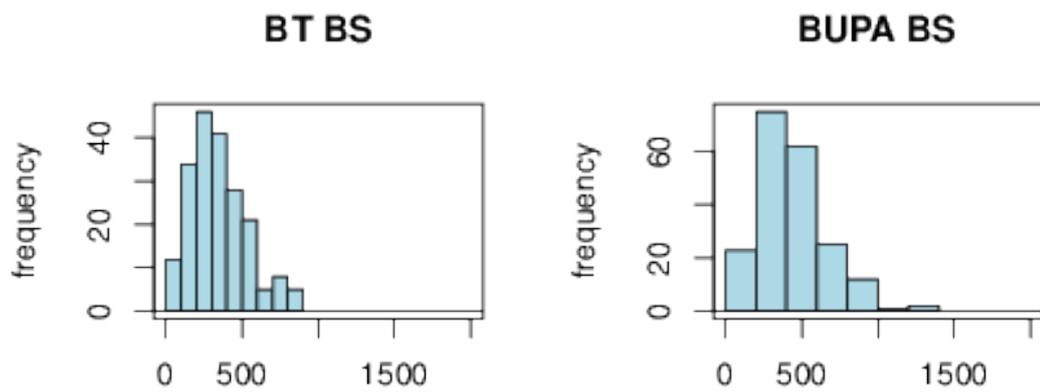


Figure 4.10: Size Frequency Distribution of Best of Run Individuals
Bootstrap Configuration: BT and BUPA Data.

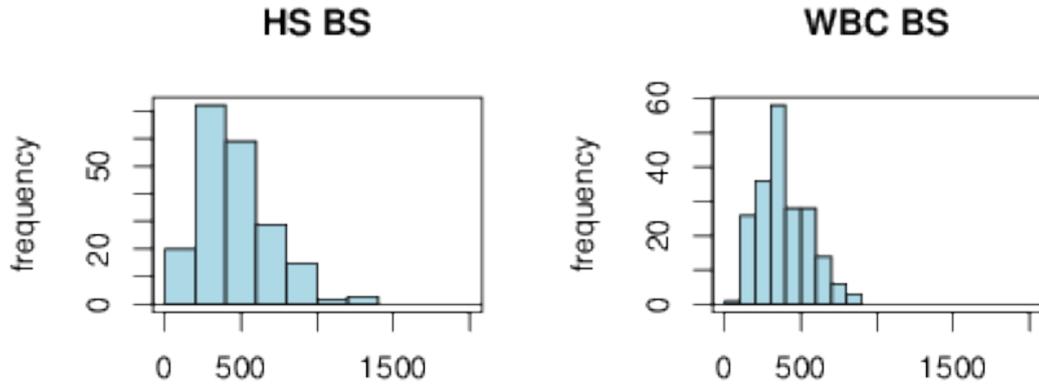


Figure 4.11: Size Frequency Distribution of Best of Run Individuals
Bootstrap Configuration: HS and WBC Data.

tural, behavioural and *fitness* diversity outlined by Jackson [2010].

In measuring genotypic diversity, we compared the individual tree structures and recorded the number of unique trees in each generation. For phenotypic diversity, as each individual was evaluated on training instances, program output for each instance was concatenated in a string. At the end of each generation, the individual strings were compared for uniqueness and the number of unique strings was recorded. This notion of phenotypic diversity is similar to that proposed by Esparcia-Alcázar and Moravec [2013] who deemed solutions to be similar if “when placed in the same environment they perform the same action”.

The measure of functional diversity captured is simply the number of unique fitness values in each generation. While each measure taken in isolation provides a coarse and not very insightful indication of population diversity, taken together they provide a useful guideline.

For both genotypic and functional diversity, similar values were achieved across the various configurations for all of the problems undertaken: genotypic diversity of between 90 and 100% was maintained throughout evolution, whereas values between 40 and 60% were typical for functional diversity. There was a greater variety in phenotypic diversity when the

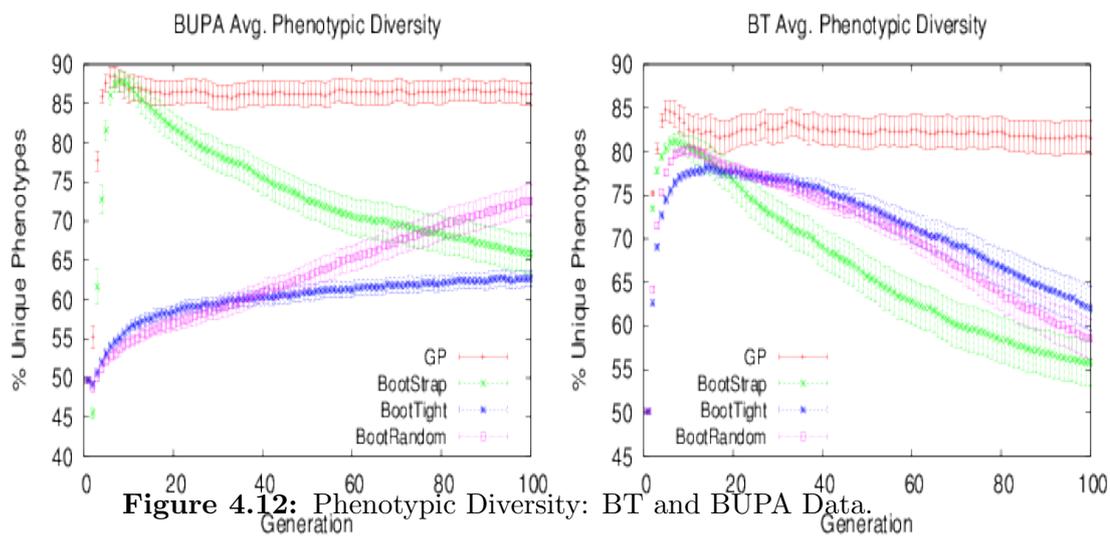


Figure 4.12: Phenotypic Diversity: BT and BUPA Data.

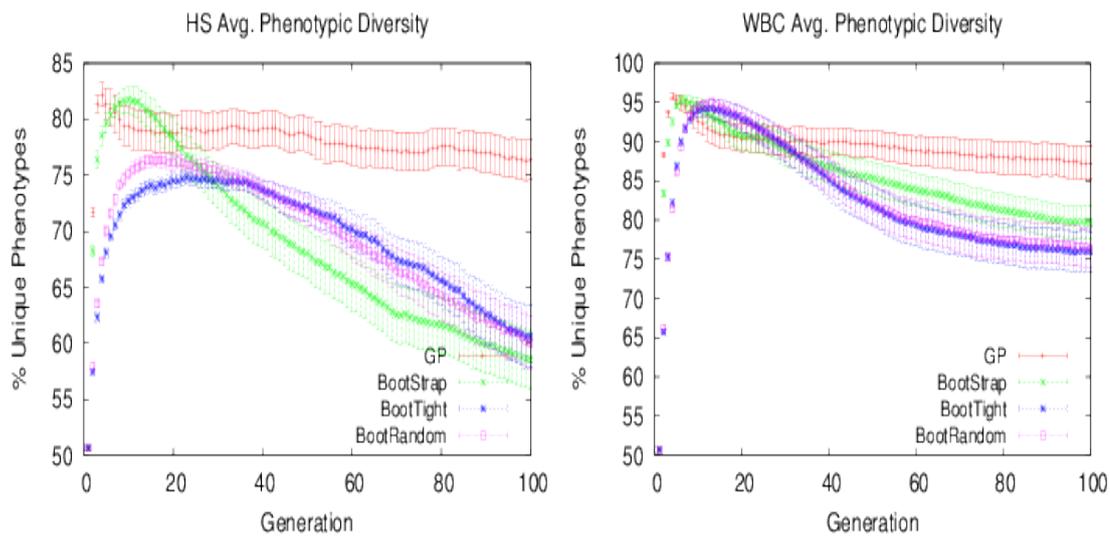


Figure 4.13: Phenotypic Diversity: HS and WBC Data.

different configurations were employed. The percentage of unique phenotypes in the population during evolution is captured in Figures 4.12 and 4.13.

It is usually recommended that a high level of diversity is maintained in the population, or at least, that potentially catastrophic losses in diversity are avoided [Poli et al., 2008a]. For our experiments, standard GP maintained a high level of phenotypic diversity throughout, whereas the diversity of the BS populations fell off steadily but to differing degrees depending on the problem undertaken. However, this loss in diversity is clearly not a disadvantage for the BS method as it seems that this method quickly converges to solutions that generalise better.

Data	GPTrain	BSTrain	GPTest	BSTest
BT	-0.75	-0.83	-0.51	-0.74
BUPA	-0.94	-0.99	-0.56	-0.83
HS	-0.84	-0.84	-0.28	-0.74
WBC	-0.53	-0.75	-0.42	-0.68

Table 4.4: Average training, test error correlated with average tree size, GP and BootStrap configurations. Values closer to -1 indicate a stronger relationship between increasing program size and decreasing error.

4.1.5.2 Size Fitness Correlation

Looking at the plots for program growth and fitness we see that both of these level off at approximately the same point in evolution with the BS set-up. Accordingly, we carried out some statistical tests with the GP and BS run data to investigate if there was a difference in correlation between program growth and fitness. We measured Pearson’s product moment correlation between the average fitness (train and test) and average tree size at each generation across all 200 runs. The results shown in Table 4.4 indicate that for training fitness, in three out of four cases there was a stronger correlation when the BS method was used, and for two of these it was appreciably stronger. The correlation for test fitness was much stronger for all of the datasets.

As program growth is seen to taper off at approximately the same time

as accuracy on test data when using BS, there may be a possibility that a consistent reduction in the rate of growth could be used as an effective early stopping measure, leading to significant time savings, potentially reducing over-fitting, and mitigating the need to guess an appropriate terminating generation.

In the context of our earlier stated goal of developing classifiers whose training performance could provide a more accurate indication of behaviour on unseen data, i.e. low variance error: it seems that (for the problems undertaken), training scores achieved using the BS method provide a better indication of test performance, as the difference between these outcomes is smaller than when standard GP is used. For the latter experiments, there is clear over-training on both the BUPA and HS datasets, each of which produce test scores significantly worse than the average training scores, whereas for the BS experiments the average best test scores are at least as good as the average training scores for all of the datasets except BUPA.

If we accept the earlier definition of bloat in [Poli et al., 2008a] as *program growth without (significant) return in terms of fitness*, we would argue the harmonious tapering off of growth and fitness, strong negative size/fitness correlation and dramatically smaller trees, is evidence that the BS method produces less bloated solutions than standard GP, without any compromise on test accuracy, and that these solutions may also be less likely to over-fit to the same degree.

4.1.5.3 Comparison with other Methods

In this section we detail results obtained using a variety of ML algorithms to classify the four datasets with the Weka ML tool [Hall et al., 2009]. The algorithms that we have tested are detailed in Table 4.5. Note that both AdaBoostM1 and Bagging use the REPTree fast decision tree classifier as their base classifier.

The performance of these algorithms on test data, is compared with that of the BS and GP test scores of best-of-run individuals averaged over two hundred runs. For compatibility the results are displayed as % Classification Accuracy, that is, 100 - classification error%. Where different

Method	Acronym	Description
J48	J48	Implementation of C4.5 decision tree algorithm
Random Forest	RF	Ensemble decision tree classifier
libSVM	SV	Support vector machine
Multi-Layer Perceptron	ML	Feed-forward ANN
Naive Bayes	NB	Probabilistic classifier using Bayes' theorem
Logistic	LG	Logistic Regression
AdaBoostM1	AB	AdaBoost
Bagging	BG	Bagging

Table 4.5: Bootstrap Experiments: Comparative ML Algorithms

results may be achieved for independent runs of the Weka algorithms, we averaged results over 200 runs. The comparative results can be seen in Table 4.6 and demonstrate that results obtained using BS are very competitive on all of the problems. To gain a clearer insight as to which method performed best overall we carried out the non-parametric Friedman test [Friedman, 1937] which is regarded as a suitable test for the empirical comparison of the performance of different algorithms [Gerevini et al., 2005]. Results indicated that the best performing algorithm in terms of test accuracy was Bagging closely followed by LG and BS. The relatively good results of the bagging approaches are likely to have been strongly influenced by the underlying base classifier, which was the Weka REPTree algorithm. This was the best performing of the available tree based methods when combined with either the AdaBoostM1 or bagging algorithms.

4.1.6 Summary

In this section we investigated a new application of bootstrapping with GP on four binary classification tasks. The bootstrapping GP system has evolved solutions that are not only good on training data but are also not very sensitive to small variations in the training data, indicating that they are less likely to exhibit high variance.

The empirical results show that, for the problems studied, applying bootstrapping improves results on test data, and that in three out of four problems, the *performance on training data reliably indicated the*

Method	BT	Bupa	HS	WBC
BS	77.82	66.05	74.84	96.28
GP	76.79	62.96	71.52	94.72
J48	75.70	63.95	74.36	95.15
RF	73.89	65.33	65.40	96.40
SVM	74.09	64.01	71.50	96.86
MLP	77.66	64.19	74.26	95.81
NB	71.50	59.30	77.89	91.02
AB	57.41	66.48	72.68	95.54
BG	78.46	67.86	76.23	96.15
LG	77.29	66.88	77.41	96.03

Table 4.6: Performance comparison of BS and GP with ML algorithms listed in Table 4.5

performance on the test data i.e. exhibited lower variance / less evidence of over-fitting.

The results with bootstrap also compare favourably with those from several benchmark ML algorithms.

Another interesting benefit of the approach is that it produces smaller individuals than standard GP: the results demonstrate a stronger negative size fitness correlation with bootstrapping than that with standard GP. This is surprising because we do not explicitly penalise individuals for growing in size. We hypothesise that this side-effect is due to bootstrapping promoting small yet accurate programs. To support this view we presented the size distributions of best of run individuals with bootstrapping and demonstrate out a clear skew towards smaller sized individuals.

The strong negative growth fitness correlation also raises the possibility of using the deceleration of program growth associated with our application of bootstrapping as an early stopping mechanism.

4.2 Validation Sets with Static Evolutionary Curtailment

In this section we investigate a new application of a validation set, with the aim of improving the generalisation performance of a GP system

on several binary classification tasks. The approach is combined with a *static* early stopping mechanism. The system uses *Validation Pressure* combined with *Validation Elitism* to influence fitness evaluation and population structure with the aim of improving the system's ability to evolve individuals with an enhanced capacity for generalisation. This strategy facilitates the use of a validation set to reduce over-fitting while mitigating the loss of training data associated with traditional methods employing a validation set. The method is tested on several benchmark binary classification data sets.

4.2.1 Research Objective

In Chapter 3 we emphasised the desirability of testing evolved solutions against unseen data to provide an indication of their ability to solve the same problems on new instances. While this provides reassurance as to the robustness of the solution, it does not really help with the problem of over-fitting the training data

Testing solutions against unseen data provides a good indication of their generalisation ability, and can be used to detect, although not *prevent or avoid* over-fitting on the training set, as it is an *a posteriori* mechanism. To address this, several researchers have proposed using a validation set to detect over-fitting and curtail the training phase once some pre-defined stopping criteria have been met. However, establishing consistent stopping criteria has been shown to be a non-trivial task [Tuite et al., 2011].

Using a three data set approach, program fitness may be determined by the performance of the individual on training alone, while a validation set may be used to detect over-fitting. A disadvantage of this method is that having now to further subdivide the data, the number of instances available for training are reduced. This may not be an issue if the available training data is statistically representative of the problem space, as, after a time, new instances will support existing patterns and will not add further information to the model. On the other hand, if the data set is too big and noisy learning will not succeed. But, in general, reducing the amount of data available for training will negatively impact

performance.

The aim of the proposed method is to, at least partially, overcome this disadvantage by *leveraging* the validation data set during the learning process. During evolution, we evaluate each individual against the validation set and use this information to provide additional training opportunities to the system. The difference between the proposed method and standard approaches is that we envisage the role of the validation set as not only to determine when evolution should be curtailed, but also to influence the evolutionary process itself. This is achieved by employing a new selection scheme which takes into account the performance of the individual on *both* the training and validation sets.

In summary, the objective of this study is to address part of CQ1: “can novel applications of *validation* and *early stopping* strategies be employed to reduce variance (over-fitting) in evolved classifiers?”

4.2.2 Related Work

A detailed review of previous work as it relates to both *early stopping* and *validation sets* is contained in Chapter 3.2.2.

Miller and Thomson [1998] were among the first to observe the beneficial effects of elitism on evolutionary learning. Since then it has become commonplace to use elitism in GP whenever a generational strategy is employed. We extend this idea so that individuals who perform best on the validation data are also maintained in the breeding pool. In the absence of such a strategy it is quite possible that their potentially useful genes will be lost if their performance on the training data falls below the threshold required for normal elite selection.

Selection pressure is a vital component in the GP paradigm and could loosely be described as a means of directing the focus of the search operation. Usually when using tournament selection, this is achieved using tournament size: the larger the tournament, the more elitist or narrow the search strategy. Various adjustments have been employed to control aspects of the evolutionary process such as selecting the smallest of two individuals where they are tied on training fitness scores, or some other form of parsimony pressure [Luke and Panait, 2002], diversity inducing

tournaments [Azad and Ryan, 2010] or generalisation promoting mechanism [Azad and Ryan, 2011b]. As far as we are aware, this is the first time that validation elitism and validation pressure have been used to influence the evolutionary process.

4.2.3 Details of Proposed Technique

The method which we call *ValPE* employs a three data set methodology to reduce the risk of over-fitting while mitigating the potential loss of richness in the training data associated with a traditional three data set approach. This is done by leveraging the validation data set during the learning process. During evolution, each individual is evaluated against the validation set and use this information to provide additional training opportunities to the system.

The difference between the proposed method and standard approaches is that the role of the validation set is envisaged as not only to determine when evolution should be curtailed, but also to influence the evolutionary process itself. This is achieved by comparing individuals' performance on the validation data when their performance on training data is identical (*validation pressure*) and by using *validation elitism*: maintaining a percentage of individuals with elite validation performance in the population.

Using a three data set methodology, in each generation, each individual is evaluated on both the training and validation sets. When it comes to tournament selection, if two competing individuals have the same training fitness, we chose the one with the best validation fitness. Also, during evolution, *validation elitism* is used, to ensure that there is a good representation of the best so far validation scoring individuals in the breeding pool. Experiments were undertaken using various levels of validation elitism from 10%-50%. The objective is to combine these techniques with the aim of breeding individuals that are less elitist with regard to either the training or validation data, rather they are good "all-rounders". In this way, it may be possible to make more efficient use of a validation set that allows the system to be flexible regarding appropriate termination without sacrificing richness in the available training

opportunities.

For the purpose of the current experiments, in order to isolate potential effects of the new method, we have chosen to terminate evolution at the same generation number for each program run. We have elected to end each run at generation forty unless a perfect classification occurs earlier. It is possible that this approach prevents the system from delivering on its full potential as the choice of terminating generation may not be appropriate for a given classification problem while the competing baseline configuration could be said to be advantaged by implicitly reducing an inherent tendency to over fit.

4.2.4 Experiments

GP parameters used for the experiments are detailed in Table 4.7. One hundred independent runs were undertaken for each configuration, giving a total of 4500 runs (five data sets with one hundred runs each for baseline, *validation pressure* combined with eight different *validation elitism* values, that is, 0, 10, 20, 25, 30, 35, 40 and 50 percent).

For these experiments the BT, BUPA, GC, HS and WBC datasets have been selected and we have used a function set consisting of addition, subtraction, multiplication and protected division and have chosen not to use constants for our experiments. The fitness measure used for evolution was the number of classification errors of each program (zero-one loss). Final fitness values are converted to error rate and % classification accuracy for reporting and comparison purposes where appropriate.

For the remainder of this document we will refer to the baseline configuration as “Base” and the proposed method as “ValPE”.

4.2.4.1 Configurations

In this investigation, we compare the results of a standard GP implementation with those achieved on the same data sets using the proposed methodology. The outlined parameter set applies to the baseline configuration. For the proposed method, an additional parameter: validation

Parameter	Value
Strategy	Generational
Initialisation	Ramped half-and-half
Selection	Tournament
Tournament Size	5
Crossover (subtree)	90
Mutation (subtree)	10
Initial Minimum Depth	1
Initial Maximum Depth	6
Maximum Depth	17
Function Set	+ - * /
Population Size	300
Maximum Generations	40

Table 4.7: Static Early Stopping Experiments: GP Parameters

elitism is added, with values in the range 0-50%. In this case crossover and mutation rates are applied with probabilities detailed in the parameter table on the remainder of the population.

4.2.5 Results and Discussion

In this section, we outline results achieved using the Base and ValPE configurations. Statistically significant results are highlighted in bold font.

4.2.5.1 Training and Test Fitness Scores

In Table 4.8 we compare results achieved using the ValPE configuration against the baseline configuration. For training data we report average best fitness along with standard deviation and best individual fitness at the final generation. Test scores represent average fitness, standard deviation and best individual resulting from testing the hundred best of run trained individuals on each task on the corresponding test data.

We have chosen to compare using the ValPE configuration with 30% validation elitism: for most of the examples improvement was at this fraction or higher, although, as can be seen in table 4.9, 30% was the

Dataset	Training			Test			
	Method	Avg.	StdD	Best	Avg.	StdD	Best
BT	Base	82.29	2.61	84.74	75.38	1.91	81.12
	<i>ValPE</i>	81.74	2.15	84.34	75.79	1.89	81.52
BUPA	Base	81.38	2.42	86.84	70.15	3.60	77.19
	<i>ValPE</i>	80.53	2.02	86.84	73.51	2.65	80.70
GC	Base	78.60	1.73	81.98	70.90	2.11	76.57
	<i>ValPE</i>	77.37	1.77	81.98	71.64	1.68	76.26
HS	Base	83.09	1.79	88.24	76.16	2.05	79.41
	<i>ValPE</i>	81.04	1.43	84.31	76.67	1.81	81.37
WBC	Base	97.02	1.69	99.12	95.46	1.56	98.67
	<i>ValPE</i>	97.22	1.71	98.24	97.59	0.82	99.12

Table 4.8: Static Early Stopping Training & Test Accuracy. For Training, Average is the average of the best-of-run trained individuals, in all cases Best is the best individual overall. Test results in bold font are significant at the 95% confidence level with both Student’s t-test and Mann Whitney U Test

optimal value for only one of the data sets.

The baseline configuration produced the best results overall on training data: it had highest average fitness and highest average best fitness on four of the five tasks, and either outscores ValPE or shares the best score for best overall individual. However, for the more important test results the ValPE configuration outscores Base, where in *all* cases the average test performance is better with ValPE.

We expected that the proposed method would deliver improvements over the baseline configuration in cases where the data instances in the training and validation sets exhibited some dissimilarity. When we looked at the composition of the data for each problem, comparing the average and standard deviation of each attribute, we saw that these statistics were similar for both training and corresponding validation data in the cases of HS and GC, whereas there was more dissimilarity between training and validation sets apparent for the BUPA, WBC and BT data. Generally, the results seem to be consistent with this, as the greatest improvement over the standard approach can be seen in the BUPA and WBC experiments. The BT experiment does not show a similar improvement, which

Data Set	0%	10%	20%	25%	30%	35%	40%	50%
BT	75.69	75.57	75.70	76.17	75.79	75.76	75.60	76.29
BUPA	70.86	73.34	73.53	73.26	73.51	73.61	73.74	73.40
GC	70.97	71.19	70.87	71.03	71.64	71.20	70.89	70.98
HS	76.97	76.20	76.84	76.46	76.67	76.64	76.77	76.87
WBC	95.67	97.56	97.48	97.14	97.59	97.56	97.75	97.46

Table 4.9: Test Fitness VP with Elitism: average fitness on test data over 100 runs at each setting

may be because it is possible to obtain relatively good results with limited data for this task as the attribute values for positive and negative instances have good separation.

Comparing average training and average test results in table 4.8 it is clear that the variance error component (over-fitting) is smaller in all cases for the ValPE configuration. Also the standard deviation is lower in call cases.

4.2.5.2 Validation Elitism

We experimented with various levels of validation elitism ranging from 0% to 50%. Results shown in Table 4.9 were recorded when the best of run individuals were applied on the test data sets of each problem.

The application of validation elitism appears to have had a beneficial effect on most of the data sets with the exception of Haberman’s Survival where the best score was achieved without any elitism, although, in that case, the difference in accuracy between 0% and 50% elitism is minimal. The effect was only mildly beneficial for BT and GC, but had a more significant impact on BUPA and WBC. In general, the pattern suggests that higher values are more effective than lower ones. Certainly, the method appears to do no harm and has the side benefit of smaller run times due to fewer evaluations.

In a similar experiment focused on the effects of varying degrees of standard elitism on program bloat, Poli et al. [2008b] observed that higher values of elitism tended to produce smaller programs and bloated more

slowly during evolution, with elite fractions 30% and 50% behaving almost identically.

A cursory analysis of the HS underlying data instances in terms of average and standard deviation of the attribute values suggests that the partitioning of instances between the training and validation sets has resulted in the sets being very similar. One of the stated aims of our approach is to compensate for the loss of richness in test data that may result from having to partition the data into three data sets, thus reducing the number of training instances. However, as already discussed, if the partitioned data is representative (as seems to be the case here) then one would not expect the application of validation elitism to yield much in the way of improvement.

4.2.5.3 Comparison with other work

In this section, results obtained using the ValPE approach are compared with other work involving the same datasets, including non-EC approaches such as neural networks and SVMs. Results are expressed as either % classification accuracy or error rate as appropriate, to aid comparison with other work. The goal is to achieve high classification accuracy which corresponds to low error rate. Here again, for consistency, we are using the results of the 30% validation elitism configurations for comparison, although higher scores were achieved on four of the data sets with different elitism settings.

Lim et al. [2000] compared the performance of over thirty classification algorithms including decision tree algorithms, statistical algorithms and neural networks. The best performing of these on the WBC and BUPA datasets reported error rates of 0.03 and 0.28. The corresponding error rates for ValPE of 0.02 and 0.26 are better.

Looking at the Blood Transfusion problem, Gunanidhi Pradhan [2009] reported average accuracies of 78.01% using a simple artificial neural network (SANN) and 76.14% with a functional link artificial neural network (FLANN). Darwiche et al. [2010] reported classification accuracies between 74.7% and 77.3% using Support Vector Machines and RPB Kernels, and 60.0% and 72.5% with multi-layer perceptrons, depending on

Author (year)	Method	%Acc.
Cheung (2001)	BNND	61.83
Cheung (2001)	Naive Bayes	63.39
Yalçın and Yıldırım (2003)	GRNN	65.55
Cheung (2001)	C4.5	65.59
Van Gestel et al. (2002)	SVM with GP	69.70
Lee and Mangasarian (2001a)	SSVM	70.33
Yalçın and Yıldırım (2003)	MLP	73.05
Current work	ValPE	73.51
Lee and Mangasarian (2001b)	RSVM	74.86
Polat et al. (2005)	AIRS	81.00
Polat et al. (2005)	Fuzzy-AIRS	83.38

Table 4.10: Static Early Stopping Experiments: BUPA Comparative Data

which attributes were chosen. The ValPE score of 75.79% delivers a competitive result.

Polat and Güneş [2008] provides a comparison for the BUPA Liver Disorders task against a wide range of machine learning tools as shown in Table 4.10. The ValPE method ranks fourth highest in this table. In other work, Loveard and Ciesielski [2001] experimented with GP using static and population based dynamic boundaries. They recorded a best test score of 69.2%. Muni et al. [2004] reported a similar result. Recent research by Badran and Rockett [2010] applied GP with varying parameters and reported a test result of 74.86%. Overall, the ValPE result of 73.51% represents a very competitive performance.

For Habermans’s survival data set we can compare our results with those recently (2010) published by [Cetisli, 2010] as detailed in Table 4.11. In other work, Thammano and Moolwong [2007] reported accuracy of 77.17% using a fascinating classification approach inspired by human social behaviour and [Jabeen and Baig, 2010a] achieved classification accuracies of 77.63% using GP and 82.12% using Particle Swarm Optimisation(PSO). With the exception of the PSO result, the performance of the ValPE configuration with a score of 76.67% has delivered results on a par with or superior to the available published results for this data

Algorithm	% Accuracy
KNN	68.27
NN	69.93
GMM	66.01
SVM	71.24
PSVM	72.54
ANFC	72.59
ANFC	73.35
ANFC-LH	74.51
BEC	74.90
ANFC-LH	75.31
ValPE	76.67

Table 4.11: Static Early Stopping Experiments: HS Comparative Data [Cetisli, 2010]

set.

Turning our attention to the German Credit classification task, where comparative scores are expressed as error rate, the ValPE results are less convincing: Eggermont et al. [2004b] reported error rates of 0.278 using clustering GP, 0.271 using a standard GP implementation and 0.272 using C4.5. Meyer [2003] compared the performance of SVMs against various other popular classification algorithms and reported average classification error of 0.236 for SVM, 0.232 using a generalised linear model(GLM), 0.273 for a multivariate regression model and 0.284 for learning vector quantization(LVQ). In other work, Baesens et al. [2002] reported results of between 0.321 and 0.285 for several Bayesian Network Classifiers and 0.288 for C4.5. Overall, the ValPE score of 0.284 is not very competitive by comparison with the available research. Also, our baseline GP score of 0.291 compares poorly with that of Eggermont et al. [2004b]. It is possible that the small population, limited function set and termination condition we have chosen are insufficient for this particular problem.

Again, Polat and Güneş [2008] compared the performance of various algorithms on the Wisconsin Breast Cancer classification problem as shown in Table 4.12. The ValPE score of 97.59% ranks third highest of those listed. In more recent work, Winkler et al. [2006] reported test

Author (Year)	Method	%Acc.
Quinlan (1996)	C4.5	94.74
Hamilton et al. (1996)	RIAC	94.99
Nauck and Kruse (1999)	NEFCLASS	95.06
Abonyi and Szeifert (2003)	FuzzyClustering	95.57
Ster and Dobnikar (1996)	LDA	96.80
Goodman et al. (2002)	Big-LVQ	96.80
Bennet and Blue (1997)	SVM	97.20
Goodman et al. (2002)	AIRS	97.20
Pena-Reyes and Sipper (1999)	Fuzzy-GA1	97.36
Current Work	ValPE	97.59
Setiono (2000)	Neuro-Rule 2a	98.10
Polat et al. (2005)	Fuzzy-AIRS	98.51

Table 4.12: Static Early Stopping: WBC Comparative Data

accuracy of 97.07% for an enhanced GP configuration, Johnston et al. [2009] reported a test score of 95.71% using a linear regression approach to numerical simplification in Tree-Based Genetic Programming and Parrott et al. [2005] achieved classification accuracy of 95.60% with a form of multi objective GP. Overall, the ValPE result compares favourably with those achieved using other methods.

Further details on the methods listed in the referenced results tables can be found in the cited works.

4.2.6 Summary

In this section, we have highlighted both the usefulness and negative aspects of using a three data set approach. We have proposed a method in which we combine validation selection pressure and validation elitism in order to efficiently employ a three data set approach to improve generalisation ability while counteracting the potentially negative effect of partitioning the available data into three. The method has been tested on five binary classification problems and has been shown to deliver improvements in test performance over standard methods. We have compared our results with others achieved on the same data sets and the ValPE

configuration has demonstrated a competitive and sometimes superior performance on four of the five problems.

In order to isolate any effects of employing the method, we chose to terminate the runs at generation forty. While this achieved the desired result, it is not an accurate reflection of how one would tackle these problems using GP in the field. Typically, in other work using a standard approach, a greater proportion of the available data is used for training than for testing, and a larger number of generations are used. Also, by applying a “one size fits all” termination criterion to the ValPE method, it was not in a position to capitalise on its perceived natural advantage: the ability to determine the best point to stop the evolutionary process. This work is essentially a preliminary step in developing an approach that can facilitate early stopping to avoid over-fitting without sacrificing training opportunities.

A logical next step is to perform an ‘in the field’ comparison whereby the ValPE method is modified to allow the system to cease evolving once over-fitting occurs, compared with a baseline configuration where a larger proportion of the available data is used for training, and which terminates (in the absence of a perfect solution) at a generation number that is consistent with other experiments in the research on the same data sets.

4.3 Validation Sets with Dynamic Evolutionary Curtailment

This section further investigates the leveraging of a validation data set with GP to help reduce the magnitude of the variance component of generalisation error. It considers fitness on both training and validation data combined with a dynamic early stopping mechanism to improve generalisation while significantly reducing run times. In this section, we investigate several *dynamic* early stopping mechanisms and undertake a more realistic, in the field, comparison as outlined previously in Section 4.2.6. The expanded method, which we call *Chronos*, is tested on six benchmark binary classification data sets.

4.3.1 Details of Proposed Techniques

The Chronos system uses a three data set methodology, where each individual is evaluated on both the training and validation sets during evolution. For the selection process, if two competing individuals have the same training fitness, we chose the one with the best validation fitness. We also employ *validation elitism*, as in the previous section, to ensure that individuals which score well on validation are adequately represented in the breeding pool.

In addition, the system offers various selectable early stopping methods:

Chronos1

In the first of these we extend the selection approach used in the previous work, by periodically choosing the individual with the best validation fitness. In the event of a tie we select the one with the best training fitness. For this option we have used a tuneable parameter to select based on validation fitness every third generation. There is a delicate balance to be struck between getting a helpful contribution from the validation set and the risk of defeating the objective by over-fitting that data also. We expect to fine-tune this setting in future work.

Chronos1 detects increasing variance (over-fitting) by monitoring the average validation fitness for deterioration: evolution is terminated if the average validation fitness worsens a pre-set number of times without an intervening improvement (no action is taken if the value remains unchanged). We have experimented with allowing this termination mechanism to commence at various points in evolution and observed that applying the technique very early in the process may result in evolution being curtailed too soon. This is consistent with a similar observation of Tuite et al. [2011]. Results of these experiments can be seen in Table 4.15.

Chronos2

A second method which we will call Chronos2 employs the same algorithm as Chronos1, but does not have selection based on validation only. It uses the basic validation pressure and validation elitism together with

monitoring average validation performance for dis-improvement. Evolution terminates if average validation fitness dis-improves a pre-set number of times without an intervening improvement. For the current experiments that value is 3 (this value is also used in the other Chronos configurations).

Chronos3

The third approach reacts to over-fitting by terminating evolution if there is a *consistent* widening gap between average training fitness and average validation fitness, where validation fitness is worse than training fitness. In this configuration, evolution may terminate while validation fitness is still improving - provided over-fitting is getting worse. Again, for consistency across experiments, we have chosen to terminate if this gap increases 3 times without an intervening improvement.

Chronos4

The fourth method combines Chronos2 and Chronos3. Evolution terminates if *both* validation fitness dis-improves *and* variance error increases a pre-set number of times without an intervening improvement.

Chronos5

The final method employs the Chronos4 algorithm with the difference that the early stopping mechanism takes effect from the beginning of evolution, i.e. for the other Chronos configurations an early stopping “trigger” generation of 30 is employed, whereas for Chronos5 this trigger value is set to 0.

For the current study we have not fine-tuned all of the various parameters and combinations thereof: the frequency of application of *validation pressure* the level of *validation elitism*, the generation at which to activate the termination mechanism and the number of dis-improvements in average validation fitness or over-fitting to tolerate before terminating. The previous experiments suggested that a validation elitism setting of 30% works quite well, and experiments detailed later in this work show that (at least for the problem studied) variance may begin to increase

C#	VE	VP	VS	Termination Criterion	Trigger Gen.
1	Y	Y	Y	Increased Avg Validation Error	30
2	Y	Y	N	Increased Avg Validation Error	30
3	Y	Y	N	Gap between Avg. Train and Avg Val	30
4	Y	Y	N	Combine C2 and C3	30
5	Y	Y	N	Same as C4	0

Table 4.13: Chronos configurations, where C# is the Chronos version, VE is Validation Elitism, VP is Validation Pressure, VS is primary selection on Validation and trigger gen is the generation at which the early stopping mechanism is applied.

sometime after generation thirty. In future work, we may investigate the optimal configuration of these parameters

4.3.2 Experiments

In this section we provide information about the data sets used in our experiments and detail the experimental configurations and parameters used. For this extension of the previous work we have added a sixth data set, the PIMA dataset, which we use to fine-tune some parameter settings. Standard GP parameters are the same as those used previously with the exception that for the current experiments the maximum generation is 100, where previously we choose a static value of 40 generations. Specific system parameters are detailed in Table 4.14.

4.3.2.1 Experimental Configurations

Two separate sets of experiments were undertaken. The first experiments were carried out on the PIMA data set and had the objective of determining an appropriate point during evolution to activate the termination mechanism for the main experiments. Values of 0, 15, 20, 25, 30, 35 and 40 were applied, with and without the proposed selection mechanism and both base and Chronos were configured to use a three data set methodology. For these parameter tuning runs, the Chronos1 algorithm was employed. A set of identical random seeds were used for each experiment with thirty dependant runs for each configuration.

Parameter	Description	Value
vElitism	Validation Elite%	30
termStartGen	Stopping mechanism start	30
maxBadVal	Max. dis-improvements	3
valFreq	Freq. of selecting on validation for Chronos1	3

Table 4.14: Chronos Specific Parameters

The second set of experiments represent an “in the field” comparison of the proposed methods with a standard approach: a base configuration where two thirds of the available data was used for training and one third for testing, and which was allowed to run for one hundred generations unless perfect classification of training data was achieved earlier, versus the various “Chronos” configurations where the data was evenly divided into training, validation and test sets, and the different termination mechanisms were activated at generation thirty (with the exception of Chronos5).

One hundred dependant runs were undertaken for each configuration using identical random seeds. The fitness measure used for evolution was the number of classification errors of each program. Final fitness values are converted to % classification accuracy for reporting and comparison purposes. For the remainder of this document we will refer to the baseline configuration as “Base” and the proposed methods as “Chronos1”, “Chronos2”, “Chronos3”, “Chronos4” and “Chronos5”.

4.3.3 Results and Discussion

Here we outline the results achieved for both sets of experiments. The first of these were undertaken to determine an appropriate point at which to introduce the termination mechanism. We chose to carry out these experiments on a single data set: the PID data set. It is possible that the key moment in evolution may vary for different problems. However, in order to have a fair comparison of methods for experimental purposes it is best that all datasets are treated the same. Ideally we would like

to allow the termination mechanism activate as early as is practical and combine this with a robust stopping criteria that will allow the programs to run to a near optimal point.

4.3.3.1 When to Stop?

In Table 4.15 we detail training and test outcomes associated with a range of trigger generations for baseline and Chronos configurations. The baseline configuration delivers consistently higher training results, whereas the Chronos experiments report a better performance on test data. Note also that tree size is smaller for the Chronos experiments.

As one would expect, the average training performance improves as evolution continues, although the changes are quite small from one trigger setting to the next. There is slightly more fluctuation in the test results and an appropriate trigger point is not glaringly obvious. For the base configuration, there were two individuals with the best so far fitness score and they occurred at generations thirty-seven and thirty-nine, whereas the best individuals for the Chronos configuration occurred at generations thirty-four and forty-three. Average test performance peaked at a trigger point of thirty-five for both set-ups. At a trigger point of thirty, the average generation at which termination occurred was generation 38.28. Taking all of the above into consideration, we chose a trigger generation of thirty for the remainder of the experiments.

4.3.3.2 Main Experiments

In Table 4.16 we compare results achieved using the Chronos configurations against the baseline configuration. Best results on the test data are in bold font.

Surprisingly, the base configuration does not dominate the training results as much as expected but is marginally better overall when results for both average fitness and best individual over the six tasks are taken into account. On the unseen test data, Chronos delivered superior results for average fitness on four cases (including PID data set from Table 4.15), very similar results for the BT task and was outperformed by the base method only on the WBC data. With the exception of the BT

Method	Training					Test			
	Trigger	Size	Avg.	StdD	Best	Best Gen	Stop Gen	Avg.	Best
Base	0	54.00	72.77	2.44	77.65	11.33	14.00	64.83	70.98
	15	97.80	74.82	1.92	79.61	20.66	22.76	65.49	70.98
	20	123.20	75.79	1.95	80.00	26.73	29.36	65.27	70.98
	25	132.20	76.39	1.55	79.61	29.63	32.80	65.67	72.16
	30	137.66	76.43	1.86	80.39	31.93	37.53	65.63	70.98
	35	152.53	76.82	1.84	80.78	37.53	41.53	65.72	70.58
	40	166.06	77.12	1.38	81.96	40.03	46.26	65.52	70.19
Chronos1	0	57.13	71.23	2.94	77.64	12.27	14.42	66.11	70.20
	15	67.40	72.63	2.44	77.65	20.30	22.56	66.22	70.98
	20	77.13	73.37	1.84	77.25	22.97	27.6	66.47	70.19
	25	68.86	74.48	1.62	78.04	29.23	33.26	66.50	70.19
	30	115.00	74.53	1.49	78.04	33.43	38.28	66.80	73.73
	35	125.60	75.82	1.32	78.43	39.13	43.30	66.99	73.73
	40	130.13	75.75	1.09	78.04	42.8	48.38	66.45	70.98

Table 4.15: PID Training and Test Results for a range of “Trigger” Generations. Best Gen is the average generation at which the best so far individual was found. Stop Gen is the average generation at which evolution was terminated.

experiments the test results comparing Base with Chronos1 proved to be statistically significant when the standard Student’s t-test and Mann Whitney tests were applied.

4.3.4 Summary

As expected, the use of an early stopping mechanism has the benefit of producing much smaller solutions, and looking at the results we can see that for the most part this is achieved without compromising accuracy.

Comparing results of runs which used a static early stopping generation of 40 with the second set of experiments where runs using a standard approach terminated at generation 100 and the Chronos method used a dynamic early stopping mechanism, we note that training fitness is universally superior in the second set of experiments. This is not surprising for the standard experiments, as they are configured to run for 60 extra generations. Looking at test fitness the situation is not as consistent where better results are achieved with the “in the field experiments” for the BT and GC datasets. Both Base and Chronos deliver worse results for the BUPA task than previously, but between the two set-ups, the Chronos method produces significantly better results.

Looking more closely at the run data, we observed that for several of the data sets, most notably BUPA, the introduction of selection on validation every three generations has the effect of improving the validation fitness and thus inhibiting the early stopping mechanism, which means that several of the runs did not terminate until generation 100. This may explain the improved training fitness for this configuration and also why there is some over-fitting, although as we can see with the BUPA data, this over-fitting is less pronounced. In the case of the standard approach, the maximum generation is reached on every run which accounts for the improved training performance and may also explain the sometimes inferior test results, where over-fitting is likely to have occurred over time.

Overall the Chronos1 configuration delivered the best test results as shown in figure 4.14 and these results are statistically significant when compared with the Base configuration. And while one or other of the

		Training					Test		
Data set	Method	Avg.	StdD	Best	Size	Gen	Avg.	StdD	Best
BT	Base	82.15	0.81	84.34	233.88	80.04	76.01	1.62	79.12
	Chronos1	81.40	1.11	83.94	82.88	29.2	76.09	1.63	79.12
	Chronos2	83.81	1.55	87.95	250.30	76.5	75.87	1.61	79.12
	Chronos3	82.11	1.37	85.94	162.40	33.45	76.00	0.85	79.12
	Chronos4	83.32	2.69	89.22	259.42	61.72	75.71	1.79	80.32
	Chronos5	82.77	1.44	85.94	168.70	55.34	76.00	1.78	79.92
BUPA	Base	76.75	2.11	82.89	238.88	83.14	64.81	7.22	75.43
	Chronos1	81.07	1.59	84.21	85.52	30.80	68.60	7.52	77.19
	Chronos2	76.43	1.95	80.71	125.68	46.02	68.23	4.01	75.43
	Chronos3	77.92	1.96	81.58	169.12	65.00	68.40	4.05	75.41
	Chronos4	77.96	2.34	83.33	100.55	71.02	69.49	3.40	76.32
	Chronos5	78.40	1.49	81.58	159.12	54.94	70.10	3.20	76.32
GC	Base	77.18	2.26	80.18	158.30	74.30	72.04	1.51	75.08
	Chronos1	77.82	1.17	79.88	104.92	34.54	72.63	1.90	78.08
	Chronos2	80.01	2.45	85.88	240.62	82.50	70.96	1.81	75.67
	Chronos3	77.75	2.14	81.68	152.48	34.67	71.02	0.84	75.08
	Chronos4	80.11	2.54	86.17	276.64	77.15	70.92	1.80	75.68
	Chronos5	78.94	1.95	83.78	136.24	56.32	71.66	1.82	75.88
HS	Base	81.31	2.31	87.75	295.0	74.6	74.78	2.50	80.40
	Chronos1	80.72	1.42	84.29	87.44	20.06	77.10	1.85	79.41
	Chronos2	84.01	2.42	89.21	244.41	73.53	75.46	2.92	81.37
	Chronos3	81.31	2.21	81.68	152.48	32.56	75.54	3.46	81.37
	Chronos4	83.01	2.69	89.22	261.44	57.66	75.55	2.73	81.37
	Chronos5	81.02	2.61	87.25	103.46	35.25	75.33	2.47	81.37
WBC	Base	98.04	0.34	98.90	185.84	65.42	96.57	0.99	98.68
	Chronos1	98.80	0.30	99.12	77.68	23.78	95.96	1.00	97.80
	Chronos2	98.48	0.90	100	186.72	54.06	96.43	1.23	98.24
	Chronos3	98.07	1.03	100	98.30	41.13	96.81	0.62	99.11
	Chronos4	98.48	0.81	100	186.72	54.06	97.00	0.91	98.68
	Chronos5	98.50	0.74	100	109.66	48.25	96.70	1.13	98.68
PIMA	Chronos2	74.43	2.69	81.68	165.48	46.41	65.98	2.21	72.15
	Chronos3	75.11	2.56	81.17	190.02	51.14	66.38	2.59	74.50
	Chronos4	75.71	2.74	81.56	228.86	65.71	65.74	2.75	74.50
	Chronos5	74.97	2.37	81.57	136.16	51.44	66.59	3.01	75.69

Table 4.16: Dynamic Early Stopping Experiments: Training & Test Results. Size is average tree size in nodes. Gen is the average generation at which the best so far individual was found. Best test results are in bold font. Best average test result for PIMA was achieved with Chronos1 as shown in table 4.15

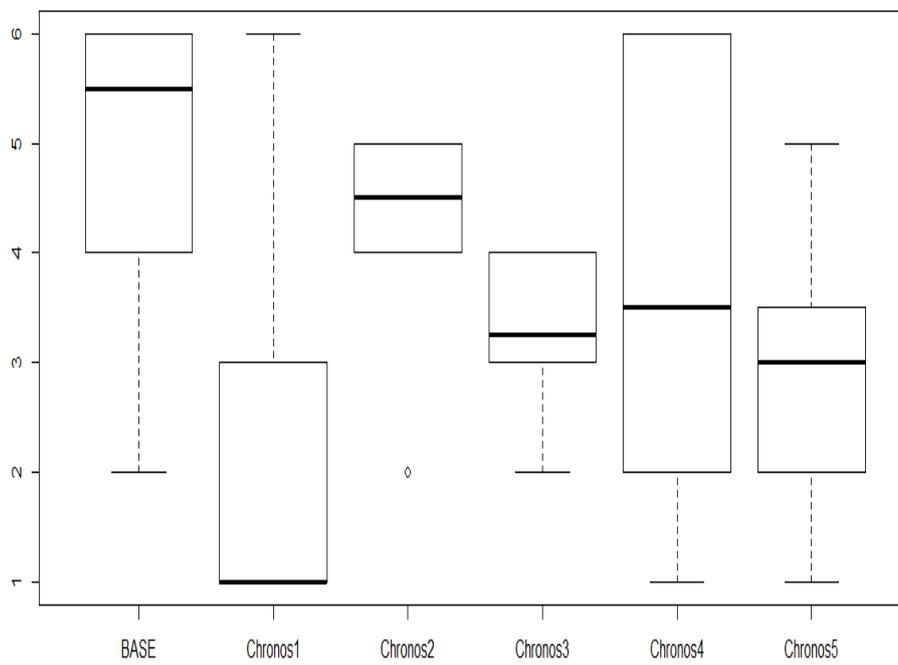


Figure 4.14: Overall ranking of algorithms across problems with Friedman test, where 1 ranks highest.

methods may have produced superior results for a particular dataset, there is not a great deal of difference between the results of the various Chronos methods when they are evaluated across all of the problems.

4.3.5 Research Outcomes

In this chapter we have used Bootstrap standard error as a measure of sensitivity of evolving classifiers to the training data. We minimised both the standard error and training classification error and find that this approach reduces variance, thus improving generalisation to unseen data. Rather surprisingly, we discovered that this method produces trees that are much smaller in size than those evolved using standard GP. We find that when using bootstrap, there is a strong negative correlation between fitness and sizes of the individual. This surprising discovery shows that using bootstrap can consistently produce small and reliable programs which is a desirable quality in any ML algorithm.

As this study deals solely with Binary Classification problems it is not possible to draw general conclusions regarding the success of the bootstrapping technique in terms of generalization and bloat in GP beyond this scope.

Also in this chapter, we have investigated a new application of a validation set together with an early stopping strategy. It could be argued that the proposed system would have merit if it performed as well as the standard configuration in terms of accuracy, as the resulting time savings would be well worth the extra effort in implementation. Also, as evolution is terminated earlier tree size is consequently much smaller. This, when combined with the simple arithmetic function set may mean reduced complexity in the evolved solutions, a feature which may be useful in arenas where this is considered important, such as the Medical domain. In this context, the superior average test results obtained using the Chronos system are encouraging.

Overall, the research has shown that bootstrapping, and validation sets combined with early stopping, have a role to play in reducing the variance component of generalisation error for the problems studied.

Chapter 5

Search Bias

In this chapter we focus on *search bias* which refers to the factors that control the way in which a learning system transforms one hypothesis into another. In terms of GP this bias is represented by the crossover and mutation operators. For binary classification problems, we also consider class boundary determination as an important factor in search bias, as different decision thresholds may have a strong impact on the selection and survivability of individual hypotheses in a population.

Here we address the second core question: CQ.2 “can new *individualised* approaches to selection of *boundary values* and *genetic operator probabilities* lead to the evolution of more generalisable solutions?”.

5.1 Introduction

Bias is inevitable and even desirable in machine learning. As Mitchell [1980] stated “only if [the learner] has other sources of information, or biases for choosing one generalization over the other, can it non-arbitrarily classify instances beyond those in the training set”. Bias may be introduced through various aspects of the chosen paradigm including available representations, search strategy and parameter settings. Depending on these, combinations thereof, and possibly other system components, a given system may be biased such that it may perform better on one type of problem over another [Loveard, 2003; Provost and Buchanan, 1995]. Indeed, the *no free lunch theorem* [Wolpert and Macready, 1995, 1997] says that any algorithm will on average perform no better than random

when it is evaluated over *all problems*.

In tree based GP, the types of solution which may be evolved are limited to those which can be expressed with that data structure using the elements from a supplied primitive set (language bias) together with the application of settings such as minimum and maximum allowable tree depths, initialisation algorithm or selection strategy (selection bias), each of which may influence the sizes and shapes of solution possible.

These types of bias could be regarded as a sort of *bias by design* in the sense that we intentionally choose a search paradigm and configure algorithm parameters in order to achieve some desired outcome, or to tackle a particular type of problem. However, it may happen that a certain design choice will introduce an undesirable or unwanted bias into the system, and in this chapter we argue that the common GP practice of employing a decision threshold (also known as a class boundary) of zero for binary classification tasks is such a design choice and constitutes an *inappropriate bias*. Traditional methods commonly use a population based, zero valued class boundary, where, for every individual in a population, instances for which the numerical program semantics are less than or equal to zero are classified as being in one of two classes, and those with values greater than zero are classified as being members of the other class. In this chapter, we propose an alternative choice which is designed to be *appropriately biased* such that search is more focused and constructive than is the case when using the traditional choice.

We initially propose a simple technique which we call *Evolved Class Boundaries* (ECB) in Section 5.4. ECB makes it easier for the GP individuals to determine an appropriate class boundary. This is similar in operation to the Linear Scaling operation proposed by Keijzer [2004] in that individuals are not penalized for not having access to ideal constants (either Ephemeral Random Constants or synthesized ones). It operates by automatically scaling the class boundary into the numerical range of the individual being tested, which permits the system to *concentrate simply on distinguishing among the classes* rather than having to evolve constants at the same time.

The technique which can be used with Genetic Programming (GP)

reduces implicit bias in binary classification tasks. Arbitrarily chosen class boundaries, such as the traditional zero boundary, can introduce bias, but if individuals can choose their *own* boundaries, tailored to their function set, then their outputs are automatically scaled into a suitable range. These boundaries evolve over time as the individuals adapt to the data.

Our initial system calculates the *Evolved Class Boundary* (ECB) for each individual in every generation, with the twin aims of reducing training times and improving test fitness. The method is tested on three benchmark binary classification data sets from the medical domain.

In Section 5.5 we develop and extend the ECB technique and investigate seven other possible methods which could be used to determine a semi-optimal decision threshold. Also in that section, we address a second bias introduced in traditional methods: the constraint that positive data instances are ranked higher than negative data instances. In traditional methods, this means that program semantics for positive or minority instances should have a value > 0 whereas those for negative or majority instances should have values ≤ 0 . Using these criteria, a classifier which perfectly separates instances of both classes may achieve the worst possible fitness score and be discarded from the population, if, for example, the program semantics are universally > 0 and all negative instances have semantic values which rank higher than those of all positive instances.

Finally, in Section 5.6 we investigate a scheme of self-adaptive, individualised genetic operators. Static population based values for parameters such as crossover, mutation and tournament size are very widely used in GP research, even though this is somewhat contrary to the practice in other EC disciplines and to our intuition and knowledge of how the search process operates: Angeline [1995] pointed out that as evolutionary algorithms manipulate a population of potential solutions in parallel we should not assume that global population based settings are the most effective; Eiben et al. [1999] argued convincingly that the use of static parameters exclusively may lead to *inferior algorithm performance* and suggested that adaptive methods were likely to be more effective.

5.2 Research Objectives

In this chapter we address the second core question: “can new *individualised* approaches to selection of *boundary values* and *genetic operator probabilities* lead to the evolution of more generalisable solutions?”.

Several individualised approaches to boundary selection are evaluated in sections 5.4 and 5.5. We compare the results achieved using these techniques with other results from GP and other ML research on the same datasets.

With regard to genetic operator probabilities, we investigate a scheme of self adaptive, individualised genetic operators combined with adaptive tournament size designed to provide balanced, self-adaptive exploration and exploitation. We test this scheme on several benchmark binary classification problems and compare results obtained with the proposed techniques with both a tuned GP configuration and a feedback adaptive GP implementation.

5.3 Previous Work on Boundary Determination

A key step in the classification process is the determination of a class boundary. For binary classification tasks in GP this has traditionally been achieved in one of three ways:

Usually [Song et al., 2001; Tackett, 1993], the boundary is set at zero, in which case the sign of an individual’s output on a given training instance is used to determine the class label. Alternatively, the output of the program is itself a binary value which translates to a class label. For example, Bojarczuk et al. [1999] used logic operators to generate an if-then-else classification rule for chest pain diagnosis.

Another approach is to pass the output of individuals to a separate component whose function is the determination of the class label. Estebanez et al. [2007] used a Linear Perceptron, where GP was used to evolve useful combinations of features and the classification step was done with the Perceptron. Lam and Ciesielski [2004] applied K-Means clustering to

the program outputs, Smart and Zhang [2005] proposed Gaussian distribution and probability models applied to program outputs. These last two approaches were applicable to both binary and multi-class classification tasks.

An early novel approach entitled “ROC Dominance” was suggested by Hunter [2002] in which classifier performance was optimised by exercising the system through a sorted range of decision thresholds to find the optimal balance of sensitivity and specificity.

Several approaches have been recommended for multi-class problems using GP, such as Static Range Selection [Zhang and Ciesielski, 1999] and Dynamic Range Selection (DRS) [Loveard and Ciesielski, 2001]. The DRS algorithm used a subset of the training data set to devise population based class boundaries. Zhang and Smart [2004] proposed two dynamic methods: Slotted Dynamic Range Selection (SDRS) and Centred Dynamic Range Selection (CDRS). The SDRS method employed “slots” in the range -25 to +25 and assigned the output of each program weighted by fitness to a particular slot, and each slot was assigned to a class. With CDRS, a centre for each class was calculated and the midpoints between centres formed the class boundary for the population as a whole. A similar approach using a decreasing weighted function was proposed by Li et al. [2008]. Each of the above methods were applied to image classification problems. Overall, it was found that Static Range Selection was effective with “easier” images but did not scale successfully. The dynamic methods performed well on more complex images but suffered from long training times and tended to produce large programs.

Several researchers have tackled the problem of multi-class classification in GP by decomposing the problem into multiple binary classification problems [Song et al., 2001] [Zhou et al., 2002]. The development of more effective binary classification algorithms could contribute towards achieving better solutions to multi class classification tasks.

5.4 Exploring Boundaries

In this section, we detail our initial, simple approach to determining a suitable decision threshold, designed to avoid bias which may otherwise be introduced into the system through the use of a standard zero valued, population based threshold.

5.4.1 Details of Proposed Technique

There are some problems with the existing techniques used to determine class boundaries: the determination of good static boundaries may require hand crafted input by human experts [Loveard and Ciesielski, 2001], and the use of dynamic population based boundaries can result in long training times as the population strains to learn appropriate boundary values [Li et al., 2008; Smart and Zhang, 2003].

$$\text{boundary}P = \frac{\frac{\sum_i PO_i(C_1)}{mC_1} + \frac{\sum_j PO_j(C_2)}{mC_2}}{2} \quad (5.1)$$

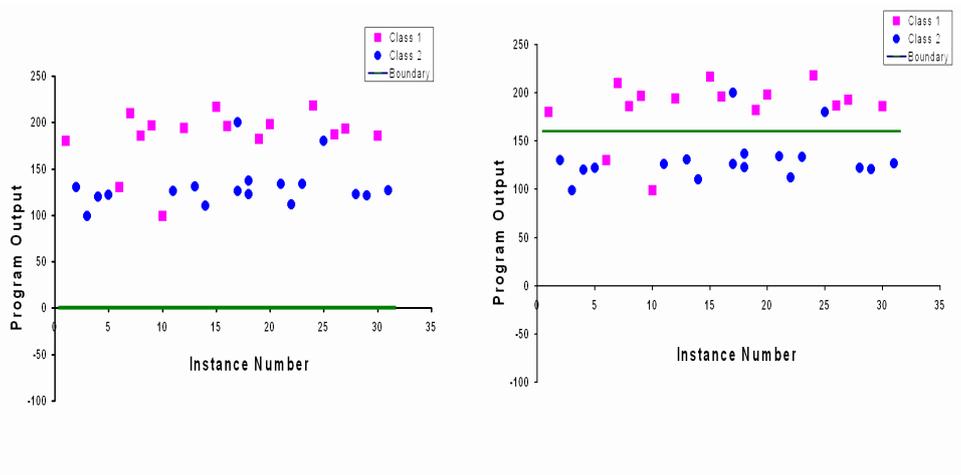
Where P represents an individual program, $PO_i(C_1)$ and $PO_j(C_2)$ are the outputs of the program for instances of Class 1 and Class 2, and mC_1 and mC_2 are the number of instances of Class 1 and Class 2 respectively. The final Evolved Class Boundary for each individual is the boundary that is applied to validation and test instances.

With binary tasks, given feature vectors of positive real valued attributes, and a static class boundary of zero, individuals will take some time to move in the direction of the zero boundary. Similarly, given a population of very diverse individuals, a dynamic boundary that is designed for the population as a whole is likely to be challenging for individuals who differ from some mean population value. This is particularly likely to occur in the initial generations, and thus potentially useful genetic material may be discarded.

Rather than attempting to calculate boundaries that suit the entire population, our approach instead allows each individual to learn a class boundary that is natural for its genotype. We suggest that if the individual is permitted to choose a boundary that arises naturally out of

the range of its outputs, then promising candidates may achieve good initial fitness and are less likely to be lost. In the proposed method, at each generation during training, the individual Evolved Class Boundary (ECB) is simply determined by calculating the mean of the individuals output for each class on the training data, and getting the midpoint of these two means.

Figure 5.1 illustrates an extreme example of how using the standard decision threshold of zero can introduce an unwanted bias into the system, and how the use of an individualised decision threshold can eliminate this particular bias.



(a) Zero Threshold

(b) Individual Threshold

Figure 5.1: Illustration of undesirable/inappropriate bias introduced when using a standard approach: zero boundary

5.4.2 Experiments

Three benchmark data sets from the medical domain have been used for experiments: The Wisconsin Breast Cancer data set, the Pima Indians Diabetes data set and the BUPA Liver Disorders data set.

5.4.2.1 Function Sets

We have used a function set consisting of addition, subtraction, multiplication and protected division. It could be argued that the use of

Table 5.1: GP Parameters for initial Class Boundary experiments.

Parameter	Value
Strategy	Generational
Initialisation	Ramped half-and-half
Selection	Tournament
Tournament Size	4
Crossover	70
Mutation	10
Elitism%	20
Initial Min Depth	1
Max Depth	17
Function Set	+ - * /
Population	500
Max Gen	50

Boolean operators would eliminate boundary bias and avoid the problem of developing dynamic boundaries. However, this approach may not scale well to multi-class tasks, and the application of strongly typed GP with mixed types may result in unnecessary complexity. More importantly, many classification problems have numeric rather than logical data, so it would seem more appropriate to use arithmetic operators in those cases. We have chosen not to use constants for our experiments, as the system should be capable of synthesising any values required using the function set and the range of values in the existing terminal sets. Terminals are drawn from the available numerical attributes.

5.4.2.2 Experimental Set-up and GP Parameters

GP parameters used for the experiments are detailed in Table 5.1. Each binary classification task was undertaken first using a baseline configuration, where the class boundary was set at zero, and the sign of the program output was used as the class label. Next, the experiments were repeated using the ECB configuration. Finally, experiments were run for the multi-class, population based “Centered Dynamic Class Boundary Determination” (CDCBD) method proposed by Zhang and Smart [2004].

Table 5.2: ECB Experimental Configurations.

Name	Description
Static	Static Class Boundary (always set to zero)
CDCBD	Centered Dynamic Class Boundary[Zhang and Smart, 2004]
ECB	Evolved Class Boundary

For each experiment, the data was divided into three sets: a training set, a validation set and a test set. The training set was used to train the GP Individuals on the particular experiment (PID,WBC or BUPA), while the validation set was employed to detect over fitting, and to select suitable candidates for testing; during training, each individual was evaluated against the validation set and a separate validation score was maintained (with no influence on training fitness). The individual with the highest validation score on each run was selected for evaluation on the corresponding test set. A useful application of a validation set is to curtail the training phase once some pre-defined validation criteria has been met, such as a deterioration or lack of improvement in validation fitness. On this occasion, we allowed the experiments to run to completion in order to obtain boundary behaviour data.

Fifty runs were undertaken for each configuration. For the static and ECB experiments, the runs were independent, whereas the CDCBD experiments were configured using the same random seeds as for the corresponding ECB runs. Overall, a total of 450 runs were completed. The fitness measure used for evolution was the number of classification errors of each program. Final fitness values are converted to error rate and % classification accuracy for reporting and comparison purposes. The GP framework used was the Open Beagle Evolutionary Framework[Gagné and Parizeau, 2002]. For the remainder of this document the naming conventions outlined in Table 5.2 apply:

5.4.2.3 Results and Discussion

Results obtained using the baseline Static configuration are compared with those achieved using the CDCBD and ECB configurations and, outcomes for each of training, validation and test fitness, program size, and nodes processed are compared. In the case of training, the results displayed are the averages and standard deviation of the final generation of each run, whereas the values for validation and test results represent the average and standard deviation of the fifty selected validation individuals for each task when applied to the validation and test data respectively. We calculated the average number of nodes processed per run using the number of individuals processed and the average tree size for each generation, totalled over each run and then averaged over the fifty runs of each experiment. For each set of experiments, the best result in each category is displayed using bold text.

Table 5.3: Training Validation & Test Fitness: Static, CDCBD and ECB Configurations. Best results are in bold font.

Dataset	Method	Training			Validation			Test			
		Average Fitness	StdDev	Best Fitness	StdDev	Average Fitness	StdDev	Best In-div.	Average Fitness	StdDev	Best Individ
WBC	Static	94.95	0.74	98.44	0.41	96.97	0.73	98.28	95.58	1.10	97.41
	CDCBD	91.6	8.75	97.52	0.36	97.50	0.66	98.66	97.42	1.41	99.55
	ECB	98.33	0.51	98.89	0.2	96.75	0.28	97.41	96.42	0.54	97.41
PID	Static	72.27	2.27	77.03	1.73	70.91	1.23	74.5	67.52	3.07	74.9
	CDCBD	67.28	6.58	75.55	2.10	71.40	1.80	74.90	69.35	3.95	78.43
	ECB	78.43	1.34	80.55	0.68	76.26	0.61	78.04	77.33	1.33	80.78
BUPA	Static	76.04	3.03	81.39	2.84	72.60	2.49	77.19	71.95	4.15	78.94
	CDCBD	74.68	4.41	81.82	1.70	74.79	1.70	79.70	69.53	4.29	77.19
	ECB	77.84	2.08	82.14	1.8	77.19	2.04	79.82	73.31	2.32	79.82

Training, Validation and Test Fitness Scores

The results for the WBC experiments seen in Table 5.3 show that the ECB method produced better training results for both best and average fitness whereas the CDCBD method scored highest on validation and test fitness. Overall, both dynamic methods outperformed the static method. For the PID experiments, the ECB method consistently outscored the Static and CDCBD methods on each of the training, validation and test categories, and for average test fitness the result for the ECB configuration was almost 10% better than that of the Static configuration. Similarly, looking at the BUPA experiments, we see that the ECB approach produced better scores on each of training, validation and test fitnesses and the static method outperformed the CDCBD approach on test fitness.

The results in Table 5.3 demonstrate that the ECB technique delivered better results overall on the chosen tasks and both dynamic methods demonstrated superior performance over the static method, with the CDCBD approach doing best on the WBC problem and worst on the BUPA task. As the main objective when attempting to evolve classifiers is to generate ones that generalise well, one could argue that the most important statistics are average and best overall test fitness. With respect to both of these, the ECB method has proven superior on two of the three experiments undertaken.

Looking at the results in terms of the bias and variance components of generalisation error, we can observe that the proposed approach delivered improvement in both aspects, and also had the lowest standard deviation in all categories. Interestingly, the CDCBD method exhibited no variance error (over-fitting) in two of the three cases, although its overall results were inferior to those achieved with the ECB method.

Average Tree Size

Results for average tree size shown in Table 5.4 represent the average tree size at generation 50 averaged over all runs for each task in the case of training. The tree sizes reported for validation/test are averages of the tree sizes of the fifty validation individuals which were selected for

testing for each experiment. The results indicate that both the ECB and CDCBD methods produce significantly smaller trees overall, and between the two of these approaches, ECB produced smaller trees for two of the three tasks. Also, in each case, the average tree size of the validation/test individuals was smaller than the average of the population from which they were selected. This suggests that for this type of problem, smaller trees may generalise better. This is consistent with previous work of Nordin and Banzhaf [1995] and the general principle of Occam’s razor.

Table 5.4: Average Tree Size: Static, CDCBD and ECB Configurations.

Data Set	Method	Train	SdDev	Test	SdDev
WBC	Static	136.3	81.44	112.92	87.70
	CDCBD	89.04	31.36	49.72	24.74
	ECB	88.2	32.88	29.96	19.3
PID	Static	241.25	190.67	176.06	213.8
	CDCBD	77.64	25.28	46.00	29.72
	ECB	77.62	30.7	37.44	28.28
BUPA	Static	480.76	666.34	346.08	625.4
	CDCBD	113.04	25.80	94.24	31.29
	ECB	163.98	146.37	106.2	147

A small tree size is a desirable outcome, particularly when an uncomplicated function set is used, as the evolved results may be easier to interpret. Comprehensibility of the evolved solutions is important in certain problem domains, notably the medical domain, where the reasons for a particular classification may be as important as the classification itself. Similarly, in areas such as texture classification, it is useful to learn if the system has discovered discriminating features not previously understood.

Average Nodes Processed per GP Run

Table 5.5 shows the average number of nodes processed per GP run for each method. The results indicate that for the current experiments, significantly fewer nodes were processed during runs where the ECB and CDCBD methods were used. Comparing the ECB approach with the Static method, there were 63%, 55% and 60% fewer nodes processed for the WBC PID and BUPA tasks respectively when the technique was employed. The CDCBD technique used significantly fewer nodes for the BUPA task and generated results similar to the ECB approach for the other tasks. In GP, the most significant proportion of the computational cost is incurred in evaluating individual fitness. It follows that smaller individuals with fewer nodes to process should deliver shorter run times and use less memory overall.

Table 5.5: Average Nodes Processed: Static, ECB and CDCBD Configurations.

Data Set	Method	Nodes (1000s)
WBC	Static	2533
	CDCBD	1034
	ECB	925
PID	Static	2555
	CDCBD	1264
	ECB	1170
BUPA	Static	8569
	CDCBD	1168
	ECB	3412

As noted earlier, one of the perceived drawbacks to the application of GP to classification problems is the belief that it has a requirement for long training times when compared with other classification methods [Loveard and Ciesielski, 2001]. We suggest that the results for the current

experiments demonstrate that use of the ECB technique can deliver dramatically reduced run times for this type of classification problem with virtually no trade off, as the boundary calculation is integrated with the normal evaluation process and does not involve any additional evaluations, which may be required when applying population based boundaries.

Statistical Tests

In order to ascertain whether the superior test results offered by the ECB method are significant, we carried out some simple statistical tests. Firstly, in order to check that the samples were normally distributed, we computed the z-score for the minimum and maximum values of each sample for each experimental task. Then, using a standard statistical hypothesis test [Triola, 1995] for large (> 30) independent samples, and choosing a significance level of 0.05, we calculated sample statistics for the differences in means of test fitness obtained for both the Static and ECB methods for each data set. The rationale in doing this is to establish whether it is likely that both samples are from the same or different populations, i.e. if the ECB method has no significant effect then the means of both populations are equal (null hypothesis).

Test statistics of 4.86, 20.73 and 2.02 were obtained. The corresponding P-Values were 0.002, 0.002 and 0.0434. The P-Value represents the probability of getting a value of the sample test statistic that is at least as extreme as the one obtained from the sample data assuming the null hypothesis to be true. Using the significance level of 0.05, the ECB results can conventionally be described as statistically significant for the WBC, PID and BUPA experiments.

In comparing the performance of the CDCBD method against the ECB approach we used the Student t test for dependant samples. Test statistics of 11.71, 1.61 and -1 were obtained from the PID, BUPA and WBC data sets respectively. Using a significance level of 0.05 the critical values of t are -1.960 and 1.960. Thus, the difference in performance can be described as significant for the PID data, and not significant for the WBC and BUPA data.

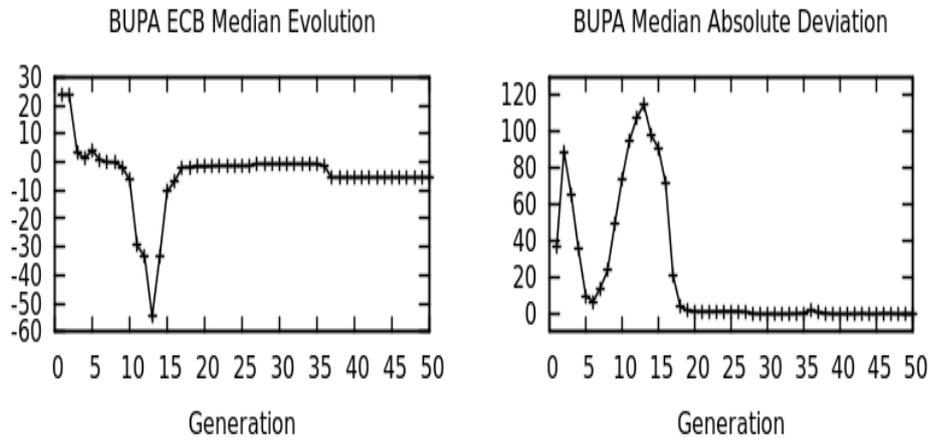


Figure 5.2: Evolution of ECB Median and Median Absolute Deviation values for typical BUPA runs, Median and MAD on left axis.

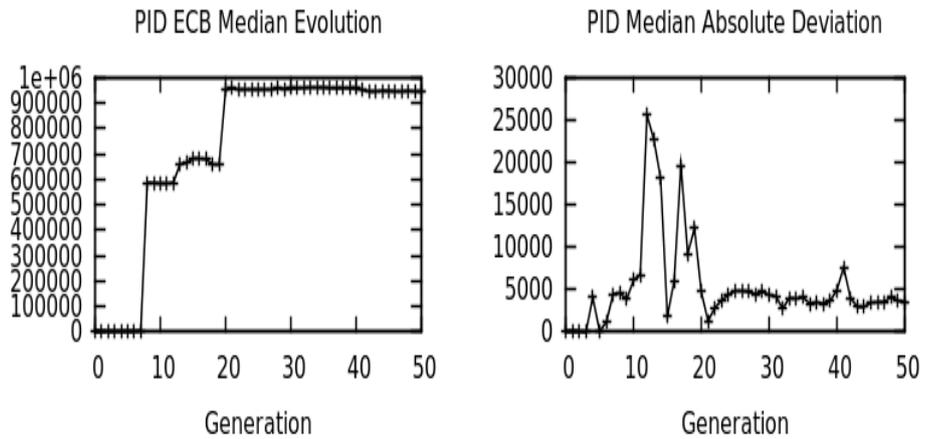


Figure 5.3: Evolution of ECB Median and Median Absolute Deviation values for typical PID runs, Median and MAD on left axis

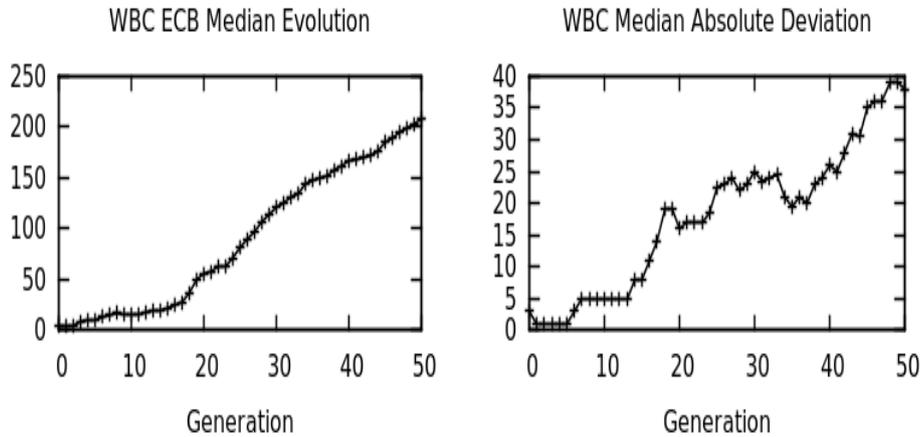


Figure 5.4: Evolution of ECB Median and Median Absolute Deviation values for typical WBC runs, Median and MAD on left axis

ECB Behaviour

ECB works because individuals dictate their own boundary values. This section examines how the boundaries change over time. Due to the extremely wide range of the ECB values and the presence of significant numbers of valid outliers, we have chosen to report the median and median absolute deviation (MAD) for each generation rather than the average and standard deviation. The median is a robust measure of central tendency and the median absolute deviation is a robust measure of statistical dispersion. We consider that these robust measures present a more meaningful representation of the ECB data [Hoaglin et al., 1983]. ECB values have been rounded to zero decimal places for reporting purposes only. Figures 5.2 to 5.4 show the behaviour of the ECB medians and median absolute deviation for a typical run of each problem. Median and Median absolute deviation values are scaled on the left axis in Figures 5.2 to 5.4 respectively.

Observing the behaviour of the ECB across the three problems, we see that while the behaviour was noticeably different between the PID, BUPA and WBC problems, it was consistent across runs for each task. This would suggest that the system has exploited the boundaries in a problem specific way.

For the PID experiments, the boundaries typically ranged from very large (up to $\pm 10^{38}$) negative values to very large positive values early in the evolutionary process, and evolved towards large and very large positive values by the final generation.

Early evolution of the BUPA boundaries was similar to the PID boundary data where the individual boundaries ranged between very large negative and positive values. However as the process progressed, they moved closer to zero and at generation 50, generally ranged from small negative to positive values with a median close to zero.

The WBC experiments showed a different behaviour. In the first few generations, the underlying boundary data typically ranged from small negative to positive values with a small number of large positive outliers. At the final generation, these had generally evolved to being almost exclusively positive, in the range 0-500.

When comparing the Evolved Class Boundaries of the top and bottom 20% of the population at the final generation, it was clear that the boundary values of the fitter individuals ranged much closer together than those of their less fit counterparts. For example, in the case of the PID experiments the boundaries of the fitter individuals tended to be several degrees of magnitude smaller on average than those of the unfit portion of the population. This is a rather nice finding which may suggest that application of ECB could lead to more constructive crossover operations.

5.4.2.4 Comparison with other work

In this section, results obtained using the ECB approach are compared with other work using the same datasets, including non-EC approaches such as neural networks and SVMs. Results are expressed as either % classification accuracy or error rate as appropriate, to aid comparison with other work.

In the previously mentioned work of Lim et al. [2000] the best performing algorithms on the WBC, PID and BUPA datasets reported error rates of 0.03, 0.22 and 0.28 respectively. The corresponding best results (rounded) for ECB of 0.03, 0.23 and 0.27 on those problems are extremely

Table 5.6: WBC Comparative Data from [Polat and Güneş, 2008]

Author (Year)	Method	%Acc.
Quinlan (1996)	C4.5	94.74
Hamilton et al. (1996)	RIAC	94.99
Nauck and Kruse (1999)	NEFCLASS	95.06
Current Work	Static	95.58
Abonyi and Szeifert (2003)	FuzzyClustering	95.57
Current Work	ECB	96.42
Ster and Dobnikar (1996)	LDA	96.80
Goodman et al. (2002)	Big-LVQ	96.80
Bennet and Blue (1997)	SVM	97.20
Goodman et al. (2002)	AIRS	97.20
Pena-Reyes and Sipper (1999)	Fuzzy-GA1	97.36
Current Work	CDCBD	97.42
Setiono (2000)	Neuro-Rule 2a	98.10
Polat et al. (2005)	Fuzzy-AIRS	98.51

competitive. However, in some cases, superior results to those of Lim et al have been reported elsewhere in the literature as detailed in Tables 5.6 to 5.8.

Looking firstly at comparative data for the Wisconsin Breast Cancer classification problem, we compare with [Polat and Güneş, 2008] who detailed the performance of various algorithms on the task as shown in table 5.6. At 96.42% the ECB scores slightly below average when compared with the other results reported by Polat et al.

For the PID classification task we compare results with those reported in Statlog as shown in Table 7[Michie et al., 1994]. Here, the rank associated with each algorithm is based on test classification error. Using this criteria, the ECB configuration ranks fourth on the StatLog scale with an error rate of 0.227. In more recent work by Eggermont et al. [2004b] several GP variants were tested. The best of these had an error rate of 0.242. Tsakonas [2006] tested four GP algorithms and reported an error rate of 0.2198 for GP with Fuzzy Rule Based Systems. Considering these results, the performance offered by the ECB technique could

Table 5.7: Statlog[Michie et al., 1994] Training and Test error rates on PID Domain.

Algorithm	Train	Test
Logdisc	0.219	0.223
Discrim	0.220	0.225
DIPOL92	0.220	0.224
ECB	0.216	0.227
SMART	0.177	0.232
RBF	0.218	0.243
ITrule	0.223	0.245
Backprop	0.198	0.248
Cal5	0.232	0.250
CART	0.227	0.255
CASTLE	0.260	0.258
NaiveBay	0.239	0.262
Quadisc	0.237	0.262
C4.5	0.131	0.270
IndCART	0.079	0.271
Baytree	0.008	0.271
LVQ	0.101	0.272
Kohonen	0.134	0.273
Kohonen	0.134	0.273
AC	0.00	0.276
CN2	0.010	0.289
NewID	0.00	0.289
ALLOC80	0.288	0.301
CDCBD	0.327	0.306
k-NN	0.000	0.324
Static	0.277	0.325
Default	0.350	0.350

Table 5.8: BUPA Comparative Data from Polat and Güneş [2008]

Author (year)	Method	%Acc.
Polat et al. (2005)	Fuzzy-AIRS	83.38
Polat et al. (2005)	AIRS	81.00
Lee and Mangasarian (2001b)	RSVM	74.86
Current work	ECB	73.31
Yalçın and Yıldırım (2003)	MLP	73.05
Current work	Static	71.95
Lee and Mangasarian (2001a)	SSVM	70.33
Van Gestel et al. (2002)	SVM with GP	69.70
Current work	CDCBD	69.53
Cheung (2001)	C4.5	65.59
Yalçın and Yıldırım (2003)	GRNN	65.55
Cheung (2001)	Naive Bayes	63.39
Cheung (2001)	BNND	61.83

be described as very competitive.

Referring once again to the work of Polat and Güneş [2008] in Table 5.8 we make a comparison for the BUPA Liver Disorders task. Here, the ECB configuration resulted in a classification accuracy of 73.31% which scores fourth highest of the methods listed and is only outscored by RSVM, AIRS and Fuzzy-AIRS. The Static method also performs respectably with accuracy of 71.95%. In other work, Loveard and Ciesielski [2001] experimented with GP using static and population based dynamic boundaries. They recorded a best test score of 69.2%. Muni et al. [2004] reported a similar result. Recent research by Badran and Rockett [2010] applied GP with varying parameters and reported a test result of 74.86%.

5.4.2.5 Summary

In this section we have presented a technique that can be used with GP for to improve performance on binary classification tasks. The suggested approach replaces an *inappropriate search bias* which is introduced through application of the widely used zero boundary value, with what we

regard as a more appropriate search bias where each individual chooses an appropriate boundary. We believe that we have shown that the proposed method, ECB, can improve training, validation and test results, reduce run times and produce smaller trees. Although the method is extremely simple, it has proven to be surprisingly effective. Most importantly, the technique offered good results on test data, and we have established that these results are likely to be statistically significant. In comparison with other work on the same problems, the proposed method has delivered competitive and sometimes superior results.

Overall, we believe that the results and comparisons show that the Evolved Class Boundary method has delivered a competitive performance for the Wisconsin Breast Cancer data, with competitive and often superior performances for both the Pima Indians Diabetes and BUPA Liver Disorders classification tasks. In terms of bias and variance, the technique resulted in reductions in both of these compared with the baseline configuration, in two of the three problems studied. Results of additional experiments in which ECB was used in evolving classifiers for texture images can be seen in Appendix D. Those results support the efficacy of individualised boundaries as described here.

5.5 Optimising Boundaries

Building on the previous work which proposed the simple ECB method [Fitzgerald and Ryan, 2011a], this section explores a range of possible class boundary determination techniques that might be used to improve performance of Genetic Programming (GP) on binary classification tasks, including an updated and optimised version of ECB. Each of these techniques involve selecting an individualised boundary threshold in order to reduce implicit bias that may be introduced through employing an arbitrarily chosen threshold, such as the commonly used zero boundary.

For this extended study, we have added several additional datasets: the Blood Transfusion Dataset (BT), Haberman’s Survival Dataset (HS) and the Ionosphere Dataset (ION).

5.5.1 Boundary determination techniques

We investigate eight different methods of individual class boundary determination, each of which could be used with GP for binary classification tasks where the outputs of the individual program are numerical. We have devised each of these methods based on some intuition that one or other of them might prove suitable.

5.5.1.1 ECB

This method involves calculating an *evolved class boundary*(ECB) for each individual. The boundary is determined by calculating the mean of the individuals output for each class on the training data, and getting the midpoint of these two means, as previously outlined in section 5.4.

5.5.1.2 AUCN

As previously described in 2.2.1.5 the AUC of a classifier is equivalent to the probability that the classifier will rank a randomly chosen positive instance higher than a randomly chosen negative instance [Fawcett, 2006] and is equivalent to the value calculated using the Mann-Whitney-Wilcoxon test. Influenced by these ideas, we investigate a boundary

Table 5.9: GP Parameters for Boundary Determination Experiments.

Parameter	Value
Strategy	Steady State
Initialisation	Ramped half-and-half
Selection	Tournament
Tournament Size	5
Crossover	80
Mutation	20
Initial Min Depth	1
Initial Max Depth	6
Max Depth	17
Function Set	+ - * /
ERC	-5 to +5
Population	500
Max Gen	60

determination method based on the notion that output values for the positive class may tend to be higher than the output values of the negative class. Using method AUCN, the outputs of each individual are sorted in ascending order and the boundary is positioned midway between the value which occurs at the array index equal to the number of “negative” training instances and the next array value, i.e. where it might be if all instances of the positive class had generated higher outputs than all instances of the negative class.

5.5.1.3 AUCP

This is similar to the previous approach but instead we assume that the instances of the positive class will tend to have *lower* values than instances of the negative class.

5.5.1.4 ECBS

This method uses the same boundary calculation as ECB but has an added selection scheme: at each generation the boundary value of the best-so-far performing individual is recorded. When it comes to tour-

nament selection, if two candidate breeders have equal training fitness, then the one whose boundary value is closest to that of the fittest-so-far individual is chosen.

5.5.1.5 OICB

Technique OICB is a boundary search algorithm we have developed which is similar to a golden section search approach[Kiefer 1953]. It attempts to find a near optimal boundary for each individual by recursively searching potentially smaller areas in the range of outputs of the individual, testing different boundary values from a sorted array of program outputs until no further improvement is found. A condensed version of the algorithm is shown in Listing B.1 which can be found in Appendix B: the full algorithm has bounds checking and terminates with the last good value if these bounds are reached or exceeded.

5.5.1.6 OICBP

This approach uses the same boundary algorithm as OICB, but assumes that positive instances are those for which the individual has generated lower values.

5.5.1.7 OICBU

In previous methods whether instances of the negative or positive class were assumed to be above or below the class boundary was determined in advance. In method OICBU this aspect, which we call *polarity* is determined automatically for the entire population. At generation zero both polarities are tested for each individual and whichever setting performs better for the majority of the population is applied to the population as a whole. The OICB boundary calculation algorithm is used here also (listing B.1). We use the term *negative polarity* to describe the situation where values of the negative class are below the decision threshold, and *positive polarity* for the reverse situation.

Table 5.10: Boundary Determination Experimental Configurations

Method	Explanation
Static	Static boundary (set to zero)
ECB	Evolved boundary [Fitzgerald and Ryan, 2011a]
AUCN	AUC boundary, negative polarity
AUCP	AUC boundary, positive polarity
ECBS	ECB, boundary selection scheme
OICB	Optimised Individual Class Boundary
OICBP	OICB, positive polarity
OICBU	OICB, population polarity setting
OICB+	OICB, Individual polarity setting

5.5.1.8 OICB+

The ultimate in individualisation, this method combines the OICB boundary search algorithm with an individual polarity setting. Each individual chooses its own polarity depending on which setting results in fewer classification errors.

5.5.2 Experiments

In this section we provide information about the datasets used in our experiments and detail the experimental configuration and parameters used.

5.5.2.1 Data Sets

Six benchmark data sets have been used for experiments, five from the medical domain and one from the scientific domain. For each data set, the ratio of negative to positive instances is preserved in the training, validation and test sets. The data sets used are BUPA, BT, HS, ION, PIMA and WBC. We have used a function set consisting of addition, subtraction, multiplication and protected division.

5.5.2.2 Experimental Set-up and GP Parameters

General Genetic Programming(GP) parameters used for the experiments are detailed in Table 5.9. Each binary classification task was undertaken

first using a baseline configuration, where the class boundary was set at zero and the sign of the program output was used as the class label. Next, the experiments were repeated using each of the individualised boundary configurations.

One hundred runs were undertaken for each configuration with the same random seeds used for each set of experiments. In each case, the data was divided into two equal sets: Set 1 and Set 2. For the first fifty runs, the system was trained on Set 1 and tested on Set 2, and for the second fifty runs the order was reversed. The fitness measure used for evolution was the number of classification errors of each program. Final fitness values are converted to error rate and % classification accuracy for reporting and comparison purposes. The GP framework used was the Open Beagle Evolutionary Framework [Gagné and Parizeau, 2002]. For the remainder of this document the naming conventions detailed in Table 5.10 apply.

5.5.2.3 Results and Discussion

Results obtained using the baseline Static configuration are compared with those achieved using the various boundary configurations. Outcomes for each of training and test fitness, program size, average nodes processed per run and the generation at which best training individual was found are compared. Training and test fitness scores are detailed in Table 5.11, results for average tree size are contained in Table 5.12, details of average number of nodes processed can be seen in Table 5.13 and best training individual generation information in Table 5.15. The training and test results displayed are the averages, standard deviations and best Individual scores of the hundred best trained individuals for each task. For each set of experiments, the best result in each category is displayed using bold text.

Training and Test Accuracy

When we examine the training and test accuracy scores in Table 5.11 we see that overall the static configuration performed best of the methods on training data, delivering the best score on four of the six data sets and

discovering the highest or joint highest result for best overall individual (although the differences are quite small in most cases). However, when it comes to performance on test data, the static approach does not perform nearly as well, and is out-performed on every task by one or other of the individual boundary methods. Boundary methods AUCN and AUCP performed worst of the boundary approaches and method ECBS with its added selection scheme, delivered very little if any improvement over the ECB method.

Comparing results with those achieved in the previous study, which compared ECB with CDCBD and a standard GP approach, we see that the universally superior training results achieved with ECB in that study are not evident in the current work on the common datasets. It is not the case that the current ECB results are worse than previously, in fact they are better, which is what one expect given that the training set is larger. For some reason the training results for the baseline configuration are better in the current study.

We also note that test results are worse than previously for both baseline GP and ECB configurations, which when taken together with the difference in training results between the two studies means that there is evidence of greater variance error (over-fitting) in the current study.

There are several differences in the experimental set-ups between both approaches: the first employed a generational replacement strategy with elitism whereas this one uses a steady state approach; the initial study did not use ERCs whereas the current study does; runs in the second investigation run for 10 generations longer; the first had a tournament of size 4 and the second a tournament size of 5; and the first investigation used a three dataset methodology with a validation set and selected individuals for testing based on validation performance, whereas the current study involved equal sized training and test sets where the best trained individuals were used for testing. Also, different random seeds were used for each study.

Of these differences, those that may effect training performance are the replacement strategy, use of ERCs, tournament size, different sized and potentially different contents of data partitions, different random

seeds and longer training time. We undertook further experiments to isolate the difference in performance on the BUPA, PIMA and WBC datasets and summarise our findings as follows:

1. There was no consistent difference in the results when ERCs were used/not used;
2. Aside from the improvement for both configurations due to the larger training set, the difference in training performance between the baseline and ECB configurations in the second study is partially due to the increased training time: ECB has much better training fitness at the beginning of evolution, but as evolution progresses, the baseline configuration learns the zero boundary and “catches up” with the ECB set-up;
3. The increased learning time also contributes to the over-fitting apparent in the second study. This is consistent both with our intuition about over-fitting and with the findings of Freund and Schapire [1996] among others;
4. A proportion of the observable increase in over-fitting seems to be attributable to the steady state replacement strategy and the tournament size; See Chapter 7 Section 7.2 for a detailed examination of variance error due to replacement strategies and tournament size;
5. The differing methods of selecting best-of-run solutions for test purposes may not have influenced training or test average accuracy. This is consistent with the research of Gagné et al. [2006] which does however suggest that the variability of test results may have been effected: standard deviation is lower in the first study where the best performing individuals on the validation set were chosen for test.

Details of these clarifying experiments can be found in Appendix C.

Statistical Tests

Applying paired t-tests for each method across the datasets indicated

Table 5.11: Training & Test Results for Selected Boundary Determination Approaches. Best results in each category are shown in bold font.

Data	Method	Training			Test		
		Avg Fitness	StdDev	Best Fitness	Avg Fitness	StdDev	Best Indi
BT	Static	82.49	0.96	85.25	77.31	4.81	80.16
	ECB	82.02	0.94	84.71	77.67	4.85	80.16
	AUCN	81.40	1.37	84.46	77.00	6.88	81.23
	AUCP	80.13	1.67	83.91	75.81	20.64	79.35
	ECBS	82.25	0.87	84.45	77.77	4.68	80.42
	OICB	82.06	0.71	84.45	77.27	3.76	79.60
	OICBP	82.02	0.70	83.91	77.44	4.20	80.43
	OICBU	81.93	0.70	83.91	77.35	3.87	79.90
OICB+	81.98	0.67	83.91	77.36	4.50	80.16	
BUPA	Static	80.02	2.50	88.95	62.99	8.77	72.07
	ECB	79.65	2.02	84.88	67.00	6.75	74.40
	AUCN	79.23	1.98	83.72	63.30	6.65	73.80
	AUCP	79.21	2.06	83.72	64.00	5.60	70.00
	ECBS	80.05	1.83	86.82	66.60	7.37	73.84
	OICB	79.41	1.70	83.72	66.27	7.28	73.84
	OICBP	79.38	1.69	85.46	66.01	7.92	74.41
	OICBU	79.08	1.58	82.55	66.86	6.71	73.84
OICB+	79.50	2.02	85.46	66.93	6.66	72.09	
HS	Static	83.61	2.07	88.81	72.35	5.17	79.00
	ECB	81.30	1.73	85.52	74.83	4.87	80.25
	AUCN	82.13	2.38	87.50	71.88	7.21	79.60
	AUCP	80.96	2.38	87.50	71.41	8.50	78.94
	ECBS	81.59	1.98	85.52	74.00	6.10	80.20
	OICB	81.71	1.37	85.53	74.21	3.78	78.95
	OICBP	81.61	1.58	87.50	74.55	4.07	78.95
	OICBU	81.41	1.28	84.21	74.41	4.36	78.29
OICB+	81.69	1.37	85.51	74.41	4.15	78.95	
ION	Static	95.69	1.87	99.42	84.05	5.04	92.00
	ECB	95.24	2.04	99.42	84.50	5.72	92.00
	AUCN	95.46	2.33	98.85	84.70	5.27	90.86
	AUCP	94.00	3.26	99.42	85.00	8.24	93.14
	ECBS	95.38	2.14	98.86	84.55	5.42	92.57
	OICB	91.77	3.47	98.28	85.23	6.65	92.77
	OICBP	90.65	2.95	96.57	84.52	7.18	93.71
	OICBU	90.61	3.13	96.57	84.41	7.55	92.57
OICB+	91.40	2.99	97.14	85.40	6.89	93.41	
PIMA	Static	75.46	2.42	80.52	65.44	10.87	74.29
	ECB	80.08	1.25	83.11	75.37	7.18	78.80
	AUCN	80.01	1.57	83.90	73.91	6.00	77.92
	AUCP	79.99	1.41	83.88	73.96	5.47	77.14
	ECBS	81.27	1.48	84.15	75.14	7.45	79.22
	OICB	79.97	1.41	83.11	75.51	6.85	80.52
	OICBP	80.58	1.60	83.64	75.07	6.46	78.70
	OICBU	80.15	1.41	82.86	75.07	6.46	78.70
OICB+	80.11	1.72	84.15	75.55	6.63	79.22	
WBC	Static	98.86	0.40	99.70	95.65	3.72	98.28
	ECB	98.40	0.42	99.71	95.61	2.63	97.35
	AUCN	98.45	0.39	100	95.59	4.95	97.94
	AUCP	98.50	0.37	99.42	95.61	4.41	97.94
	ECBS	98.68	0.44	99.71	95.48	3.45	97.64
	OICB	98.62	0.34	99.70	96.45	2.79	97.94
	OICBP	98.76	0.32	99.70	96.37	3.49	97.94
	OICBU	98.69	0.29	99.41	96.46	3.19	97.94
OICB+	98.62	0.27	99.41	96.47	3.26	98.82	

that the performance of certain methods were statistically significant in several cases. However, to gain a clearer insight as to which method performed best overall we carried out the non-parametric Friedman test as recommended by Demšar [2006]. The Friedman test is regarded as a suitable test for the empirical comparison of the performance of different algorithms [Gerevini et al., 2005]. The test resulted in a p-value of 0.040, indicating that there was a significant difference between some of the methods. Post-hoc tests showed that overall, the performance of the OICB+ method was statistically significant across the datasets when compared to the Static approach with a p-value of 0.041. A box plot of the Friedman analysis can be seen in figure 5.5. The best performing algorithms are ranked highest: where 1 represents the highest rank and 9 the lowest. The plot clearly shows that when compared across the various datasets, the static method ranks lowest of all the methods investigated, whereas the OICB+ method ranks highest. Methods ECB and OICB also performed well.

Average Tree Size

Results for average tree size shown in Table 5.12 represent the tree size for the best-of-run individuals averaged over all runs for each task. The results indicate that all of the individual boundary determination methods produce significantly smaller trees overall, but no single method consistently generates the smallest programs. On closer examination we observed that in each case the average tree size of the best-of-run individuals was smaller than the average of the population from which they were selected. This suggests that for this type of problem, smaller trees may generalise better. This is consistent with previous work of Nordin and Banzhaf [1995] and the general principle of Occam's razor.

Smaller program size combined with a simple function set may yield evolved results which are easier to interpret. As mentioned earlier, comprehensibility of the evolved solutions is important in certain problem domains, notably the medical domain: where the reasons for a particular classification may be as important as the classification itself. Similarly, in areas such as texture classification, it is useful to learn if the system

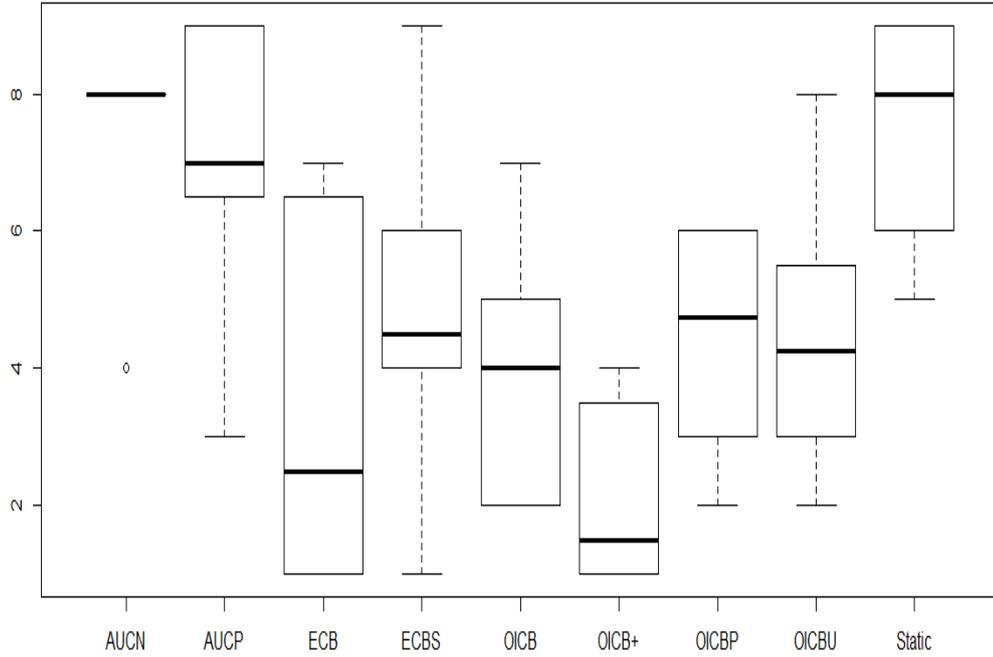


Figure 5.5: Relative ranking of Boundary Determination Algorithms, where 1 represents the best performance and 9 the worst.

Table 5.12: Boundary Determination Experiments Average Tree Size

Method	BT	BUPA	HS	ION	PIMA	WBC	Average
Static	288.54	313.68	313.98	214.94	273.88	237.06	273.68
ECB	227.58	217.34	203.58	214.86	191.56	137.86	198.79
AUCN	207.60	168.84	206.66	202.88	169.22	124.38	179.93
AUCP	125.14	179.54	178.44	121.24	197.84	141.92	157.35
ECBS	225.64	222.74	205.12	192.52	181.40	120.30	191.28
OICB	207.36	188.04	245.26	63.20	175.34	109.04	164.71
OICBP	212.16	181.74	238.78	85.46	240.88	163.18	187.03
OICBU	197.74	173.74	217.82	70.20	173.18	112.38	157.50
OICB+	218.08	199.16	215.48	106.46	185.7	109.60	172.41

Table 5.13: Avg. Nodes Processed (1000s)

Data	Static	ECB	AUCN	AUCP	ECBS	OICB	OICBP	OICBU	OICB+
BT	8808	8840	7125	4540	8176	7183	7780	5640	7229
BUPA	9022	6914	6893	7118	7486	5939	6453	7060	6476
HS	9593	8309	7852	8230	7933	9006	9369	8428	8327
ION	6518	6459	6668	5169	5927	2089	3625	3014	3842
PIMA	8015	7431	6398	7546	5799	5941	7526	6119	6713
WBC	8156	7425	7875	8438	4605	7782	7275	7723	7233
Avg.	8352	7563	7135	6840	6654	6323	7004	6330	6636

has discovered discriminating features not previously understood.

Average Nodes Processed per GP Run

Table 5.13 shows the average number of nodes processed per GP run for each method. The results show that for the current experiments, fewer nodes were processed during runs where individual boundary methods were used. For all datasets, the static method had the highest number of nodes processed. In GP the most significant proportion of the computational cost is incurred in evaluating individual fitness. It follows that smaller individuals with fewer nodes to process should deliver shorter run times and use less memory overall. There is some computational cost incurred in calculating the class boundary, which depends on the particular boundary method chosen. However such costs are small compared with fitness evaluation, and are more than offset by the reduced program size. As noted earlier, one of the perceived drawbacks to the application of GP to classification problems is the belief that it has a requirement for long training times when compared with other classification methods [Loveard and Ciesielski, 2001]. We suggest that the results for the current experiments demonstrate that use of individual boundary methods can deliver dramatically reduced run times for this type of classification problem with minimal trade off, as the boundary calculation is integrated with the normal evaluation process and does not involve any additional evaluations. Also, as the best-of-run Individual tends to be discovered sooner,

Table 5.14: Comparison of OICB and other Machine Learning Algorithms % Accuracy, ranked from top to bottom according to average accuracy. AdaBoostMI is a Weka implementation of Adaboost [Freund and Schapire, 1996] and the implementations of SVM and RandomForest are those of Chang and Lin [2011] and Breiman [2001] respectively.

Method	BT	BUPA	HS	ION	PIMA	WBC	Average
MLP	79.35	67.72	75.00	86.00	75.95	95.67	79.95
OICB+	77.36	66.93	74.41	85.11	75.55	96.47	79.31
AdaBoostM1	77.74	63.80	72.03	89.71	74.99	95.20	78.91
Logistic	77.34	67.73	73.35	81.42	76.17	97.20	78.87
RandForest	76.80	62.78	67.76	92.28	74.88	96.35	78.48
J48 (C4.5)	78.01	60.47	69.38	88.57	73.69	94.70	77.47
SVM	76.00	59.34	71.70	91.71	65.23	95.43	76.57
Static	77.31	62.99	72.65	84.05	65.44	95.65	76.35
BayesNet	74.39	49.12	71.38	89.43	73.95	97.79	76.01

it should be possible to further reduce training times by implementing an early stopping mechanism.

5.5.2.4 Comparison with Other Methods

Earlier in this chapter we compared ECB classification accuracy with that of the CDCBD method of Zhang and Smart [2004]. We also compared with results for the same datasets reported in other research for several GP variants and several other machine learning algorithms. However, when making comparisons with other methods, it frequently happens that one cannot obtain detailed information concerning the set-up of those experiments, particularly with regard to data partitioning, which may have a significant impact on results. Thus, such comparisons, while useful, must be taken with “a pinch of salt”.

For this work, we have carried out our own experiments using a variety of popular machine learning algorithms² to classify the six datasets using the Weka Machine learning tool [Hall et al., 2009]. In order to gain as fair

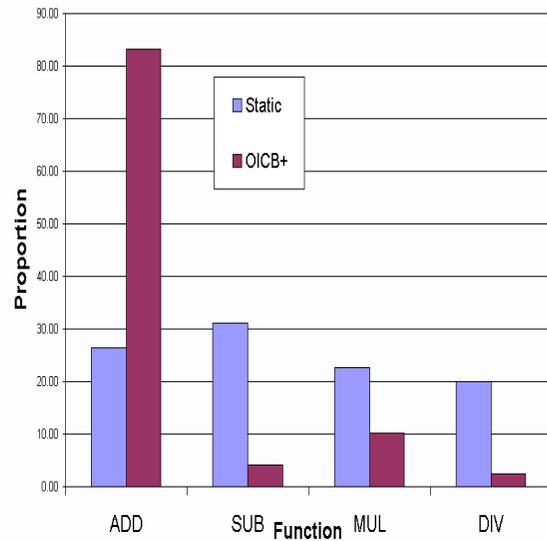
²See Weka documentation [Hall et al., 2009] for details of these machine learning algorithms.

a comparison as possible, we have used the same data splits that were employed in the main experiments for this task. In configuring these experiments we have used default Weka parameters for each algorithm. The results of comparing both the baseline GP configuration and the OCIB+ method with these other machine learning algorithms as seen in Table 5.14 show that the proposed method delivers a very competitive performance for the majority of the problems tackled.

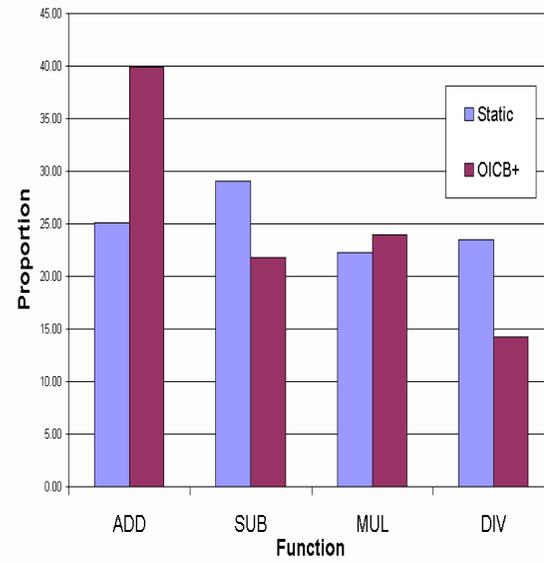
5.5.2.5 Program Analysis

Post-run analysis of the best-of-run individuals in terms of functions and attributes used yielded some interesting findings which are illustrated in Figures 5.6 and 5.7. We compared the behaviour of the static configuration with that of OICB+ which was the best performing of the boundary configurations. Analysing the average number of attributes used per individual, we saw that the results were very similar for all of the datasets except WBC and ION: for the former the OICB+ method used on average one attribute less than the static approach, but for the ION data set (which has thirty-two attributes) the difference was quite dramatic: the static method used on average seventeen attributes whereas the OICB+ configuration used only seven.

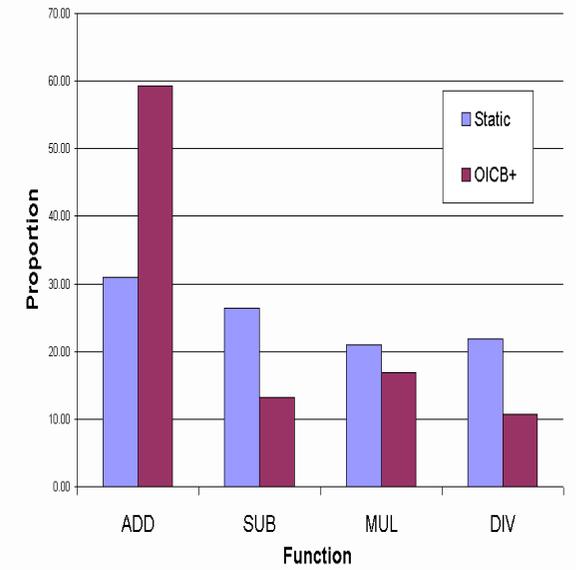
Looking at the proportions of each function utilised by each program also yielded some interesting results: for the PIMA, WBC and ION datasets, while the static method utilised each of the functions in roughly similar proportions, the OICB+ configuration made much more frequent use of addition, and in the cases of WBC and ION this difference was quite pronounced.



(a) BUPA



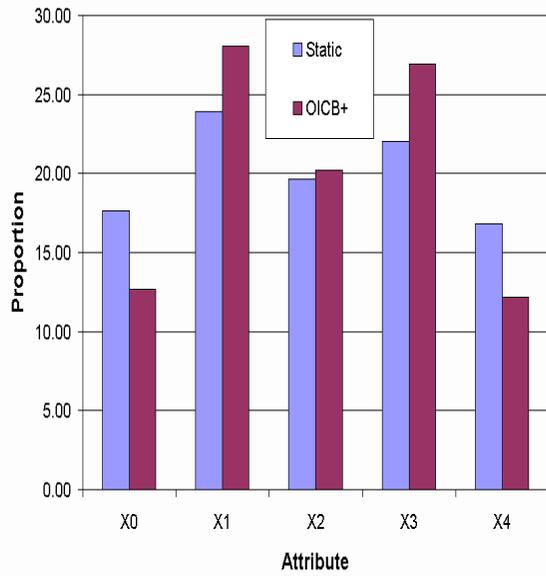
(b) PIMA



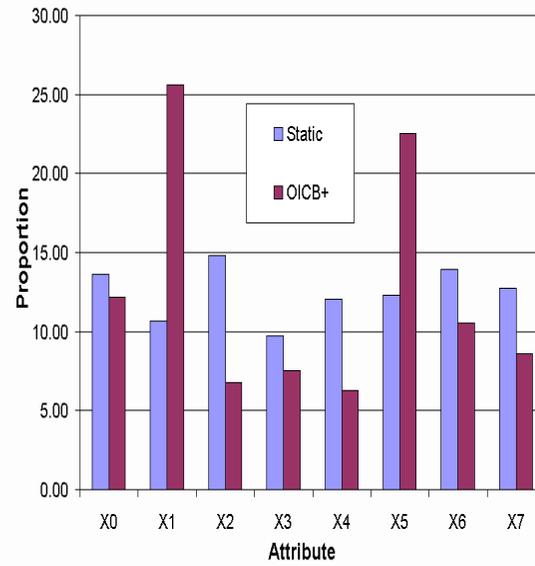
(c) WBC

Figure 5.6: Comparison of function usage for OICB+ and Static configurations.

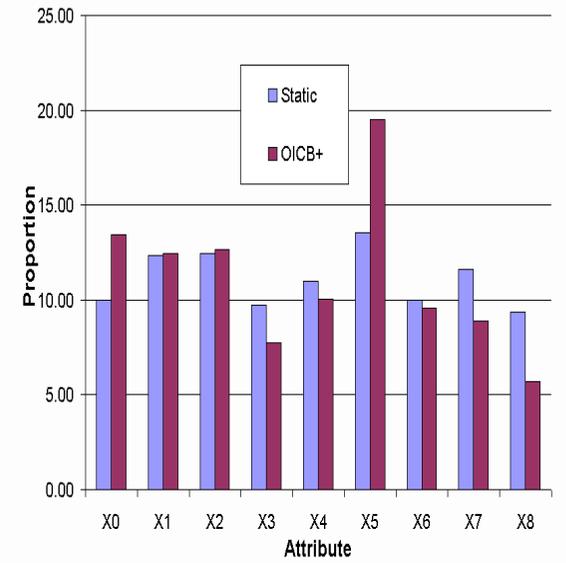
Finally, we examined the proportion of each attribute used by both configurations. Again, there were notable differences except for BT and HS which were fairly similar. For the PIMA dataset, the static configuration used roughly equivalent numbers of each attribute but the OICB+ method highlighted the importance of the attributes X1 (blood glucose) and X5 (body mass index) both of which are known indicators for diabetes. For the BUPA data set, both approaches used attributes X1 and X3 more frequently than the others, but with the OICB+ configuration this was much more pronounced. The static method used the available WBC attributes in very similar proportions, but the OICB+ method made greater use of the X5 (bare nucleoli) attribute. Bare nucleoli are cells without a nucleus, and are often found where cancer is present.



(a) BUPA



(b) PIMA



(c) WBC

Figure 5.7: Comparison of attribute usage for OICB+ and Static configurations.

Table 5.15: Generation of best-of-run Individual

Data	Static	ECB	AUCN	AUCP	ECBS	OICB	OICBP	OICBU	OICB+
BT	49.23	39.46	42.19	39.33	41.11	38.24	37.13	37.04	36.58
BUPA	51.19	44.98	34.45	31.85	43.33	42.72	40.21	41.41	40.99
HS	46.75	32.41	35.90	26.41	34.13	37.12	32.42	35.13	33.04
ION	45.67	43.20	39.98	31.27	40.55	39.03	27.21	31.93	25.96
PIMA	54.72	37.21	31.81	34.46	48.99	39.84	47.64	36.52	38.58
WBC	37.38	25.24	17.68	19.18	32.24	16.95	27.57	17.81	17.33
Avg.	47.49	37.08	33.67	30.42	40.05	35.65	35.36	33.31	32.08

This analysis of the best-of-run individuals would seem to suggest that in general, the OICB+ method results in programs which are less complex and more meaningful in the context of understanding the reasons for a possible classification. We observed similar behaviour in previous experiments, but as we had not supplied the system with constants on that occasion, we surmised that in cases where a zero boundary was sub-optimal the static method may have been using the attributes to synthesise constants. For the current experiments the system had ephemeral random constants available.

5.5.2.6 Generation of Best Individual

On examining details of runs for the various experiments, we learned that in all cases the average population fitness and best individual fitness tended to be higher at generation 1 for each of the individual boundary methods when compared with the static configuration, and was significantly higher most of the time. Also, on average, these methods discovered the best-of-run individual sooner. This latter information is captured in Table 5.15.

5.5.3 Summary

In this chapter we have investigated several methods that can be used with GP to improve performance on binary classification tasks, and aris-

ing out of this investigation we propose a combined technique *OICB+*. We believe that this flexible and dynamic method can be used to automatically calculate near optimal boundary and polarity settings for a diverse range of classification datasets. Experimental results suggest that the method can improve test results, reduce run times and produce smaller more easily interpreted programs. Most importantly, the technique offered good results on test data, and we have established that these results are likely to be statistically significant.

This investigation suggests that while a particular boundary selection method may deliver better performance for a given problem, no single method performs best on all problems studied. We propose a new flexible combined technique which gives near optimal performance across each of the tasks undertaken. Experimental results obtained suggest that the strategy can improve test fitness, produce smaller less complex individuals and reduce run times. Our approach is shown to deliver superior results when benchmarked against a standard GP system, and is very competitive when compared with a range of other machine learning algorithms.

Also, since it is accepted that any multi-class problem of size n can be reduced to n binary classification tasks using a “one versus all” approach [Loveard and Ciesielski, 2001; Smart and Zhang, 2005], it follows that the application of tools that facilitate the development of more effective binary classifiers can offer benefits in the multi-class situation.

5.6 The Role of Self-adaptation

Earlier in this chapter we have demonstrated the benefits of using individualised, dynamic boundary determination methods over static population based approaches. In this section, we extend the notions of individualism and self-adaptation and investigate if individualised, self-adaptive genetic operators may also have a role to play in counteracting monolithic bias and improving generalisation in GP.

5.6.1 Previous Work on Self-Adaptive Operators

Angeline [1995] distinguishes adaptive evolutionary computations according to the level at which the evolutionary operators operate: *population level* adaptation adjusts evolutionary parameters on a global scope, where all adaptations apply to the entire population; *individual level* adaptive techniques modify individualised parameter settings, allowing the individual a measure of control over his own destiny; and *component level* adaptations manipulate different components of an individual independently of each other. In all cases the objective is to dynamically adjust the values of the evolutionary parameters in order to encourage the system to produce better solutions faster.

Self-adaptation has been popular for some time in the field of evolutionary computation, having notable success in the area of evolution strategies [Kramer, 2010]. In a review of evolutionary self-adaptation, Kramer [2010] noted that currently self-adaptation mechanisms most often involve modification of mutation step sizes. This is not surprising, as the accepted wisdom convincingly argued by Banzhaf et al. [1996] amongst others, is that it is actually mutation that brings innovation to GP. Certainly, this view is widely reflected in the choices of self-adaptive mechanisms found in the literature.

In the fields of GP and its many extensions, there has been considerable research on the topic of population level adaptation: Yin et al. [2007] reported significant improvement over canonical GP when they employed a feedback adaptive mechanism which increased the probability of mutation once a predetermined number of generations had elapsed without an

improvement in fitness of the best-so-far individual. Fagan et al. [2012] applied a similar approach to Grammatical Evolution [O’Neill and Ryan, 2003] which they called *Fitness Reactive Mutation* in which mutation was increased in 0.01 increments when a fitness plateau was detected. In other work, Kim et al. [2011] investigated the effect of porting some popular EA adaptive techniques to GP, and reported that an *adaptive pursuit* mechanism which operates by applying a greater increase in probability to the most effective operator, and correspondingly decreased probability to the other operators was more successful than other approaches.

Compared with population level adaptation, individual level adaptation has not been given the same attention. Goldman and Tauritz [2011] applied a method of *Self Configuring Crossover (SCX)* with Linear Genetic Programming (LGP) to a variety of benchmark problems and reported good results when compared with other highly tuned methods. Harper and Blair [2006] suggested an adaptive process for selecting an appropriate crossover variant. In the suggested method, *Self-Selecting Crossover* used with GE, each individual had an associated crossover operator which was either randomly selected or inherited depending on when it originated. Their results demonstrated that self-selecting, variable crossover, delivered superior results than either of the (two) trialled crossover operators did when operating alone.

Al-Madi and Ludwig [2012] proposed several different adaptive techniques with GP for classification tasks, which operated variously on the function set, the selection process or the evolutionary operators. The results of their experiments indicated that each of the trialled methods achieved competitive results in terms of classification accuracy, together with run times which were appreciably shorter than those achieved using a standard GP configuration.

On a larger scale, Riekert et al. [2009] developed a new GP variant *Adaptive Genetic Programming* which extended GP to incorporate variable elitism and culling, designed to operate in dynamic environments.

An alternative to self-adaptation is an approach known as *Adaptive Allocation Rule* [Agrawal et al., 1988], where the values of genetic operator probabilities are adjusted according to a learning rule depending

on the quality of the new solutions generated by the operators. The key difference between self-adaptive operators and adaptive allocation rule methods is that in the former the process of adaptation is usually closely coupled with evolving fitness, whereas in the latter the learning rule is “out of the evolutionary loop” [Thierens, 2005]. See for example, [Fialho et al., 2008] and [Maturana and Saubion, 2008] for relatively work in this area.

There have been various approaches taken to adaptive *selection pressure* in the literature. Early work by Ryan [1994] proposed *disassortative mating* which ensured that parents selected for breeding were phenotypically different. In this method the population was divided into two categories: *Pygmies and Civil Servants* based on the desirable properties of shortness or efficiency respectively. The method was effective in maintaining diversity in the population. Luke and Panait [2002] investigated two fitness ranking methods where individuals were placed in “buckets” based on fitness similarity. Their method *Lexicographic Parsimony Pressure* delivered competitive results with significantly smaller programs. In more recent work Xie and Zhang [2009]; Xie et al. [2006] employed a clustering approach, where individuals were clustered based on various criteria, such as fitness rank or phenotype, where tournaments involved clusters rather than individuals, and where an individual to be used for mating or mutation was chosen at random from the winning cluster. See [Fang and Li, 2010b] for a review of various tournament selection strategies.

The approach that we have taken is to combine population level and individual level self-adaptation: mutation and crossover rates are based on the *relative fitness* of the individual weighted by the progress of the population in the search process. We combine this with adaptive tournament selection in an effort to produce a holistic environment, where promising individuals are encouraged to explore and weak individuals are encouraged to change early in the process. As evolution progresses, this balance shifts such that better individuals have a higher probability of mutation than previously and weaker individuals a higher probability of crossover. In addition, we apply weak selection pressure early in

the evolution process to encourage exploration. As learning progresses, selection becomes increasingly more elitist, placing greater emphasis on exploitation.

5.6.2 Proposed Method

The proposed methods extend the basic idea that the fittest individuals in the population should be allowed to reproduce with higher probability than the less fit. Our strategy applies higher mutation rates to relatively unfit individuals in the early stages of evolution and higher crossover rates to fitter candidates. As evolution progresses this balance changes subtly such that fitter individuals have an increased chance of mutating. Synergistic with this technique, we also incorporate adaptive tournament selection with the aim of effectively exploiting acquired knowledge in the system.

5.6.2.1 Individualised Adaptive Genetic Operators

The system is designed to operate within a range of predefined values for crossover and mutation. In the early stages of the search the fittest individuals have higher probabilities of reproducing whereas the less fit have higher probabilities of being mutated. An initial minimum and maximum percentage mutation value apply, whereby the minimum value is applied to the fittest individual(s) and the maximum is applied to the least fit individual(s) at the first generation. A weighting (W) is calculated, which is the current best fitness value (CBF) as a proportion the original best fitness value (OBF) found in the population at the initial generation. As evolution proceeds, this value naturally decreases, thus reducing the rate of crossover and increasing the rate of mutation for the fitter individuals.

In the first generation the OBF is stored and the weight W is set to 1. In each generation the population fitness values are normalised in the range $mutMin$ to $mutMax$ where $mutMin$ and $mutMax$ are input parameters to the system. The weight W is calculated where $W = CBF/OBF$ i.e. Current Best Fitness / Original Best Fitness. Finally crossover and

mutation probabilities (expressed as a percentage) for each individual are applied as shown in figure 1

$$\begin{aligned} Cx(i) &= 100 - mutMax + ((mutMax - N(i)) * W) \\ Mut(i) &= 100 - Cx(i) \end{aligned} \tag{5.2}$$

Where $Cx(i)$ is the crossover probability expressed as a percentage, applied to program i , $Mut(i)$ is the mutation probability applied to program i and $N(i)$ is the *normalised* fitness value for the i 'th individual and W is a weight calculated based on proportion best fitness.

Thus for an individual program the probability of crossover or mutation depends on the *relative* fitness of the individual itself together with the learning progress of the population as a whole. For example, at generation 1 given a mutation range of 0 – 100, programs with the best fitness score will have a crossover probability of 99% and a mutation probability of 1%:

$$Cx = 100 - 100 + ((100 - 1) * 1) \tag{5.3}$$

It is important to note that the supplied minimum and maximum values for mutation are only applied at the first generation. Thereafter values for these probabilities are determined by the weight and are not deterministic across problems. How the weight changes will depend on the extent to which the system is learning, and will vary from problem to problem; for easy problems where the current best fitness improves rapidly, so that it becomes a small percentage of the original value, the fitter individuals will have higher values for mutation towards the end of evolution, than would be the case for more difficult problems.

5.6.2.2 Adaptive Tournament Selection

An effective search strategy involves achieving a good balance of exploration and exploitation. In GP, and similar paradigms, the exploration function is achieved using genetic operators such as crossover and mu-

tation, whereas selection is the main instrument of exploitation and is critical in determining which individuals have the opportunity to reproduce or be transformed.

We hypothesize that the GP system should be allowed unconstrained freedom to explore the search space early in the evolutionary process. Thus, our proposed method begins with a non-elitist tournament size of 2. As the learning process continues, tournament selection will become progressively more elitist, depending on how well the system is performing on the training data: we maintain a threshold value t such that if the mean training fitness fails to improve for t generations, the size of tournaments is increased by 1. For this initial investigation t is set to 10. We apply a maximum tournament size of 10 for this investigation.

5.6.3 Experiments

In this section we provide information about the data sets used in our experiments and detail experimental configurations and Genetic Programming parameters which we have chosen to apply.

5.6.3.1 Experimental Set-up and GP Parameters

For these experiments we have used five benchmark data sets: BT, BUPA, HS, ION and WBC. Parameters used for the experiments are detailed in Table 5.16. One hundred runs were undertaken for each configuration with the same random seeds used for each set of experiments. The fitness measure used for evolution was the number of classification errors of each program. Final fitness values are converted to error rate and % classification accuracy for reporting and comparison purposes. For the remainder of this document the naming conventions detailed in Table 5.17 apply.

Experimental Configurations

In this investigation we compare our proposed methods with three other configurations: a tuned standard GP configuration (TGP), the feedback adaptive method of Yin et al. [2007](FAGP) and a method which ran-

Table 5.16: GP Parameters for Self-adaptive Individualised Genetic Operator Experiments.

Parameter	Value
Strategy	Steady State
Initialisation	Ramped half-and-half
Selection	Tournament
Tournament Size(1)	3
Tournament Size(2)	2+
Initial Min Depth	4
Initial Max Depth	6
Max Depth	17
Function Set	+ - * /
Population	300
Max Gen	100

domly selects a mutation probability for each individual. In the latter configuration, for each individual at each iteration, a percentage mutation value $rsMut$ in the range 0 – 100 is randomly selected and applied to the individual together with a complementary crossover value of $100 - rsMut$.

We have chosen two variants utilising self-adaptive individualised GP: the first of these, uses the proposed self-adaptive mechanism without tournament adaptation, known as OSA from now on; in the second method we introduce a variable adaptive tournament size, and combine this with self-adaptive evolutionary operators, and we refer to this configuration as TOSA from now on.

In order to make a fair comparison between the various methods. we firstly ran a series of experiments with standard GP using mutation values in the range 5% to 40% with two hundred runs of each for every dataset. Having studied these results and based on the criteria of best average test fitness and best overall test individual, we selected a value of 20% for mutation for the TGP configuration. In the original experiments of Yin et al. [2007] crossover and mutation rates were tuned to suit a task relating to options pricing and were set at 40% and 60% respectively in the initial generation. From then, if a new best of run individual was not found in ten generations, mutation was increased by 2% up to a maximum

Table 5.17: Method Configurations for Individualised Self-adaptive Genetic Operator Experiments.

Method	Description	Mut. Min%	Mut. Max%
GP	Tuned GP	20	20
FAGP	Feedback Adaptive GP Yin et al. [2007]	20	40
OSA	Individual self-adaptive GP	1	100
TOSA	Individual self-adaptive + Tournament	1	100
RAND	Random Individualised operators	1	100

of 90%. Once a new best of individual was discovered, the probabilities of crossover and mutation reverted to the original setting. For comparison purposes, and because our earlier experiments demonstrated that 20% was a good mutation value for TGP, we set the initial values of crossover and mutation at 80% and 20% respectively for the FAGP configuration, with increments of 2% once a threshold was reached, whereby a new best of run individual had not been found.

In their work, Yin et al. [2007] ran their experiments for a large number of generations (up to 800) which suited the problem undertaken. As the experiments in this study were run for 100 generations, which is quite a lot fewer by comparison, we investigated whether we should reduce the threshold of 10 generations to some smaller value to reflect the shorter evolution. In experiments of twenty runs using a threshold value of 5 generations, it became clear that this lower setting resulted in a marked deterioration in performance, so we maintained a threshold value of 10 for the current work.

For the OSA and TOSA configurations, a maximum value for mutation of 100 is permitted, where effective percentage values for each individual are calculated as shown in Figure 5.2. For the remainder of this document, the naming conventions outlined in Table 5.17 shall apply.

5.6.3.2 Results and Discussion

Experimental results can be seen in Table 5.18. The results values represent classification accuracy in terms of percentage of instances correctly classified. These are end of run statistics for the best performing individuals taken from the hundred runs of each configuration. The best results

in each category: median best training fitness; best overall trained individual; median best test fitness and best test individual, are highlighted in bold font.

Although the initial mutation rate selected was carefully tuned to suit the TGP method, being based on a large number of experiments, this system was the least successful of the various methods investigated. Although the results are close for each of the problems undertaken, the TGP did not achieve the best score under any heading for any dataset which suggests that adaptive methods are likely to be more effective than static ones for this type of problem.

On the Blood Transfusion data set, the TOSA configuration which utilises individualised self-adaptive crossover and mutation, combined with adaptive tournament size delivered the best result in each category. Overall, the tuned GP set-up TGP produced the worst results.

Looking at results for the Bupa Liver Disorders data set, we can see that the tuned standard GP configuration TGP again performed worst of the various configurations: even the RAND approach performed better on the test data. Unsurprisingly, given the adaptive tournament size, the TOSA configuration delivered the best results on the training data. It also produced the best result for median test fitness, and shared the highest score for best overall individual on test data with that produced using the OSA set-up.

The results for the Habermans' survival dataset illustrate that the TOSA configuration also performed well on this task. It achieved the highest result for best trained individual together with the best overall individual score on test data on a score of 80.92%. All configurations produced very similar results for median test fitness.

Turning our attention to the results for the ION data set we can see that this seems to be quite an easy task to score well on as two of the configurations achieved at least one perfect classification in the training phase. For this data set the best results in test data categories were achieved with one of the methods which employed an individualised self-adaptive approach. In this instance, the tuned GP (TGP) configuration was more successful than previously, with relatively good results

Table 5.18: Best Training & Test Results for Self-adaptive Experiments.

Data	Method	Training		Test	
		Med. Fitness	Best Fitness	Med. Fitness	Best Individual
BT	TGP	80.01	82.57	77.48	79.89
	FAGP	81.50	84.58	77.47	79.89
	RAND	81.10	84.01	77.47	80.69
	OSA	81.90	84.38	77.75	80.70
	TOSA	82.44	84.72	78.02	81.23
BUPA	TGP	81.40	84.88	65.69	72.67
	FAGP	80.81	84.88	66.86	74.41
	RAND	80.23	84.30	66.26	73.84
	OSA	79.94	83.14	67.44	75.00
	TOSA	81.69	87.79	68.02	75.00
HS	TGP	81.90	84.87	75.65	80.26
	FAGP	82.56	84.87	75.66	79.60
	RAND	80.92	84.21	75.66	79.60
	OSA	81.58	84.86	75.66	79.60
	TOSA	82.24	86.84	75.65	80.92
ION	TGP	96.98	100	86.20	93.10
	FAGP	95.69	100	86.20	93.96
	RAND	95.25	99.14	86.20	92.24
	OSA	96.12	99.14	86.21	96.55
	TOSA	96.12	99.14	87.07	96.55
WBC	TGP	97.94	99.41	95.88	97.35
	FAGP	98.53	99.11	96.10	97.64
	RAND	98.38	99.12	95.88	97.35
	OSA	98.53	99.41	96.18	97.35
	TOSA	98.53	99.41	96.18	97.64

for training fitness.

Finally, for the Wisconsin Breast Cancer data set, the results were very close for all configurations. The OSA and TOSA methods were slightly better than the other methods on median test fitness whereas the FAGP and TOSA approaches shared the high score for overall best individual.

5.6.4 Summary

In this chapter we have investigated a self-adaptive individualised approach for controlling crossover and mutation probabilities during evolution. This method was investigated both on its own and combined with a technique which adaptively increased the tournament size during evolution. These two methods were compared with; a standard GP approach which was tuned to use mutation and crossover values which were empirically selected, a method from the recent literature proposed by Yin et al. [2007], and an approach which employed random selection to set individual mutation and crossover probabilities.

The results demonstrate, that while the scores for several of the methods are quite close, the tuned standard GP approach is clearly the poorest performer overall. The results achieved using the RAND method were competitive with both the TGP and FAGP methods, perhaps confirming the conclusions of some of the previous work discussed in section 5.6.1 [Luke and Spector, 1997; Spears et al., 1992; White and Poulding, 2009] which suggest that traditional parameter settings for standard crossover and mutation are not necessarily optimal.

While all of the adaptive approaches investigated here performed better than the TGP configuration, suggesting that adaptive and/or individualised approaches may be superior to static population based methods, the benefits of the application of individualised self-adaptive operators as implemented here are not conclusive. However, the performance of the TOSA configuration, which produced the best results overall on the all-important test data does suggest that an approach which adapts the tournament size during evolution may be beneficial in reducing variance error. In this respect, the experimental results support the intuition that

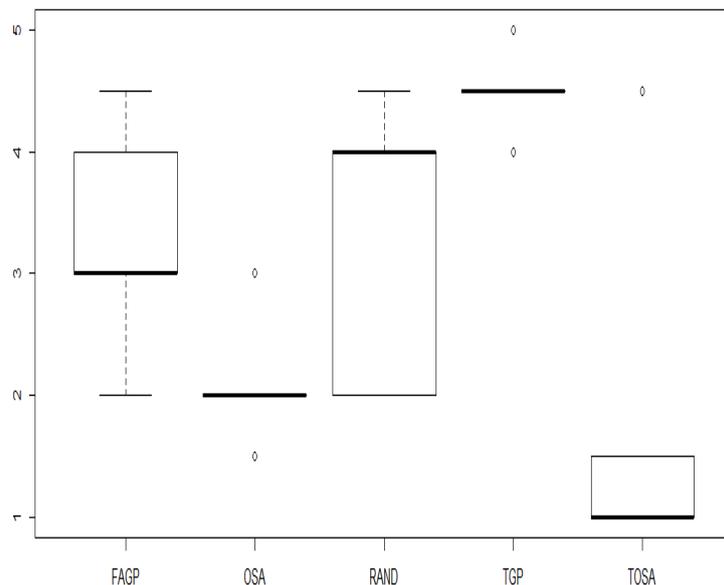


Figure 5.8: Methods ranked from 1 to 5 based on average Test Accuracy, where 1 is best and 5 is worst.

an initial small tournament size which becomes progressively more elitist may be appropriate to encourage exploration early in the evolutionary process and to subsequently exploit the fruits of that exploration as evolution progresses.

To determine if any of the results were statistically significant, we undertook a non-parametric Friedman test which produced a p-value of 0.032. Post-hoc tests indicated that the difference between TOSA and TGP is highly significant (0.032) and between OSA and TGP is slightly less significant (0.071). The corresponding Friedman plot can be seen in Figure 5.8.

5.7 Research Outcomes

Towards the beginning of this chapter, we restated the core question “can new *individualised* approaches to selection of *boundary values* and

genetic operator probabilities lead to the evolution of more generalisable solutions?”. Throughout this chapter we have outlined sources of what we consider *inappropriate bias* in traditional approaches to binary classification using GP, and we have proposed several individual level adaptations which may be used to avoid such biases or to replace them with more appropriate inductive bias.

When using a standard approach with a class boundary of zero and an arithmetic function set, without use of negative ephemeral constants, any successful strategy will *require* evolved expressions to use the subtraction operator *merely to meet the zero boundary constraint*, possibly without any improvement in its true discrimination ability. In Section 5.5.2.5, we can see that using our OICB+ approach, the subtraction operator is used far less frequently than when a standard approach was used and we observed this behaviour both when ephemeral random constants were used and when they were not.

The results have also indicated that fewer attributes were used to discriminate between classes when OICB+ was employed, and that this effect was more pronounced in the data set with the most attributes. We also observed that the OICB+ method on several occasions favoured some of those attributes which are known to have predictive power in the relevant domain.

In Tables 5.13, 5.12 and 5.15 the experimental data shows that the proposed approaches resulted in smaller programs overall, where the best of run programs were discovered earlier in the evolutionary process.

To summarise research outcomes of the two studies on class boundary determination, we would argue that the empirical research in this chapter shows that the use of individualised decision boundaries and polarity settings may result in a more focused evolutionary process which offers several advantages over standard approaches:

1. Improved generalisation: reduced bias and variance error;
2. Smaller programs;
3. Reduced run times;

4. Better identification of predictive variables;
5. Improved interpretability of evolved solutions;
6. Competitive with other Machine Learning algorithms.

Another interesting implication of these first two studies, is that unlike symbolic regression problems, where ephemeral random constants (ERCs) may be needed for the GP system to evolve an acceptable target function, they may not be necessary for solving binary classification tasks using GP.

With reference to the research question of whether individualised self-adaptive genetic operators may have a role to play in improving generalisation, the beneficial results of an individualised approach in this empirical study are less pronounced. All of the self-adaptive methods performed better than the static population based method and the differences between both the OSA and TOSA methods and the static TGP approach are statistically significant at the 92% and 96% level respectively.

Regarding the current implementation of self-adaptive crossover and mutation, the results would seem to support the previous findings of Spears et al. [1992], Luke and Spector [1997], and White and Poulding [2009]. The experimental results concerning the application of a self-adaptive tournament size, where tournament strategy becomes increasingly more elitist as evolution progresses, are very encouraging.

Overall, the individualised boundary determination methods provided a reduction in both bias and variance error and the self-adaptive methods produced both superior training and test results in 4 of the 5 tasks, indicating a reduction in bias error when these methods were employed.

Chapter 6

Language Bias

In Chapter 1 we introduced the concept of *language bias* [Whigham, 1996], which when applied to GP refers to inductive bias associated with the function and terminal sets. In addressing our third core question, we choose to include parameters which control permissible program size under this heading also. This is a reasonable course of action because of the historical tight coupling of size and complexity in GP terminology around the topic of generalisation.

6.1 Size Versus Complexity

As previously discussed in Chapter 3, Section 3.2, the topics of bloat, size, complexity, over-fitting and generalisation have, for some time, been closely linked in GP research. In this chapter, we carry out a simple empirical study designed to discover what effects may be observed when program size and functional complexity are *varied in combination*.

6.1.1 Background

Chapter 3 contains an overview of some important previous research concerning size and complexity as they relate to generalisation in ML and GP. In studying some of this work it is sometimes difficult to differentiate the concepts, as several researchers seem to take the view that large programs are by definition complex, see for example Cavaretta and

Chellapilla [1999] and Gagné et al. [2006]. If this were the case, then an obvious approach to minimising complexity is to reduce program size, for example, by encouraging the evolution of smaller programs or by penalising larger ones, and as previously outlined Gagné et al. [2006] demonstrated that controlling program size with *parsimony pressure* lead to improvements in generalisation. However, in contrast, Soule and Foster [1998] showed that the application of parsimony pressure could have a negative effect on performance when the technique was applied to every individual in the population. However, the latter research focused on training data and was not concerned with generalisability.

Much of the previous work dealing with a possible relationship between *functional* complexity and generalisation has focused on symbolic regression problems [Castelli et al., 2011; Vanneschi et al., 2010b] where the term *functional complexity* usually refers to model/solution complexity. That is to say, the complexity aspect relates to the relative complexity of the function encoded in the solution. This is a point of view that may be particularly useful when considering symbolic regression tasks, but may not be quite so relevant to binary classification problems.

The previously mentioned (Chapter 3) *order of non-linearity* complexity measure proposed by Vladislavleva et al. [2009b] differentiated *size complexity* from *expressional complexity*. Similarly, for the current study, we define complexity in terms of the *complexity of the function set available to encode possible solutions*: the simplest function set contains operators such as addition and subtraction, whereas a more complex function set may contain trigonometric operators such as *sin* and *cos*. From now on, we refer to this concept as *operator complexity*.

6.1.2 Research Objectives

The objective of this study is to gain a better understanding of relationships which may exist between size, operator complexity and over-fitting, and in doing so, potentially learn rules of thumb to guide configuration of the relevant parameters. Specifically we address Core Question 3: "Is it possible to improve generalisation performance by implementing heuristics derived from a deeper understanding of the contributions of *program*

size and operator complexity to language bias?”

6.1.3 Experiments

Here we outline the various experiments carried out for this investigation. GP parameters are detailed in Table 6.1, experimental configurations are shown in Tables 6.3 and 6.2. For this study we use the BUPA, CAR, ECO, HS and ION datasets, as between them they offer good variety in the numbers of attributes and instances together with differing levels of class imbalance.

Table 6.1: GP Parameters for Size, Complexity Experiments

Parameter	Value
Strategy	Steady State
Initialisation	Ramped half-and-half
Selection	Tournament
Tournament Size	2
Crossover	70
Mutation	20
Reproduction	10
Initial Min Depth	Various
Initial Max Depth	Various
Max Depth	Various
Function Set	Various
Population	500
Max Gen	60

Crossover and mutation are applied with the stated percentages on the remainder of the population after reproduction has been applied.

6.1.4 Experimental Configurations

For this study we investigate the effects of varying both size and operator complexity with GP on a variety of binary classification tasks. We

use relative terms such as “tiny” and “huge”, which while they do not have an intrinsic scientific meaning, provide a description of the relative values of each, and are easily understood for discussion purposes. Table 6.2 details size constraints used for the experiments: parameter values for initialisation minimum depth, initialisation maximum depth and maximum depth permissible, in five different configurations. Obviously, there are numerous combinations of depth parameters which could be used. We have chosen these to provide a range of initial minimum, initial maximum and overall maximum allowable values.

Table 6.2: Size Complexity Experiments: Size Constraints

Configuration	Init Min Depth	Init Max Depth	Max Depth
TINY	1	2	8
SMALL	1	4	10
MED	2	6	16
LARGE	4	8	20
HUGE	8	16	32

Table 6.3 outlines four different function sets facilitating varying degrees of operator complexity, as the term is defined in Section 6.1.1

Table 6.3: Size Complexity Experiments: Operator Complexity Configurations

Configuration	Functions
MIN	+ -
LOW	* /
MID	+ - * /
HI	+ - * / sin cos

From now on, we refer to the various configurations either by the acronyms outlined in Tables 6.3 and 6.2 or by a conjunction of these, in size complexity order. For example TINYMIN uses the TINY size constraints combined with the least complex function set.

6.1.5 Results

In this section we report experimental results in two categories: the first subsection 6.1.5.1 outlines and discusses results for classification accuracy and over-fitting, and the second subsection 6.1.5.2 details results relating to program size. All results reported relate to the fifty best-of-run individuals for each configuration for each data set.

6.1.5.1 Training and Test Accuracy, AUC and Over-fitting

In this section we firstly report results for average and best % accuracy on training and test data, together with the average and best AUC score on test data. We also report the degree of over-fitting in each case. The measure of over-fitting used here is similar to that proposed in [Vanneschi et al., 2010b], except that we calculate a measure of over-fitting with regard to the training and test fitness for the *best-of-run individuals at the end of evolution*, also, we report negative over-fitting in the case where training fitness is lower than test fitness.

For this study, the most important measure is the AUC score for each configuration. However we are also interested in information regarding classification accuracy, variability and over-fitting.

BUPA Results

Looking at results for the BUPA data set in Table 6.4 we can see that the configuration which had the best overall average AUC was simplest function set used with the large size constraints(LARGEMIN). In fact, for each size configuration, the least complex function set produced the best average AUC result. For best overall individual AUC score, the TINYMID and SMALLMID configurations shared the top score.

The LOW complexity configuration (* /) produced very weak results for classification accuracy on both training and test data. Aside from that configuration, the high complexity setting resulted in the next worst performance on training data for each size category and it also exhibited both the greatest variability in accuracy and the highest degree of over-fitting. The MIN complexity settings did not exhibit any over-fitting,

while for the more complex MID and HI complexity settings over-fitting was present and tended to increase with size.

HS Results

The results for the Habermans's Survival data, shown in Table 6.5, illustrate how important it is to use a performance measure such as AUC for classification tasks even on relatively balanced data sets. Even though we have used *average* accuracy as our fitness measure, both the MID and HI complexity configurations had poor accuracy on the minority class which resulted in low average AUC scores in spite of apparently superior overall classification accuracy. This data set is relatively balanced, having a minority class percentage of 36%. As the minority class is by convention treated as the positive one, poor performance on this class will result in a low AUC score as AUC plots the true positives against false positives. The higher overall accuracy of the MID and HI configurations is a result of classifying most of the instances as belonging to the majority class.

The LOW complexity configuration resulted in very poor results for training and test accuracy on this data, delivering worse than 50% classification accuracy.

For this data set the best average AUC score is shared by a configuration which involve the least complex function set, combined with either the large or huge size constraints: LARGEMIN and HUGEMIN. Again, the most complex function set resulted in the greatest variability and most over-fitting. However the SMALLHI set-up had the best individual AUC score.

With regard to over-fitting, the simplest function set delivered test results that were actually better than the corresponding training results and tended to improve with size. In contrast, over-fitting increased as the allowed size increased for the more complex configurations.

ECO Results

The ECO data set is somewhat different to others used in this investigation, as it was originally a multi-class data set consisting of seven classes. We have collapsed six classes into one, resulting in an unbalanced binary

Table 6.4: Size Complexity Experiments: BUPA Training & Test Accuracy

		Training			Test					
Size	Complexity	Avg Accuracy	Best Accuracy	Std. Dev.	Avg Accuracy	Best Accuracy	Std. Dev.	% OverFit	AUC	Best AUC
Tiny	MIN	70.63	73.68	1.95	72.77	75.43	2.15	-2.14	0.77	0.80
	LOW	43.42	43.42	0	43.00	43.86	0.12	0.42	0.40	0.52
	MID	70.90	75.87	2.71	68.96	76.31	6.20	1.93	0.71	0.83
	HI	68.41	72.81	2.81	65.61	76.31	7.94	2.80	0.69	0.79
Small	MIN	72.69	74.12	0.96	73.75	76.31	1.37	-1.06	0.78	0.80
	LOW	43.45	43.85	0.10	43.00	43.86	0.12	0.42	0.40	0.53
	MID	71.53	75.43	2.30	70.15	78.07	5.76	1.38	0.72	0.83
	HI	70.33	75.43	2.88	67.73	77.19	7.00	2.60	0.70	0.81
Med	MIN	72.93	74.12	0.80	73.22	77.19	1.58	-0.28	0.78	0.80
	LOW	43.56	44.29	0.24	43.19	43.86	0.38	0.37	0.44	0.70
	MID	72.78	76.75	2.02	69.61	78.07	5.67	3.17	0.71	0.81
	HI	70.61	75.87	2.90	66.70	76.31	6.68	3.91	0.70	0.79
Large	MIN	73.12	74.56	0.72	73.71	77.19	1.53	-0.60	0.79	0.80
	LOW	43.79	44.30	0.28	43.31	43.85	0.55	0.41	0.47	0.74
	MID	72.40	77.19	2.02	68.57	76.31	5.37	3.82	0.68	0.81
	HI	70.60	75.43	2.62	66.52	77.19	5.94	4.07	0.69	0.79
Huge	MIN	73.03	74.56	1.32	74.01	78.07	1.48	-0.98	0.78	0.80
	LOW	46.58	69.29	6.19	44.84	60.52	4.45	1.74	0.40	0.71
	MID	72.25	78.07	2.40	66.70	77.19	5.65	5.66	0.67	0.80
	HI	69.72	75.87	2.50	57.19	78.07	8.90	12.53	0.59	0.77

Table 6.5: Size Complexity Experiments: HS Training & Test Accuracy.
Best results are in bold font.

Size	Complexity	Training			Test					
		Avg Accuracy	Best Accuracy	Std. Dev.	Avg Accuracy	Best Accuracy	Std. Dev.	% OverFit	AUC	Best AUC
Tiny	MIN	61.87	62.06	0.94	65.49	65.69	0.97	-3.61	0.71	0.72
	LOW	57.14	57.14	0	60.78	60.78	0	-3.64	0.62	0.63
	MID	72.28	75.76	1.90	75.25	78.43	3.03	-2.96	0.64	0.76
	HI	71.37	76.35	2.49	71.31	79.41	4.30	0.07	0.61	0.75
Small	MIN	62.10	62.10	0.28	64.04	65.69	0.14	-3.55	0.71	0.72
	LOW	57.14	57.14	0	60.78	60.78	0	-3.64	0.62	0.63
	MID	72.92	77.34	1.89	75.67	78.43	3.12	-2.74	0.62	0.75
	HI	72.21	78.32	2.74	71.98	78.43	4.15	0.23	0.61	0.77
Med	MIN	63.76	70.44	2.10	66.29	77.45	3.27	-2.53	0.72	0.73
	LOW	57.14	57.14	0	60.78	60.78	0	-3.64	0.61	0.63
	MID	73.65	77.34	2.02	74.66	78.43	2.40	-1.01	0.64	0.76
	HI	71.79	77.83	2.75	72.17	78.43	3.63	-0.38	0.65	0.76
Large	MIN	69.14	70.44	0.39	77.45	77.65	1.43	-8.30	0.73	0.73
	LOW	57.14	57.14	0	60.78	60.78	0	-3.64	0.61	0.63
	MID	74.01	78.32	1.99	74.45	78.43	2.30	-0.43	0.63	0.76
	HI	73.37	78.81	2.57	72.35	78.43	3.90	1.02	0.64	0.74
Huge	MIN	69.00	69.45	0.62	77.27	77.45	0.87	-8.27	0.73	0.73
	LOW	57.78	58.13	0.16	60.78	60.78	0	-3.60	0.52	0.62
	MID	75.45	78.82	1.62	72.90	78.43	2.56	3.32	0.62	0.74
	HI	73.27	80.09	3.88	66.58	77.45	6.28	6.68	0.61	0.75

classification data set. For almost half of the runs of the TINYMIN set-up the system consistently converged on the same seven node solution which had good accuracy on training data but very poor accuracy on test data.

In constructing the training and test sets, we maintained the same ratio of the two classes in each set, however we did not ensure that for the majority class, the same balance of the underlying, original six classes was in place. This may explain the high level of over-fitting and variability for this configuration. Aside from that set-up, the least complex configuration in general exhibited the least amount variability over-fitting.

The best average AUC result for this data set again resulted from one of the configurations which combined the simplest function set with either the large or huge set-up (LARGEMIN or HUGEMIN). The best single AUC score came from the TINYHI configuration.

Where there was over-fitting in the MIN complexity experiments, this tended to reduce as program size was allowed to increase, whereas for the more complex set-ups, particularly the MID configuration, *the opposite was true*.

ION Results

For the ION data set shown in Table 6.7, we note again that the best overall individual AUC result was achieved using the more complex function sets, where SMALLMID and LARGEHI both had a result of 0.97, which was considerably better than the average AUC produced by either configuration. The best average AUC resulted from the MEDMIN set-up and, as with several of the other data sets, increased operator complexity resulted in greater variability in training and test accuracy.

For this data set, some over-fitting was apparent for each experimental set-up, and this tended to increase as size was permitted to increase.

CAR Results

The CAR data set is a difficult one: it has a relatively large number of features (86) and instances (5822) and is quite imbalanced, having approximately 6% minority instances. The best average AUC score re-

Table 6.6: Size Complexity Experiments: ECO Training & Test Accuracy. Best results are in bold font.

		Training			Test					
Size	Complexity	Avg Accuracy	Best Accuracy	Std. Dev.	Avg Accuracy	Best Accuracy	Std. Dev.	% OverFit	AUC	Best AUC
Tiny	MIN	77.15	80.91	2.42	52.85	78.57	18.67	24.28	0.69	0.90
	LOW	90.06	90.45	0.15	88.36	88.40	0.17	1.70	0.17	0.90
	MID	82.04	90.00	2.90	74.30	83.93	10.91	7.94	0.79	0.90
	HI	83.85	89.09	2.92	74.86	83.92	10.91	8.90	0.76	0.93
Small	MIN	79.52	80.91	1.78	71.64	80.35	12.61	7.88	0.84	0.90
	LOW	90.19	90.45	0.23	88.25	88.39	0.33	1.94	50.45	0.90
	MID	84.64	90.91	3.49	76.05	84.82	10.81	8.58	0.75	0.90
	HI	84.77	91.81	3.00	74.35	84.82	14.24	10.40	0.75	0.90
Med	MIN	80.28	81.36	1.02	76.84	80.36	2.80	3.44	0.88	0.89
	LOW	71.46	90.45	29.82	68.14	88.39	31.40	3.32	0.56	0.91
	MID	87.56	91.82	2.46	77.42	84.82	8.24	10.14	0.73	0.89
	HI	86.08	92.27	2.72	75.14	84.82	10.94	9.77	0.74	0.90
Large	MIN	81.05	81.81	0.61	78.84	80.36	1.79	2.18	0.89	0.89
	LOW	79.18	80.90	1.06	79.34	82.19	1.53	-0.16	0.77	0.90
	MID	89.61	92.27	1.59	78.12	83.93	7.01	11.50	0.71	0.89
	HI	87.68	92.73	2.22	77.51	84.82	6.79	10.16	0.74	0.90
Huge	MIN	81.26	81.82	0.61	78.81	80.36	1.79	2.46	0.89	0.89
	LOW	82.61	88.63	2.76	71.73	82.14	7.49	10.88	0.65	0.87
	MID	90.01	93.18	1.83	76.46	83.92	6.82	13.55	0.72	0.89
	HI	88.95	93.18	2.23	77.91	84.82	5.07	11.04	0.73	0.89

Table 6.7: Size Complexity Experiments: ION Training & Test Accuracy. Best results are in bold font.

		Training			Test					
Size	Complexity	Avg Accuracy	Best Accuracy	Std. Dev.	Avg Accuracy	Best Accuracy	Std. Dev.	% OverFit	AUC	Best AUC
Tiny	MIN	87.30	89.65	1.38	82.31	91.37	3.42	4.99	0.81	0.92
	LOW	86.34	87.50	0.54	83.58	87.07	2.34	2.76	0.82	0.93
	MID	87.17	90.09	1.41	85.48	90.51	3.40	1.68	0.82	0.90
	HI	87.26	90.94	1.50	86.27	93.10	3.31	0.99	0.83	0.90
Small	MIN	88.78	90.95	0.97	83.79	92.24	3.25	4.99	0.83	0.92
	LOW	86.50	89.22	1.29	83.34	89.65	2.72	3.15	0.73	0.91
	MID	89.50	93.10	1.73	86.00	94.83	4.02	3.44	0.79	0.97
	HI	89.07	93.10	1.88	86.58	95.69	3.86	2.50	0.81	0.93
Med	MIN	89.81	91.38	1.00	82.84	87.93	2.82	6.96	0.84	0.90
	LOW	86.97	90.08	1.14	82.53	87.07	2.41	4.43	0.68	0.88
	MID	90.17	93.96	1.98	85.56	92.24	4.54	4.60	0.77	0.88
	HI	89.43	92.67	1.51	85.24	92.24	3.68	4.20	0.82	0.92
Large	MIN	90.50	92.67	0.86	83.50	91.37	2.41	6.99	0.82	0.91
	LOW	87.82	89.65	1.05	82.05	89.65	3.11	5.77	0.66	0.91
	MID	90.53	93.96	2.07	86.08	91.37	4.11	4.44	0.77	0.92
	HI	89.73	93.53	1.67	85.27	93.10	4.59	4.46	0.82	0.97
Huge	MIN	90.28	92.24	1.54	83.20	89.65	3.68	7.07	0.82	0.90
	LOW	88.12	92.24	1.46	83.08	89.65	3.45	5.03	0.67	0.83
	MID	88.01	93.10	4.45	81.67	92.24	8.21	6.34	0.73	0.89
	HI	89.60	93.10	1.97	83.06	90.51	4.50	6.53	0.82	0.94

sulted from a configuration using MIN complexity combined with either medium or large size constraints. Again, the simplest in terms of complexity had the best average AUC over all of the size configurations. Excluding the tiny configuration, the least complex function set exhibited the least variability in training and test accuracy. The best overall AUC result of 0.76 was shared between MIN combined with Large and High combined with Huge. In the latter case, the difference in the best score of 0.76 and the average of 0.65 illustrates the degree of variability in performance when solutions using the most complex function set were applied to the test data.

Overall, there was no evidence of over-fitting on this data set.

6.1.5.2 Size

In this section we examine size data for each configuration and each data set. Tables 6.9 to 6.13 provide information on population average size and standard deviation at the final generation, average size of best-of-run trained individual and size of the individual with the best AUC score on test data.

As previously discussed in Section 6.1.1, the effective size of a GP program is the size of the effective code, in nodes, where effective code is code which is executed at least once [Rosca, 1996]. As there are no lazy functions in any of the function sets used for this study, the entire program tree of each individual can be considered *effective*, and the $effectivesize = programsiz$ e for all individuals.

BUPA Size Data

Looking at the size data for the BUPA data set, we can see in Table 6.9, that with the exception of the TINY set-up, the MIN complexity setting resulted in the largest population average program size. This is not surprising, as semantics that are encoded using addition and subtraction operators will naturally result in a bigger expression than if multiplication and division are employed. For the HUGEMIN configuration the population average program size was appreciably larger than any of the others,

Table 6.8: Size Complexity Experiments: CAR Training & Test Accuracy. Best results are in bold font.

		Training			Test					
Size	Complexity	Avg Accuracy	Best Accuracy	Std. Dev.	Avg Accuracy	Best Accuracy	Std. Dev.	% OverFit	AUC	Best AUC
Tiny	MIN	62.99	74.46	6.3	64.56	76.30	6.47	-1.58	0.67	0.74
	LOW	70.91	73.54	3.54	71.70	74.19	3.40	-0.79	0.46	0.63
	MID	64.31	78.27	6.15	65.41	78.97	6.46	-1.10	0.62	0.74
	HI	63.35	72.61	5.06	64.63	73.31	5.15	-1.27	0.59	0.71
Small	MIN	65.08	71.94	3.85	66.58	73.57	3.82	-1.49	0.70	0.74
	LOW	69.78	73.18	3.88	70.58	73.67	3.80	-0.79	0.47	0.60
	MID	65.01	72.66	3.62	65.56	74.70	4.01	-0.54	0.63	0.72
	HI	64.26	77.24	4.94	65.04	77.69	5.06	-0.77	0.62	0.74
Med	MIN	67.95	73.33	2.85	68.62	74.81	2.61	-0.67	0.73	0.75
	LOW	66.71	75.21	3.61	67.04	76.40	3.61	-0.33	0.52	0.60
	MID	67.74	76.96	3.20	67.97	77.22	3.20	-0.23	0.63	0.74
	HI	66.57	79.41	4.04	67.11	77.79	3.84	-0.54	0.62	0.73
Large	MIN	67.35	71.81	3.01	68.09	72.69	2.76	-0.74	0.73	0.76
	LOW	65.90	72.40	3.36	66.19	72.38	3.32	-0.29	0.51	0.60
	MID	67.28	72.97	3.64	67.80	72.85	3.43	-0.52	0.66	0.75
	HI	66.57	74.88	3.62	67.33	74.24	3.09	-0.76	0.63	0.73
Huge	MIN	67.05	75.52	2.85	67.74	76.20	2.74	-0.68	0.72	0.75
	LOW	65.55	73.56	4.26	65.79	74.09	4.27	-0.23	0.50	0.62
	MID	66.34	77.32	3.63	66.57	76.98	3.62	-0.23	0.66	0.73
	HI	64.76	72.35	3.34	65.73	71.61	3.07	-0.97	0.65	0.76

If we compare the average size of the best-of-run individuals with the population average program size, we can see that, in general, these are broadly similar. In contrast, the size of the individual with the best AUC score *on test data*, is generally much smaller than the population average. In the case of the HUGEMIN configuration, for example, the population average is 19496 nodes, whereas the best individual on test data has 4399 nodes. This does not hold true for the TINY and SMALL size constraints, and a possible explanation for this is that the average population size may not be quite big enough to encode a satisfactory solution.

ECO Size Data

Turning our attention to the ECO size data as seen in Table 6.10, we observe a similar pattern to the previous data set: MIN complexity configurations result in larger programs, which in the case of HUGEMIN are disproportionately larger compared with the other size configurations; with some exceptions, the program with the best individual AUC score was smaller than the population average.

HS Size Data

The size experimental results for the HS data set (shown in Table 6.11) do not follow the same pattern which we observed in the BUPA and ECO data sets. For several of the configurations, the MIN complexity set-up resulted in smaller programs, and the sizes of the best AUC scoring programs were not in general smaller than the population average. On the other hand, the average size of the best-of-run trained individuals were broadly in line with the population average - which is similar behaviour to the other data sets.

ION Size Data

For the Ionosphere data in Table 6.12, again the more complex solutions are smaller on average than the MIN complexity ones. In this case, there is also more variability in size in the MIN complexity solutions. For the larger configurations, the best AUC solutions are considerably smaller than the population average.

Table 6.9: Size Complexity Experiments: BUPA Size Data

Size	Complexity	Pop Avg Size	Std Dev	BOR Avg Size	Best AUC Size
Tiny	MIN	11	5.44	18	33
	LOW	33	9.08	4	3
	MID	40	11.09	37	53
	HI	33	9.08	34	57
Small	MIN	177	40.61	169	345
	LOW	50	13.70	9	15
	MID	65	17.17	55	71
	HI	50	13.70	48	50
Med	MIN	203	46.9	1175	63
	LOW	85	26.40	52	41
	MID	154	37.01	149	223
	HI	85	26.40	83	46
Large	MIN	532	111.44	445	127
	LOW	169	48.44	227	229
	MID	317	77.49	306	51
	HI	169	48.44	169	258
Huge	MIN	19496	1058.95	13626	4399
	LOW	1129	266.22	19104	499
	MID	1158	199.00	1142	663
	HI	1129	266.22	1042	334

Table 6.10: Size Complexity Experiments: ECO Size Data

Size	Complexity	Pop Avg Size	Std Dev	BOR Avg Size	Best AUC Size
Tiny	MIN	17	7.49	14	30
	LOW	32	10.35	29	19
	MID	30	11.28	32	39
	HI	32	10.35	33	26
Small	MIN	38	14.85	27	17
	LOW	50	15.35	7	15
	MID	55	16.74	46	71
	HI	50	15.35	50	23
Med	MIN	157	46.04	80	35
	LOW	107	30.73	53	57
	MID	67	39.98	150	209
	HI	107	30.73	107	46
Large	MIN	502	113.78	289	347
	LOW	226	58.05	291	349
	MID	422	87.66	378	219
	HI	226	58.05	205	32
Huge	MIN	26940	1366.86	17292	3999
	LOW	713	161.16	13744	3095
	MID	1173	269.75	1035	815
	HI	713	161.61	658	494

Table 6.11: Size Complexity Experiments: HS Size Data

Size	Complexity	Pop Avg Size	Std Dev	BOR Avg Size	Best AUC Size
Tiny	MIN	2	3.24	13	15
	LOW	47	10.72	2	7
	MID	55	12.66	54	71
	HI	47	10.72	48	52
Small	MIN	3	3.33	13	15
	LOW	62	14.37	4	7
	MID	80	19.33	74	89
	HI	62	14.37	55	56
Med	MIN	19	7.42	53	221
	LOW	120	29.68	16	63
	MID	176	40.28	171	329
	HI	120	29.68	109	72
Large	MIN	480	112.40	538	441
	LOW	194	48.65	55	49
	MID	362	75.55	332	337
	HI	194	48.65	182	352
Huge	MIN	29523	2137.27	26359	65757
	LOW	968	179.27	10166	20657
	MID	3038	436.12	2912	615
	HI	968	179.27	901	469

An interesting aspect of these particular results is the population average size of the HUGEMIN experiments, relative to the other data sets previously discussed. For the BUPA, ECO and HS data sets the average size for this configuration is 19496, 26940 and 29523 respectively, whereas for this ION data set the average size is 5533. This is surprising in and of itself, but especially so when we consider that this data set has considerably more attributes (34) than those others (4 – 7). The number of instances for the ION data set is similar to the BUPA ECO and HS data sets. Looking at the underlying run data, we observe that the very large average values mentioned here are a result of a small number of extremely large outliers.

CAR Size Data

Finally, looking at the size results for the CAR data set in Table 6.13, we see that the behaviour is very similar to that of the ION data: again the HUGEMIN population average size, while large compared to the other configurations for this data set, is small relative to the BUPA ECO and HS data sets. Again, the CAR data set has a relatively large number of attributes (86) and it also has a lot more instances than the other data sets.

6.2 Research Outcomes

This simple study, delivers some interesting results regarding size and functional (operator) complexity in GP for binary classification tasks. For the problems studied, the following observations can be made from a study of the experimental results:

1. The simplest configuration in terms of complexity (MIN) consistently results in the *best average AUC* score and in many cases the result is appreciably better. Best results are achieved when this MIN complexity set-up is combined with a MED LARGE or HUGE potential size.

Table 6.12: Size Complexity Experiments: ION Size Data

Size	Complexity	Pop Avg Size	Std Dev	BOR Avg Size	Best AUC Size
Tiny	MIN	23	7.12	27	41
	LOW	16	6.73	10	11
	MID	15	6.13	18	41
	HI	16	6.73	19	15
Small	MIN	81	17.50	81	71
	LOW	39	12.84	49	21
	MID	49	15.36	50	61
	HI	39	12.83	37	49
Med	MIN	236	48.10	224	281
	LOW	85	26.66	126	21
	MID	123	36.19	117	23
	HI	85	26.66	80	84
Large	MIN	551	94.78	507	399
	LOW	172	46.28	307	47
	MID	209	62.59	187	175
	HI	142	40.28	133	73
Huge	MIN	5563	716.63	3276	1003
	LOW	332	109.90	846	875
	MID	1409	462.12	1726	785
	HI	332	109.90	314	236

Table 6.13: Size Complexity Experiments: CAR Size Data

Size	Complexity	Pop Avg Size	Std Dev	BOR Avg Size	Best AUC Size
Tiny	MIN	15	6.68	17	21
	LOW	16	7.02	18	15
	MID	16	7.0	19	21
	HI	16	7.03	15	11
Small	MIN	21	9.78	24	57
	LOW	27	11.52	27	11
	MID	32	12.12	35	19
	HI	27	11.52	30	17
Med	MIN	180	44.26	167	247
	LOW	71.09	26.09	114	147
	MID	119	36.72	110	95
	HI	71	26.09	74	40
Large	MIN	518	100.41	473	475
	LOW	131	41.30	230	151
	MID	239	65.22	230	263
	HI	131	41.30	115	84
Huge	MIN	4714	395.26	3871	1745
	LOW	363	108.39	584	391
	MID	607	175.05	584	821
	HI	363	108.39	334	277

2. The simplest configuration *almost always* produces programs with the least variability *regardless of size*;
3. Best *overall AUC* may be produced across the range of permissible sizes and was in each case achieved with either a MID or HI complexity setting;
4. In general, greater complexity results in greater variability;
5. In four of the five data sets, the MIN complexity configurations had negative or no over-fitting;
6. In four of the five data sets, the more complex set-ups showed evidence of over-fitting, and this *increased as size was permitted to increase*;
7. With some explainable exceptions, the size of the best overall individual with respect to AUC on test data was *significantly smaller* than the population average size whereas the size of the best trained individual in terms of classification accuracy was broadly in line with the average population size.

In Chapter 3 we asked the question: is it the case that small, simple solutions generalise better, or that solutions which generalise well will tend to be simple and smaller than average? The results of this investigation would seem to suggest that the latter is closest to the truth: given that the MED LARGE or HUGE configurations delivered the best average AUC where the size of the individual with the best overall AUC score was in many cases smaller than the population average.

We can also make several other, possibly less important observations from the results:

- Given identical size constraints, the least complex set-ups produce the smallest programs when combined with the tiny or small size configurations;
- The most complex configuration *always* produces the smallest programs when the system is set-up to use the larger size configurations;

- The two data sets with the highest numbers of attributes had significantly smaller population average program size for the HUGEMIN configuration, relative to the problems with far fewer attributes.

6.3 Conclusions

Experimental results from this study which lead to the observations outlined in 6.2 provide a deeper understanding of the contributions of *program size* and *operator complexity* to language bias and also to variance error. Against this background we can respond to Core Question 3 by tentatively suggesting the following heuristics:

1. A simple function set consisting only of addition and subtraction may be suitable if consistent results are required, where system performance on training data is likely to provide a reliable indication of possible test performance.
2. A more complex function set may be more appropriate if we are seeking to evolve the best individual possible, and are prepared to tolerate high variability together with some over-fitting.
3. Solutions which generalise well may tend to be relatively small and simple but the evolution of these solutions may be more successful in an environment which facilitates medium to large programs.

Of course these remarks, together with earlier observations, may apply only to the data sets investigated combined with the GP parameters applied in this particular study. However with regard to the efficacy of a reduced function set for binary classification, we have carried out some additional experiments using images of various textures, and the results of these experiments indicate that a simple function set can be very effective in that domain also. These results can be seen in Appendix D

Previously we defined over-fitting as being equivalent to the variance component of generalisation error, which we in turn defined as being the difference between training and test fitness/accuracy. In the GP community it is common practice to use the terms over-fitting and generalisation

almost inter-changeably, which may add to the lack of clarity around the topics.

Some of the results in this study demonstrate clearly that it is not all about over-fitting. For example, in the CAR data set there is no evidence of over-fitting, yet there is significant variation in average AUC score across configurations. Also, in the ION experiments, several of the configurations which had the most over-fitting also delivered the best average AUC score. As we know that generalisation error is composed of bias and variance error, it is reasonable to conclude that either in the absence of or in spite of any variance error (over-fitting), there are language biases Whigham [1996] inherent in parameter choices relating to size and operator complexity which may have a significant influence on the generalisation capability of GP for binary classification tasks.

As to why it is that the set ups with the MIN complexity option had the least over-fitting, intuition tell us that this is not surprising, as the evolved solutions are by definition simple and thus less likely to closely fit any idiosyncrasies in the data. However, this lack of over-fitting may also be related to the low variation in the quality of the solutions generated using the simple function set. This view is consistent with that expressed by Elkan [2001]: “the danger of over-fitting is not reduced just because a learning algorithm outputs a model with only a few parameters. For the risk of over-fitting to be lessened, all models that the algorithm *could have* output must be equally simple”.

Chapter 7

Selection Bias

This chapter addresses the fourth core question which concerns *selection bias* where selection bias refers to the method used to select one hypothesis over another. In GP this relates to the chosen fitness function, together with aspects such as tournament size and replacement strategy. In this chapter we identify further sources of inappropriate bias, specifically, we focus on inappropriate bias in the standard choice of fitness function and supporting training methodology. To mitigate the effects of these undesirable biases, we propose an approach which replaces these biases with more appropriate inductive biases.

In the second section of this chapter we carry out an empirical study of the effects on generalisation of selection bias which may arise from different choices of replacement strategy and tournament size. We compare outcomes of generational and steady state strategies combined with a range of tournament sizes from 2 to 15, with and without elitism.

7.1 The Class Imbalance Problem

Each day 2.5 *quintillion* bytes of data are created. This is a relatively recent phenomenon, such that 90% of the data in the world today has been created in the last two years alone [Zikopoulos et al., 2011]. This explosion in data offers tremendous opportunities for knowledge acquisition and decision support, but the potential for unleashing the power of these insights is balanced by several complex challenges. Aside from the problem of handling the sheer volume of data, there is the challenge

of identifying those instances which may be interesting or useful, in an environment where such items may be in the minority. From a machine learning perspective, at its simplest, this can be viewed as a binary classification problem.

In binary classification tasks, the class imbalance problem arises where there is a disparity in the number of instances of each class in a particular dataset. Greater disparity makes classification tasks more difficult, as there is an inherent bias towards the class which has greater representation in the dataset. When a machine learning algorithm, designed for general classification tasks, is confronted with significant imbalance, the “intelligent” thing for it to do is to classify all instances as belonging to the majority class. Ironically, it is frequently the case that the minority class is the one which contains the most important or interesting instances. In datasets from the medical domain, for example, it is generally the case that instances which represent malignancy or disease are far fewer than those which do not. Thus, in Machine Learning classification tasks, the *class imbalance problem* is an important one which has received a lot of attention in the last few years.

7.1.1 Research Objective

This chapter is concerned with addressing the fourth core question: Can new sampling and algorithmic methods designed to mitigate the *class imbalance problem*, which is influenced by *inappropriate selection bias*, deliver improvements in generalisation?

A variety of strategies have been applied to the imbalance problem over the years, with varying degrees of success. Typically previous approaches have involved attacking the problem either algorithmically or by manipulating the data in order to mitigate the imbalance. We propose a hybrid approach which combines a technique which we call Proportional Individualised Random Sampling (PIRS) with two different fitness functions, each of which is designed to improve performance on imbalanced classification problems in GP. We investigate the efficacy of the proposed methods together with that of five different algorithmic GP solutions, two of which are taken from the recent literature. We conclude that

the PIRS approach combined with either average accuracy or Matthews Correlation Co-efficient, delivers superior results in terms of AUC score when applied to either balanced or imbalanced datasets.

The way in which GP, similar to other EC algorithms, evolves a population of individuals over time facilitates a very flexible and potentially granular approach for tackling this type of problem, in that the practitioner can influence the process both at the individual and the population level, as well as on a generational basis. We have chosen to investigate a hybrid approach which seeks to influence the learning process at both the individual and population levels, using a strategy which combines sampling and algorithmic techniques. In this work, we propose a new sampling technique which we call *Proportional Individualised Random Sampling* which we combine with Matthews Correlation Co-efficient and balanced accuracy.

7.1.2 Previous Work on Class Imbalance

The class imbalance problem is an important one which has generated a lot of interest in the research community in recent years. In general, approaches can be divided between those which tackle the imbalance at the data level, and those which seek an algorithmic solution. There have also been several hybrid techniques proposed which combine aspects of the other two.

Methods which operate on the data try to repair the imbalance by creating more balanced datasets for training purposes. This is done by under-sampling the majority class or over-sampling the minority class, where the former involves removing some examples of the majority class and the latter is accomplished by adding duplicate copies of minority instances until some predefined measure of balance is achieved. Over or under-sampling may be random in nature [Batista et al., 2004] or “informed” [Kubat et al., 1997], where in the latter, various criteria are used to determine which instances from the majority class should be discarded. An interesting approach called SMOTE (Synthetic Minority Oversampling Technique) was suggested by Chawla et al. [20] in which rather than over sampling the minority class with replacement they gen-

Table 7.1: GP Parameters for Class Imbalance Experiments

Parameter	Value
Strategy	Generational
Initialisation	Ramped half-and-half
Selection	Tournament
Tournament Size	2
Crossover	80
Mutation	20
Initial Min Depth	1
Initial Max Depth	6
Max Depth	17
Function Set	+ - * /
ERC	-5 to +5
Population	500
Max Gen	60

erated new synthetic examples.

At the algorithmic level Joshi et al. [2001] modified the well-known AdaBoost [Freund and Schapire, 1996] algorithm so that different weights were applied for boosting instances of each class. Akbani et al. [2004] modified the kernel function in a Support Vector Machine implementation to use an adjusted decision threshold. Class imbalance tasks are closely related to cost based learning problems, where misclassification costs are not the same for both classes. Alfaro-Cid et al. [2007] successfully employed a cost based fitness function in GP for bankruptcy prediction, on highly unbalanced data.

Adacost [Fan et al., 1999] and MetaCost [Domingos, 1999a] are popular examples of the cost based approach. See [2004; He and Garcia, 2009; Kotsiantis et al., 2006a] for detailed reviews of these and various other approaches found in the literature.

7.1.2.1 GP

In the field of GP, much of the work on algorithmic approaches has been undertaken by Bhowan et al. [2009a, 2011, 2009b] in which they have studied the efficacy of a wide range of different fitness functions

on various imbalanced data sets. In this work, we compare with two of those methods: *Correlation Ratio* based fitness, and *Geometric Mean* based fitness, with which the researchers reported good results. These are described in Sections 7.1.4 and 7.1.4. In other work Patterson and Zhang [2007] investigated the use of average accuracy as a fitness function and also a modified version which used the squares of the individual accuracies for each class. Both methods resulted in improved performance on the minority class and a more balanced classification overall.

With regard to sampling approaches in GP, Hunt et al. [2011] examined several different sampling approaches including under sampling, over sampling and a combined approach. In each case they maintained equal numbers of instances from both classes in their training set and sampled the majority class with replacement. While they found that the various sampling approaches improved the classification accuracy on the minority class, performance on the majority class decreased. Overall, they reported that the method was not as successful as algorithmic approaches previously suggested by Bhowan et al. [2009a].

In other work, Doucette and Heywood [2008] proposed a *Simple Active Learning Heuristic (SALH)*: a hybrid approach which combined a simplified version of the Random Subset Selection algorithm proposed by Gathercole and Ross [1994], together with a modified Wilcoxon-Mann-Whitney statistic. They reported that their hybrid approach compared favourably with several other machine learning algorithms.

7.1.3 A Hybrid Approach

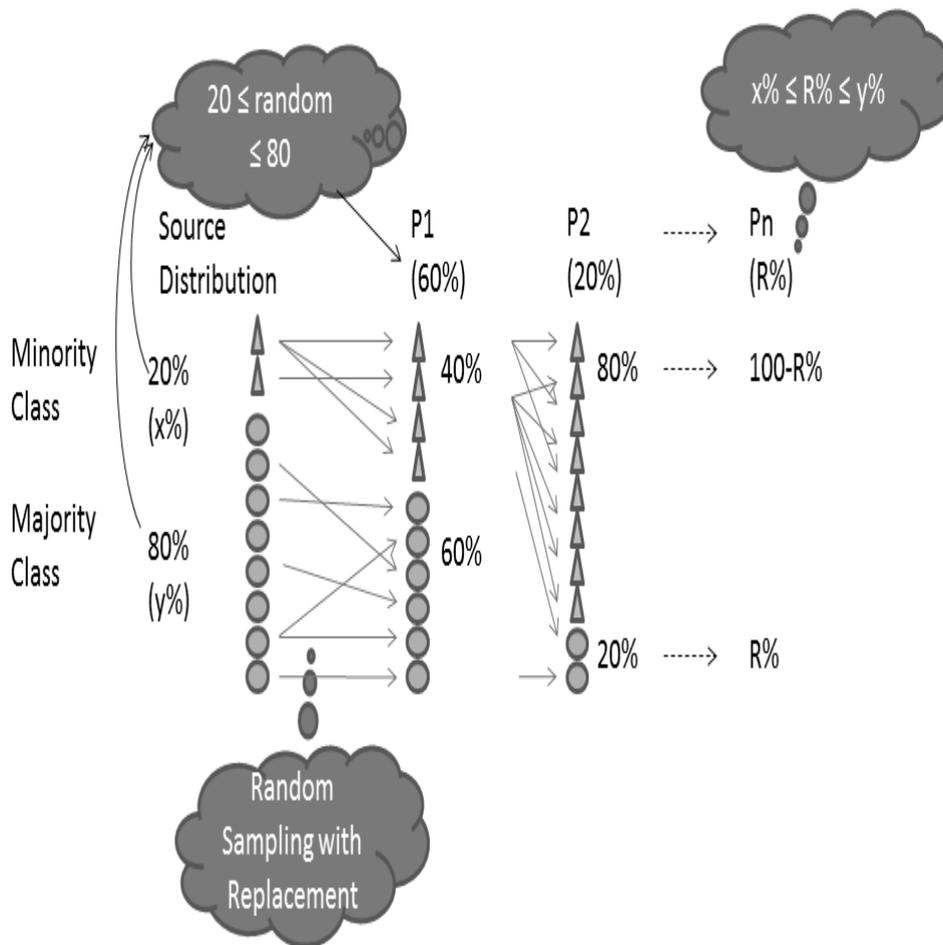
There are several disadvantages associated with the use of over- or under-sampling strategies for tackling the class imbalance problem. The obvious disadvantage with under-sampling is that it discards potentially useful data. The main drawback with standard oversampling is that it introduces exact copies of minority instances which may increase the potential for over-fitting. Also, the use of over-sampling increases the size of the dataset, thus adding to the computational cost. Here we propose a sampling approach which we call *Proportional Individualised Random Sampling (PIRS)* which either eliminates or mitigates these disadvantages.

Firstly, the size of the dataset is exactly the same as the original, so there is no additional computational cost, as is generally the case with random over-sampling. Instead, in a new sampling technique, we vary the number of instances of each class maintaining the original size of the dataset. At each generation and for *each individual* in the population the percentage of majority instances is randomly selected in the range between the percentages of minority (positive) and majority (negative) instances in the original distribution. Then, *that particular individual* is trained on *that percentage of majority instances* with instances of the minority class making up the remainder of the data. In both cases, each instance is randomly selected *with replacement*.

For example, in the case of the Yeast1.5 dataset, where 98.5% of the data makes up the majority class and 1.5% the minority, the training data for a given individual will be divided n percent majority instances where $1.5 \leq n \leq 98.5$ and m percent minority instances, where $m = 100 - n$. In this way, individuals within the population are trained with different distributions of the data within the *range of the original distribution*. From a holistic viewpoint, this approach combines the advantages of over- and under-sampling while avoiding the disadvantages of either method.

The benefit of this approach from the under sampling perspective is that while the majority class may not be fully represented at the level of the individual, all of the data for that class is available to *the population* as a whole. Because all of the available knowledge is spread across the population the system is less likely to suffer from the loss of useful data that is normally associated with under-sampling techniques. From the under-sampling viewpoint, over-fitting may be less likely as the distribution of instances of each class is varied for each individual at every generation. Also, as all sampling is done with replacement, there will be duplicates of negative as well as positive instances.

A useful advantage of our proposed approach is that it is equally applicable to both balanced and unbalanced datasets. Previous work of Liu and Khoshgoftaar [2004] has shown that, aside from the consideration of balance in the distribution of instances, the use of random sampling techniques may have a beneficial effect in reducing over-fitting.



Proportional Random Individualised Sampling Method Example

Figure 7.1: Proportional Individualised Random Sampling example for a source distribution with 20% minority instances. P_1 - P_n are programs in the population trained on potentially different input samples where the percentage of majority instances for that individual is in the range 20-80%

Thus, we believe that the proposed sampling approach can offer improved performance on a wide range of binary classification tasks, whether a particular dataset is balanced or not. This important proposition was simply addressed by Provost [2000] in the invited paper for the AAAI 2000 Workshop on Imbalanced Data Sets ..“isn’t the best research strategy to concentrate on how machine learning algorithms can deal most effectively with *whatever* data they are given?”.

We combine the PIRS sampling technique with two different fitness functions which are designed to function well with either balanced or unbalanced data: *Average Accuracy* and *Matthews Correlation Co-efficient*. In Machine Learning, Matthews Correlation Co-efficient is widely regarded as a good measure for evaluating the performance of a given model on binary classification tasks, in part because it has fewer inherent biases than some other popular methods [Powers, 2011]. But also, because it is considered suitable for both balanced and imbalanced data sets. Here, rather than using the measure to assess the performance of our model, we investigate its use in the actual evolution of the model by incorporating it as a fitness function as described in Equation 7.1.

$$MCC(P) = \frac{(TP * TN - FP * FN)}{\sqrt{(TP + FP)(TP + TN)(FP + FN)(TN + FN)}} \quad (7.1)$$

MCC is regarded as a balanced measure of the quality of a binary classifier, which can be used even if the classes are of different sizes. It is, in essence, a correlation co-efficient between the observed and predicted binary classifications. MCC returns a value between -1 and $+1$: where a value of $+1$ represents a perfect prediction, a value of 0 no better than random and a value of -1 represents total disagreement between predicted and observed class labels.

In addition to investigating the use of PIRS with Matthews Correlation Co-efficient, we also study the combination of PIRS and *average accuracy* also known as *balanced accuracy* which is a well know performance measure used in classification. This method modifies the calculation for overall accuracy to better emphasise the performance on *each*

class as shown in Equation 7.2.

$$AVGA(P) = 0.5 * \left(\frac{TP}{(TP + FN)} + \frac{TN}{(TN + FP)} \right) \quad (7.2)$$

7.1.4 Experimental Configurations

For the purpose of discussing class imbalance, we are interested in the *majority* and *minority* classes, where the majority class corresponds to the negative class and the minority class to the positive class. TP represents the number of minority class instances correctly classified, TN the number of majority class instances correctly classified, FP the number of majority class instances which have been incorrectly classified as belonging to the minority class, and FN the number of minority class instances which have been mis-classified as majority class instances. In describing the various experimental configurations below, we adhere to the standard nomenclature for clarity.

Standard GP (StdGP)

The fitness measure used for the standard GP configuration is a commonly used measure of “overall” classification accuracy. If a program P correctly classifies all instances, its overall accuracy will be 1. The fitness function for each program P is $1 - Accuracy(P)$, where $Accuracy(P)$ is as described in Equation 7.3.

$$Accuracy(P) = \frac{TP + TN}{TP + TN + FP + FN} \quad (7.3)$$

Standard GP with *Average Accuracy* (AVGA)

For the second configuration, we use a slight modification of the overall accuracy, which aims to maximise the average of the accuracy over both classes. The fitness function to be minimised is $1 - AVGA(P)$ where $AVGA(P)$ is described by Equation 7.2.

GP with Matthews Correlation Co-efficient (MCC)

For this configuration we employ a standard GP implementation with the

Matthews Correlation Co-efficient as the fitness function. This fitness function is described in Section 7.1.3 and Equation 7.1.

GP with Correlation Based Fitness (CORR)

Bhowan et al. [2009b] proposed a correlation ratio fitness measure to mitigate bias introduced by class imbalance for image classification problems. In this method the correlation ratio is used to measure how well the outputs of a GP Individual for the minority and majority classes are *separated* with respect to each other. The higher the correlation ratio achieved by a particular model, the better the classification performance. This fitness function is aimed at evolving solutions that perform equally well on both classes with the minimum loss to the overall classification rate. The correlation ratio "r" (generalised for M classes) is described in Equation 7.4.

$$r(P) = \sqrt{\frac{\sum_{c=1}^M N_c (\bar{\mu}_c - \bar{\mu})^2}{\sum_{c=1}^M \sum_{i=1}^{N_c} (P_{ci} - \bar{\mu})^2}} \quad (7.4)$$

Where $\bar{\mu}_c$ is the mean of the outputs of the program for instances of class c , $\bar{\mu}$ is the mean of the program outputs over all classes, M is the number of classes, N is the number of total instances, N_c is the number of examples of class c , and P_{ci} represent the output of a genetic program classifier P when evaluated on the i 'th example belonging to class c . This equation returns a value between 0 and 1, where values closer to 1 indicated better separability.

The researchers also imposed an identity function to guide the evolution such that outputs for instances of the majority class would be greater than zero, and outputs for instances of the minority class would be less than zero. Their final fitness function is shown in Equation 7.5

$$Correlation(P) = r + I(\bar{\mu}_{minority}, \bar{\mu}_{majority}) \quad (7.5)$$

Where the indicator function, I returns 1 if the mean of the minority and majority observations are positive and negative respectively, and 0 otherwise. Thus, the final fitness function returns a value between 0 and 2 where values closer to 2 represent good fitness, and those nearer to 0,

poor fitness.

GP with Geometric Mean based Fitness (GMF)

In other work, Bhowan et al. [2009a] proposed a fitness function using a *geometric mean* as shown in Equation 7.6.

$$GMF(P) = \sqrt{\frac{TP}{TP + FN} * \frac{TN}{TN + FP}} \quad (7.6)$$

This function has the property that if the number of instances of either class correctly classified is zero, then the geometric mean itself will also be zero, which has the effect of penalising individuals which perform badly on one or other class.

Proportional Individualised Random Sampling with Balanced Fitness Function (PIRS-BAL)

In this configuration, we employ the balanced fitness function defined in Equation 7.2. But we also randomly select training instances to train each individual. The data is randomly selected *with replacement*, varying the proportions of minority and majority class instances. The detail of our sampling technique is as previously described in Section 7.1.3.

Proportional Individualised Random Sampling with MCC (PIRS-MCC)

In this final experimental configuration we investigate Proportional Individualised Random Sampling (PIRS) together with Matthews Correlation Co-efficient: an aggregate objective function which represents a particular confusion matrix as a single value. For the PIRS-MCC configuration, we minimise $1 - MCC(P)$ where $MCC(P)$ is as previously outlined in Equation 7.1. Here again, the sampling method is as described in Section 7.1.3.

7.1.5 GP Parameters

The Genetic Programming parameters used for this investigation are as described in Table 7.1 and The datasets used are detailed in Table 7.2.

Table 7.2: Class Imbalance Data Sets

Data Set	Acronym	Features	Instances	%Minority
Bupa Liver Disorders	BUPA42	7	345	42
Habermans Survival	HS36	4	306	36
Yeast	Yeast16	8	1484	16
Yeast(1)	Yeast1.5	8	1484	1.5
Ecoli	Ecoli10	7	332	10

The yeast and ecoli datasets were originally multi-class datasets. In order to experiment with various levels of class imbalance, we have “collapsed” several of the classes into one to create binary classification tasks. The acronym used for each dataset indicates the % of the minority class in each dataset. In each case we have used two thirds of the available data from training and the remaining one third for test. We undertook 50 runs for each configuration, on each dataset, using identical random seeds for each set of 50 runs.

7.1.6 Results and Discussion

For this investigation we have chosen the Area Under the Receiver Operating Curve (AUC) as the primary measure of classification performance. As previously discussed in 2.2.1.5, values for this measure are calculated using the equivalent [Yan et al., 2003] Wilcoxon-Mann-Whitney statistic. We are also interested in the overall classification accuracy (particularly on test data), performance on the minority and majority classes, the sizes of the evolved classifiers and how early or late in the evolutionary process the best-of-run individual is discovered.

In the following subsections, we detail for each dataset investigated, run statistics for the best of run individuals; the AUC measure, average overall %accuracy on training and test data, best individual %accuracy for training and test data, average %error on the minority and majority classes for both training and test data, the average size in nodes and the average generation in which the best-of-run individual emerged.

To gain a clearer insight as to which method performed best *overall* we, once again, carried out the non-parametric Friedman test which is

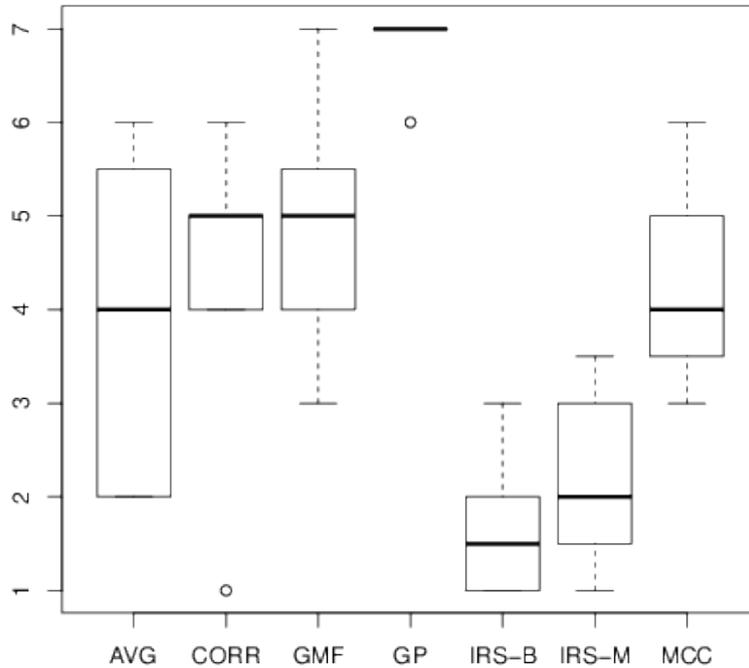


Figure 7.2: Class Imbalance Experiments: Methods ranked from 1 to 7 based on average AUC, where 1 is best and 7 is worst.

regarded as a suitable test for the empirical comparison of the performance of different algorithms [Gerevini et al., 2005]. This resulted in a p-value of 0.003 and indicated that the best performing algorithm in terms of AUC score was PIRS-BAL closely followed by PIRS-MCC as shown in Figure 7.2.

Results for the BUPA dataset shown in Table 7.3 illustrate that the stdGP method which uses the overall accuracy fitness measure performs very poorly on the minority class. The best approach overall is the PIRS-BAL method which combines PIRS with *average* accuracy. This method delivered a superior AUC measure of 0.80, produced the smallest programs where the best of run individual was discovered earliest in the evolutionary process. It also exhibits an absence of over-fitting, where the average test performance for both the minority and majority classes

were actually *better* than the training results.

Looking at the Ecoli10 results we can see that both methods which employed PIRS achieved good AUC scores and performed very well on the minority class, having several runs with perfect classification in the training phase. The GMF and AVGA approaches also achieved good training scores on the minority class, but these did not translate into good test results.

For the HS36 task, once again both PIRS methods produced the best AUC scores, the best minority performance and smallest programs. Again these programs were discovered earlier in the evolutionary process.

For the Yeast16 dataset, the results in Table 7.3 show that the CORR fitness function resulted in the best AUC score of 0.83. This method delivered the best accuracy on the minority class and the results were balanced across both classes. PIRS-BAL PIRS-MCC and MCC each had AUC scores of 0.82. MCC had relatively weak accuracy on the minority class but very good results for the majority class. Between PIRS-BAL and PIRS-MCC, the former had the better results on the minority class.

The Yeast 1.5 dataset is the most unbalanced of those tested, and proved to be the most difficult from the point of view of minority classification. The results illustrate that StdGP, MCC and CORR achieved relatively poor results in this respect: correctly classifying fewer than half of the minority examples. In contrast, the PIRS-BAL method produced relatively good results on both classes and had the highest AUC score.

Looking at trends in the reported results it is clear that the overall accuracy fitness measure commonly used for classification tasks in GP is inferior to all of the other methods investigated, performing poorly even on the relatively balanced Bupa42 dataset. In contrast, both of the PIRS methods performed well on all of the tasks, under each of the criteria examined: either PIRS-BAL or PIRS-MCC achieved or shared the best AUC score for all but one of the tasks, each delivered competitive results for overall accuracy on training and test data, and for both minority and majority classification. These configurations also produced the smallest trees on average, and the best-of-run individuals were discovered on average earlier in the evolutionary process.

These results suggest that Individualised Random Sampling combined with a fitness function that is designed to operate well with unbalanced datasets can deliver superior results on *both* balanced and unbalanced data.

Table 7.3: Class Imbalance Experiments: Performance of best-of-run Trained Individuals. Majority and minority class results are expressed as %Error Rate, Gen is the generation at which the best-of-run individual was discovered. Best result in each category is in bold font.

	Method	AUC	Avg. Train	Best Train	Avg. Test	Best Test	Min. Train	Min. Test	Maj. Train	Maj. Test	Size	Gen
BUPA 42	StdGP	0.74	73.68	76.32	70.96	75.44	46.97	51.95	22.85	11.75	213.0	50.50
	AVGA	0.80	80.99	83.33	74.75	78.95	27.79	42.89	10.22	11.93	88.85	48.22
	MCC	0.78	76.22	78.95	74.03	78.95	34.02	41.02	16.33	14.02	144.72	54.16
	CORR	0.76	66.46	75.43	70.10	78.94	31.52	31.79	35.00	28.46	114.25	57.08
	GMF	0.69	73.32	77.19	68.28	76.31	28.12	36.24	25.62	28.31	161.04	56.28
	PIRS-BAL	0.83	65.82	72.80	76.14	80.70	41.41	37.89	26.72	13.29	63.48	36.16
	PIRS-MCC	0.78	80.66	84.21	73.91	78.07	22.23	33.46	17.05	20.52	89.76	48.96
Ecoli 10	StdGP	0.52	91.17	94.54	86.07	89.29	80.09	91.07	6.58	3.80	79.76	23.44
	AVGA	0.72	87.68	93.18	75.71	84.82	0.63	36.92	19.07	22.63	198.40	49.00
	MCC	0.64	92.92	95.45	79.79	85.71	13.81	56.77	6.32	15.42	185.85	49.50
	CORR	0.56	73.86	91.82	70.30	88.39	27.27	36.15	26.01	28.54	121.00	50.84
	GMF	0.74	90.43	93.64	78.82	85.72	0.45	43.38	10.58	18.20	164.80	47.96
	PIRS-BAL	0.85	99.70	100	72.24	83.04	0.00	8.50	2.26	30.32	70.88	34.16
	PIRS-MCC	0.86	99.61	100	71.80	83.03	0.00	6.61	3.05	31.03	70.16	39.82
HS 36	StdGP	0.44	78.22	79.90	75.22	79.41	75.32	81.77	14.32	4.26	223.60	47.68
	AVGA	0.65	72.06	75.49	76.17	80.40	34.98	46.07	26.66	15.81	228.32	52.42
	MCC	0.73	75.32	80.39	77.63	80.40	36.68	41.71	22.37	15.05	167.96	50.38
	CORR	0.72	66.81	77.94	72.61	80.39	35.58	42.44	27.39	21.97	205.40	58.82
	GMF	0.66	72.87	76.47	76.11	79.41	32.72	45.26	25.17	16.18	190.88	52.80
	PIRS-BAL	0.75	79.74	83.25	75.63	80.39	20.85	32.28	24.37	21.38	101.85	46.80
	PIRS-MCC	0.75	80.19	83.82	76.78	81.37	23.40	34.28	23.21	19.02	104.88	47.66
Yeast 16	StdGP	0.71	87.05	88.63	86.43	88.16	61.74	58.00	9.05	4.89	166.08	46.32
	AVGA	0.80	82.04	86.12	81.91	86.33	29.95	32.03	17.26	15.37	141.36	50.04
	MCC	0.82	88.80	90.04	86.14	87.75	38.65	40.07	5.77	8.74	114.20	53.46
	CORR	0.83	74.81	88.43	75.52	87.75	26.70	23.42	24.48	24.69	119.76	58.50
	GMF	0.80	81.96	85.41	81.13	84.28	27.45	31.15	16.16	16.47	150.64	55.12
	PIRS-BAL	0.82	83.66	88.63	82.32	86.53	24.35	33.42	10.83	14.61	65.56	40.90
	PIRS-MCC	0.82	84.73	90.24	84.18	86.94	28.99	37.20	7.84	11.64	61.48	42.28
Yeast 1.5	StdGP	0.61	99.30	99.39	99.15	99.59	53.69	44.57	0.45	0.21	57.48	18.08
	AVGA	0.78	84.83	98.49	83.67	97.96	26.46	32.86	15.41	16.09	96.36	38.16
	MCC	0.64	99.30	99.40	99.06	99.39	53.38	46.57	0.00	0.28	51.36	58.38
	CORR	0.75	98.85	99.30	98.32	99.38	53.07	33.42	0.46	0.71	151.16	58.30
	GMF	0.77	86.86	96.17	85.39	95.30	12.61	32.57	13.25	14.34	168.24	56.38
	PIRS-BAL	0.80	92.58	99.59	87.36	99.18	15.87	29.14	2.47	12.39	71.16	39.32
	PIRS-MCC	0.77	99.32	99.70	99.16	99.39	25.18	32.85	0.08	0.37	37.25	29.38

7.2 Replacement Strategy, Tournament Size and Elitism

In the previous section we examined an inappropriate selection bias resulting from an inadequate fitness function applied to biased data. However, in GP the fitness function is not the only source of bias which may have an effect on either bias or variance error. Other sources of potential bias include the replacement strategy chosen and the means of applying selection pressure. Different replacement strategies may result in different individuals surviving in the population (or not), whereas different selection methods and configurations thereof determine which individuals get to mate or be mutated.

In this section we investigate the possible effects on generalisation of two popular replacement strategies together with tournament selection with a range of tournament sizes, with and without elitism. To our knowledge this is the first study on this topic which is focused on possible implications for generalisation with regard to classification problems. Aside from tournament selection, there are various other selection methods possible, but we choose to concentrate on tournament selection as it is the most popular method used by GP practitioners today.

7.2.1 Research Objective

The research objective of this study is to learn what influence, if any, tournament size, replacement strategy and elitism may have on the generalisation performance of GP for binary classification tasks.

7.2.2 Previous work

A comparison of generational and steady-state replacement strategies by Jones and Soule [2006] concluded that the steady-state approach resulted in more robust solutions. Another investigation by Whitley et al. [2006] compared steady-state with generational replacement strategies using tournament sizes of 2 and 7 on several popular GP problems including Artificial Ant, 11 Multiplexer and a symbolic regression prob-

lem. The results of that study showed that a generational strategy with tournament size of 2 was the worst performing whereas the steady-state strategy with tournament size of 2 was best overall.

With regard to GE, Ryan et al. [1998] demonstrated that the steady state approach delivered superior performance when used to solve various symbolic regression problems. The authors hypothesised that this was because GE has a tendency to produce some sub-optimal solutions which a generational algorithm may allow to filter through to later generations, but which are usually eliminated with a steady state algorithm.

In early work comparing various selection strategies for GAs Goldberg and Deb [1991] explained that when steady-state genetic algorithms use a selective strategy which involves replacing the worst individual coupled with tournament selection the actual selective pressure is much greater than the tournament size might suggest. Whitley et al. [2006] suggested that in terms of the time it takes for the best individual to take over the population under selection, a tournament size of 2 under the steady state model behaves more like a tournament size of 7 in a generational model.

Xie [2008] studied both tournament and population size from the perspectives of selection pressure and an individual's probability of being sampled. They concluded that tournament size has an effect on sampling probability where larger tournaments result in higher sampling probability and conversely lower probability of selection. The issue of multiple sampling of individuals was shown not to be a serious problem whereas for binary tournaments the higher probability of individuals not being sampled at all may lead to sub-optimal solutions. Xie investigated tournaments of size 2, 4 and 7. Poli [2005] also looked at the not sampled question, but from the perspective of investigating novel approaches to freeing up wasted resources.

There has been quite a lot of other research on tournament selection, see for example [Blickle and Thiele, 1995; Sokolov and Whitley, 2005; Thierens, 1998; Xie et al., 2007], and much of this research concentrates on effects and properties of tournament selection with regard to fitness distribution, diversity, selection pressure or sampling and is focused on training data. See Fang and Li [2010b] for a review of the topic.

In this study we instead examine the effects of various tournament sizes on generalisation for binary classification problems

7.2.3 Experimental Configurations

For these experiments we compare the performance of generational and steady-state replacement strategies with tournament sizes of 2,3,6,9,12 and 15 on nine different binary classification tasks. We report the AUC, best AUC, average training accuracy, best training accuracy, average test accuracy, best test accuracy and degree of over-fitting. Each of these relates to the best-of-run individuals. The measure of over-fitting used here is similar to that proposed in [Vanneschi et al., 2010b], except that we calculate a measure of over-fitting with regard to the training and test fitness for the *best-of-run individuals at the end of evolution*, also, we report negative over-fitting in the case where training accuracy is lower than test accuracy. A lower value for over-fitting indicates a (relatively) smaller difference between training accuracy and test accuracy, and a negative value indicates that test accuracy is better than training accuracy.

$$avgOverfit = \frac{\sum_{i=1}^n (BOR_i(TrainAcc) - BOR_i(TestAcc))}{n} \quad (7.7)$$

Where BOR_i is the best of run individual of the i^{th} run, n is the number of runs, $BOR_i(TrainAcc)$ is the training accuracy of the best of run individual and $BOR_i(TestAcc)$ is the test accuracy of the best of run individual.

For each configuration we carried out 50 runs with identical random seeds. We choose a population size of 300 and terminated evolution after 60 generations. A crossover rate of 0.8 and a mutation rate of 0.2 were employed. A generational approach is indicated with “G” and a steady-state one with “S” with the tournament size appended, together with an elitism indicator: S3 represents a non elitist steady-state algorithm with

tournament size of 3 whereas G2E represents an elitist generational replacement strategy with a tournament of size 2. Where elitism is applied, it is at the rate of 10%.

7.2.4 Results and Discussion

Experimental results for the nine datasets are extensive and can be found in Appendix E. Here we summarise the results, showing averages across the datasets under the relevant headings: tournament size, replacement strategy and elitism.

7.2.4.1 Tournament Size

In Table 7.4 we show the averages results for tournament size across all datasets and configurations. These results indicate that smaller tournaments (2 or 3) result in better average and overall AUC scores and that the smallest tournament produces the least over-fitting and the smallest solutions, on average.

Table 7.4: Selection Bias Experiments: Results by Tournament Size

Tourn. Size	AUC	Best AUC	Avg. Train	StdDev	Best Train	Avg. Test	StdDev	Best Test	Avg Size	OverFit
2	0.75	0.85	77.87	2.20	82.22	75.68	3.62	81.92	157.40	2.21
3	0.75	0.85	78.89	2.18	83.29	75.71	3.69	80.60	183.79	3.22
6	0.74	0.84	79.53	2.53	84.65	75.55	3.91	82.42	200.88	4.00
9	0.74	0.84	79.58	2.77	84.84	75.48	4.07	82.73	200.65	4.07
12	0.74	0.84	79.36	2.87	84.79	75.10	4.28	82.45	197.70	4.24
15	0.74	70.84	79.38	2.78	85.07	75.19	4.17	82.57	200.34	4.37

7.2.4.2 Replacement Strategy

Looking at results in Table 7.5 organised by replacement strategy, we can see that the Generational approach results in best average and overall AUC and produces smaller programs which have lower over-fitting.

Table 7.5: Selection Bias Experiments: Results by Replacement Strategy

Algorithm	AUC	Best AUC	Avg. Train	StdDev	Best Train	Avg. Test	StdDev	Best Test	Avg Size	OverFit
Steady-State	0.74	0.84	79.12	2.76	84.52	75.09	3.64	81.48	210.24	4.05
Generational	0.75	0.85	79.08	2.35	83.77	75.82	4.28	82.75	170.02	3.32

7.2.4.3 Elitism

Finally, if we examine the results with respect to elitism as seen in Table 7.6 we can observe that the application of elitism produces better results on training accuracy for smaller tournaments but that the same benefit is not apparent for larger tournaments. Equally good results for average AUC are achieved with both elitist and non-elitist strategies, with the elitist approach delivering the best overall AUC on average. With regard to test accuracy, the application of elitism would not appear to confer any significant advantage. The best configuration overall from the perspective of generalisation is a non-elitist strategy with a tournament size of 2.

Taking a more detailed look at the data in Appendix E we observe that in 8 of the 9 problems a non-elitist strategy resulted in the least over-fitting overall. For the most part this was with a tournament of size 2. However, when we compare the effects of elitism on over-fitting across the range of tournament sizes for the same replacement strategy there is little difference.

7.2.4.4 Further Remarks

In other experiments (also shown in Appendix E using an optimised GP set-up with bootstrapping and OICB+, the experimental results indicated that performance in terms of average AUC was sub-optimal for both the GC and CAR datasets using a generational approach. Further investigation indicated that the average program size with the generational method was disproportionately smaller for both the GC and CAR

Table 7.6: Selection Bias Experiments: Elitism

Size/Elite	AUC	Best AUC	Avg.Train	StdDev	Best Train	Avg. Test	StdDev	Best Test	Avg Size	OverFit
2E	0.75	0.85	78.75	2.13	82.96	75.84	3.73	82.74	154.15	2.89
3E	0.75	0.85	79.23	2.16	83.66	75.91	3.74	82.65	174.32	3.33
6E	0.74	0.84	79.46	2.49	84.55	75.53	3.81	82.09	181.93	3.93
9E	0.75	0.84	79.42	2.69	84.61	75.32	4.13	82.62	187.16	4.10
12E	0.74	0.84	79.17	2.89	84.86	75.07	4.30	82.42	188.51	4.10
15E	0.74	0.84	79.45	2.76	84.67	75.12	4.22	82.67	189.11	4.32
2	0.75	0.84	76.99	2.27	81.49	75.52	3.51	81.10	160.66	1.53
3	0.74	0.84	78.56	2.19	82.93	75.52	3.64	82.53	196.97	3.11
6	0.74	0.84	79.59	2.56	84.74	75.57	4.02	82.75	219.84	4.07
9	0.74	0.84	79.73	2.85	85.06	75.65	4.01	82.85	214.14	4.04
12	0.74	0.84	79.55	2.85	84.72	75.14	4.26	82.49	206.89	4.38
15	0.74	0.84	79.31	2.79	85.46	75.27	4.12	82.47	211.57	4.42

datasets in comparison with a steady state configuration. A comparison of program size is shown in Table 7.7.

As both our bootstrapping and OICB+ optimisations produce smaller programs, we hypothesise that this combined with the experimental observation in this chapter that a generational replacement strategy produces smaller solutions than a steady state one, created a situation where the system was unable to evolve programs of sufficient size to encode a good solution for these datasets, both of which have relatively large numbers of attributes and instances. We informally confirmed this hypothesis by re-running the CAR experiments using a steady-state algorithm with parsimony pressure and reduced crossover probability, whereupon the performance in terms of average AUC was significantly degraded. This problem could be overcome by running evolution for more iterations or by increasing population size.

Hunt et al. [2012] demonstrated that for classification, GP has good scalability with regard to instances but does not scale well with as the number of attributes increases. Our experience here may hint that program size is a factor in scalability with regard to attributes.

Table 7.7: Optimised GP: Replacement Strategy Size Comparison

Strategy	BIO	BT	BUPA	CAR	GC	HS	ION	PIMA	WBC
Generational	223	134	254	75	120	196	239	116	122
Steady-state	442	276	569	255	458	371	593	362	152

7.2.5 Summary

Considering these results and the more detailed ones shown in the various tables in Appendix E we can make several interesting observations:

1. Average solution size is *universally smaller* for the generational replacement strategy. A possible reason for this phenomenon is that steady-state algorithms are *overlapping* in the sense that until such time as the new population is filled, all individuals in the population, including newly created offspring, are available for selection. Since crossover increases program size [Langdon, 2000], these newly created offspring are likely to be larger than the individuals which they replace, and if these are in turn selected for crossover in the same iteration that they were created in, this will likely lead to an increase in average size over a generational strategy;
2. In 8 of the 9 problems, a non-elitist strategy resulted in the least over-fitting. Of these, 6 were associated with a generational algorithm;
3. in 7 of the 9 problems, a tournament size of 2 resulted in the least over-fitting;
4. For 7 of the 9 problems variance error is tending to increase with tournament size. This, taken together with the previous point, suggests that larger tournaments are more prone to over-fitting than smaller ones, and that a small generational algorithm is least likely to over-fit;

5. The best performing configuration in terms of AUC score is the G2 configuration.
6. When using a generational strategy together with parameter or algorithmic settings which may constrain program size, it may be necessary to run evolution for longer or with a larger population in order to achieve optimal results.

7.3 Research Outcomes

The empirical study detailed in this chapter has addressed the core question of whether there are individualised or dynamic methods which can be used with GP to mitigate the class imbalance problem. The results show that applying a proportional individualised random sampling approach, together with a fitness function designed to reward individuals with more balanced fitness, delivers models which perform significantly better than standard methods on unbalanced data sets. The hybrid method has produced better results on test data than the solely algorithmic approaches study.

In addition it is reasonable to assume that *proportional, individualised* random sampling is superior to some previous sampling methods which use equal sample sizes, given that the suggested hybrid approach outperformed the proposed method of Bhowan et al. [2009a] which in turn delivered superior results to the sampling strategy investigated by Hunt et al. [2011].

In our view, one of the most important research outcomes of this study is the observation that standard methods perform poorly on even relatively balanced data sets: the standard accuracy measure is extremely misleading as to the real underlying predictive accuracy of a GP binary classification system. In contrast, the proposed approach delivered more accurate results without loss of generalisation. The strong implication of this finding is that researchers should take care to mitigate inappropriate bias which may be introduced when using data sets which are even mildly unbalanced and should be wary of only using standard classification accuracy as a performance metric for reporting and comparison

purposes.

In the second section of this chapter we examined the influence of selection bias introduced through choices of tournament size, replacement strategy and application of elitism on generalisation behaviour. The results of the empirical study illustrate that, at least for the 9 problems studied, a tournament size of 2 combined with a non-elitist generational replacement strategy produced the best results in terms of test AUC and over-fitting.

Chapter 8

Optimised GP

In this chapter we bring together some of the enhancements and ideas discussed in previous chapters. We combine several of the more promising techniques in this final study with the aim of addressing the final research question of the thesis: “Can an optimised GP classifier deliver results on unseen data which are competitive with other state of the art classification algorithms?”.

In the sections which follow, we outline which of the various techniques proposed in previous chapters are combined for this final investigation and we list the ML classification algorithms we have chosen to compare with. Finally, we present and discuss our experimental results.

8.1 Optimising GP

Our criteria for selecting which of the previously proposed techniques to combine in this final optimised configuration, focus on those methods that have been shown to have a beneficial effect in reducing either bias or variance error and which we feel may work well together.

For the purposes of the current study the following techniques are combined to construct a holistic, Optimised GP (OGP) configuration:

Class Boundary Determination

Firstly we choose to adopt the OICB+ method which combines two techniques: *individualised class boundaries* combined with *individualised po-*

larity. For the current study we have implemented these as has been previously described in Chapter 5 Section 5.5.

Sampling

In Chapter 7 we proposed a proportional individualised random sampling method (PIRS) whereby individuals in the population are trained on potentially different numbers and instances of minority and majority class training examples. The results of that investigation revealed that the method, when combined with an *average accuracy* fitness function, delivered superior results, even when the data sets used were relatively balanced. Thus, for the current experiments, we again combine PIRS with a fitness function that has average accuracy as a component.

Bootstrapping

Previously in Chapter 4 Section 4.1.2 we investigated a novel implementation of bootstrapping, where we computed individual *bootstrap standard error (BSE)* on the results of instance classification for each GP individual and aimed at minimising this error during evolution. In that investigation, results demonstrated that the application of our particular implementation of bootstrapping produces smaller programs and improves generalisation performance. For the current investigation, we apply bootstrapping as described previously, but combine this with an average accuracy measure of classification performance for fitness function. Thus, the fitness function to be minimised used for this investigation is as outlined in equation 8.1.

$$Fit(P) = (1 - (0.5 * \left(\frac{TP}{(TP + FN)} + \frac{TN}{(TN + FP)} \right))) * BSE \quad (8.1)$$

Self-adaptation

In Chapter 5 we trialled a method of self-adaptive genetic crossover and mutation operators combined with self adaptive tournament selection, and we compared this with a population based self-adaptive mutation approach. In that study, the effects of self-adaptive genetic operators were somewhat inconclusive whereas both our self-adaptive tournament

selection and the self-adaptive mutation method proposed by Yin et al. [2007] delivered some benefits in terms of improved generalisation. We now combine these two approaches for the current investigation, alternating between adjusting either the tournament size or the mutation probability if the best training fitness has not improved for five generations or more.

Replacement Strategy

For the optimised GP configuration we ran the experiments using a generational strategy as results from our previous study in Chapter 7 suggested that a generational algorithm would yield better generalisation performance and may also yield slightly better AUC results.

Size and Complexity Constraints

Results of experiments outlined in Chapter 6 revealed that best average AUC performance was achieved when a very simple function set was used and the permitted size of programs, both at initialisation and during evolution, was relatively large. Arising out of those observations, we choose a very simple function set consisting of the arithmetic operators $+$ and $-$ an initial minimum tree depth of 4, initial maximum tree depth of 8 and a maximum tree depth of 20.

Use of Constants

Arising out of the empirical studies in Chapter 5 we observed that ephemeral random constants may not be necessary for good performance on binary classification tasks. This is perhaps not very surprising when we consider the differing objective of the model we are aiming to evolve for classification compared with, say, the type of model we need for symbolic regression tasks. Thus, for this final investigation we do not employ ephemeral random constants.

8.2 ML Classification Algorithms

We have chosen to compare with the following well known machine learning algorithms, because of the facts that they have performed well in previous experiments (as detailed in chapters 5 and 4), they are well regarded in the research as suitable choices for classification tasks, and they represent different approaches to classification.

Table 8.1: ML Algorithms

Acronym	Algorithm
J48	Decision Tree
kNN	k Nearest Neighbours
LG	Logistic Regression
NB	Naive Bayes
MLP	Multi-Layer Perceptron
SVM	Support Vector Machine

These ML algorithms have been previously described in Chapter 2.

8.3 Experiments

In this section we provide additional details of the datasets and parameter settings used for our experiments.

8.3.1 Data Sets

Nine different datasets were used for this final study: BIO, BT, BUPA, CAR, GC, HS, ION, PIMA and WBC. Detailed descriptions of these datasets can be found in Appendix A.

8.3.2 Parameter Settings

Initialisation parameters and function set are as previously explained in paragraph 8.1, initial tournament size is set to 2, we use a population size of 1000 individuals, and evolution terminates at generation 60.

SVMs have been used extensively for classification tasks, with a large degree of success. However their performance degrades significantly when faced with even moderate class imbalance in the data set [Akbari et al., 2004]. The Weka implementation provides a weight parameter which may be used to apply potentially different weights to each class. For these experiments we have weighted the minority class proportional to the degree of imbalance for each dataset for the SVM experiments. Without this weighting, the SVM AUC figures are much worse than those shown in the results. The applied weighting is potentially different for each dataset.

Another important parameter in SVM configuration is the choice of kernel function. For each dataset, we initially experimented with a radial basis kernel function, as recommended by Hsu et al. [2003], however we found that the linear kernel produced much better results for several of the datasets. Ultimately we ran each experiment with both kernels and chose the one which delivered the best results.

Probably the most important parameter for the MLP experiments is the "learning time". The default value is 500, but our experiments showed that performance was better if this value was increased to 700, above that value, performance worsened as the model may have overfit. For this setting, we initially experimented with the BUPA and BIO datasets, and subsequently applied the same setting for learning time for all of the experiments for MLP.

When using a kNN algorithm the parameter which controls the bias variance trade-off is the value of k : the number of nearest neighbours. If this value is too small the bias error is low but the variance error is high, whereas the opposite applies for high values of k , and as the value of k increases towards the number of instances new instances will be classified as belonging to the majority class. After some experimentation, we choose a value of 5 for k together with a distance weighting of $1/distance$.

8.4 Results and Discussion

In this study, we have chosen to report the average and best AUC on test data together with the average and best test accuracy for the best-of-run individuals over one hundred runs for comparison with the chosen machine learning algorithms. We have chosen to use the simplest function set, which as shown in Chapter 6, produces low variability across runs. Based on those results reported in Chapter 6, all other things being equal, we could reasonably have expected to get better best *overall* individuals if we had chosen a more complex function set. However, as we are choosing to use the *best AUC for comparison purposes* in this final study, we justify choosing the function set which delivers the least variability as in so doing we can make a stronger claim that the AUC values reported can reasonably be achieved. In any case, if we were developing a GP classifier for a real world classification task, we would likely instigate a large number of runs and then select the best solution on test data for use in the field.

In this section we compare performance of the various algorithms in terms of AUC, reporting the average and best overall result in the case of OGP. We also report the average and best test accuracy for information purposes, but the AUC measure is the primary evaluation criterion - as we have previously established in Chapter 7 that classification accuracy is *extremely* misleading in cases where there is even moderate class imbalance. See Ling et al. [2003] for a detailed explanation of why AUC should be preferred to accuracy as a measure of performance.

Beginning with table 8.2 we can see that, MLP has the best AUC score for the BIO dataset. kNN and LG also do well on this dataset, which is not unexpected as both perform best when data is plentiful which is the case here, where the BIO dataset has 1082 instances. OGP and OGPB are very competitive on this data also. The performance of the J48 decision tree algorithm is appreciably worse than the others on this problem. In the first use of this data set by Mansouri et al. [2013] they reported test accuracy of 85% with an optimised SVM. However, they did not report the AUC measure in their work.

Table 8.2: Optimised GP Experiments: Comparison of Optimised GP with ML Algorithms, BIO to BUPA

Data	Method	AUC	Best AUC	Test Accuracy	Best Test
BIO	OGP	0.89	0.91	80.44	84.88
	J48	0.78	-	81.48	-
	KNN	0.89	-	86.59	-
	LG	0.91	-	83.33	-
	NB	0.87	-	79.62	-
	MLP	0.92	-	81.11	-
	SVM	0.82	-	82.60	-
BT	OGP	0.77	0.78	74.38	78.00
	J48	0.75	-	81.18	-
	KNN	0.77	-	79.12	-
	LG	0.78	-	78.49	-
	NB	0.76	-	63.98	-
	MLP	0.79	-	81.18	-
	SVM	0.71	-	68.81	-
BUPA	OGP	0.80	0.86	74.97	81.40
	J48	0.62	-	68.60	-
	KNN	0.72	-	72.09	-
	LG	0.83	-	73.25	-
	NB	0.69	-	60.46	-
	MLP	0.80	-	73.26	-
	SVM	0.70	-	72.09	-

On the Blood Transfusion data set, the results shown in Table 8.2 again indicate that MLP was the best performing algorithm, closely followed by LG. The performance of SVM is relatively poor on this problem, which may be due to the fact that the BT data set is relatively unbalanced, although as previously mentioned, we have applied weights in the parameter set to compensate for this. KNN is also known to be sensitive to unbalanced data but does well on this task, possibly due to the small number of attributes (5) combined with a reasonable number of instances (768). Again, both OGP and OGPB compare well.

Turning our attention to the BUPA results, we can see also in Table 8.2 that the best AUC result is that produced by the OGPB configuration. Again, the LG method delivered a very good result, closely followed by the OGP average AUC. Both NB and J48 delivered relatively poor results on this data set. This is slightly surprising as we might have expected the Naive Bayes algorithm to outperform Logistic Regression on this one, as it has fewer training instances than some of the other problems studied.

Table 8.3: Optimised GP Experiments: Comparison of Optimised GP with ML Algorithms, CAR to HS

Data	Method	AUC	Best AUC	Test Accuracy	Best Test
CAR	OGP	0.76	0.79	69.64	74.92
	J48	0.51	-	89.24	-
	KNN	0.66	-	88.98	-
	LG	0.72	-	88.97	-
	NB	0.69	-	77.36	-
	MLP	0.71	-	89.10	-
	SVM	0.68	-	68.78	-
GC	OGP	0.70	0.72	64.77	71.40
	J48	0.54	-	70.80	-
	KNN	0.73	-	73.20	-
	LG	0.77	-	76.00	-
	NB	0.76	-	74.40	-
	MLP	0.67	-	65.6	-
	SVM	0.70	-	69.60	-
HS	OGP	0.80	0.85	68.84	82.89
	J48	0.50	-	76.31	-
	KNN	0.66	-	72.40	-
	LG	0.72	-	80.26	-
	NB	0.74	-	82.29	-
	MLP	0.78	-	84.21	-
	SVM	0.71	-	73.69	-

The CAR data set is the largest of those studied, both in terms of instances and attributes (5946 and 85). It also has the greatest degree of class imbalance with 6% minority class instances. Results in Table 8.3 indicate that the OGP configuration produced significantly better AUC scores than those produced by any of the other methods. Of the remaining configurations, LG was the next best performer, with MLP not far

behind. The good LG result is not surprising, as one would have expected it to do well on this task, as it is known to work well with large numbers of instances. kNN is also known to work best in this scenario, however it can be overwhelmed when faced with large numbers of attributes where some of these are not predictive - as it uses all of the attributes. Its poor result on this problem would seem to indicate that this was the case.

Table 8.4: Optimised GP Experiments: Comparison of Optimised GP with ML Algorithms, ION to WBC

Data	Method	AUC	Best AUC	Test Accuracy	Best Test
ION	OGP	0.91	0.97	85.81	90.80
	J48	0.98	-	93.10	-
	KNN	0.98	-	88.51	-
	LG	0.90	-	89.65	-
	NB	0.94	-	86.20	-
	MLP	0.86	-	93.10	-
	SVM	0.89	-	91.95	-
PIMA	OGP	0.79	0.80	69.80	73.96
	J48	0.74	-	73.73	-
	KNN	0.76	-	71.35	-
	LG	0.80	-	74.47	-
	NB	0.79	-	72.39	-
	MLP	0.77	-	70.31	-
	SVM	0.74	-	78.81	-
WBC	OGP	0.99	1	97.94	100
	J48	0.97	-	95.57	-
	KNN	0.99	-	96.80	-
	LG	0.99	-	97.35	-
	NB	0.99	-	97.35	-
	MLP	0.99	-	96.46	-
	SVM	0.98	-	97.35	-

Looking at the German Credit results reported in Table 8.3 we see that, once again, Logistic Regression scores very well. Naive Bayes also does well on this dataset which is not surprising as it is known to perform well with large datasets of high dimensionality. SVM, OGP and OGPB configurations deliver an average performance. MLP produced a worse than expected AUC score of 0.67, given that one of its advertised advantages is that it works well with a large number of inputs. In the interests of making a fair comparison of the various methods we have applied the same parameter settings for a given model for each dataset. In subsequent experiments we observed that doubling the learning time for MLP on this dataset lead to an improved AUC score of 0.71.

The OGP configuration resulted in the best AUC score on the HS data set, where both the average and best AUC were significantly better than those achieved using any of the other methods. In Table 8.3 we see that MLP had the best result and J48 the worst of the other ML algorithms.

On the ION data set, the J48 Decision Tree learner produced the best AUC score of 0.98 and the OGP best AUC is very competitive with this. The ION dataset has fewer training instances relative to some of the other datasets (348), but it has more attributes (34) than all but the CAR dataset. It is also somewhat unbalanced with 36% minority instances. MLP performed poorly on this problem whereas kNN did surprisingly well: given the size and composition of the data.

For the PIMA dataset, the best AUC scores were produced by the OGP and LG and NB configurations. The relatively weak performance of J48 is a bit surprising on this problem, as one of the attributes which is a measure of blood glucose level is known to be a strong predictor on this task. Overall, the LG method produced the best combination of AUC and classification accuracy.

The Wisconsin Breast Cancer dataset seems to be a relatively easy one for most algorithms, where similar results were achieved with all methods. However, the OGP set-up was the only one which achieved perfect classification of the test data.

Consistent with the “no free lunch” theorem [Wolpert and Macready,

1995, 1997], no single method performs best on *all* of the classification problems undertaken. However, clearly some methods have shown themselves to be better than others, at least on the datasets studied: Logistic Regression, OGP and the Multi-Layer Perceptron would appear to have produced the best results in terms of AUC score. In order to obtain an objective view of the relative performance of the various algorithms, we have analysed the results using a Friedman box plot as shown in figure 8.1. The various algorithms are initially individually ranked (from 1 to 8) separately for each data set. This data for the nine different datasets is combined to provide a ranking of the algorithms across all of the datasets.

Using this approach, which does not simply count wins, but rather takes into account the relative performance of each algorithm compared with every other algorithm on all of the classification problems tackled, makes it easier to gain a clear insight into which are most effective.

For clarity, the average and best results for the OGP configuration are shown as two separate models: for the purpose of this exercise OGP represents the average AUC and OGPB represents the best overall AUC score for the optimised GP configuration.

The box plot of the Friedman analysis shows that the OGPB model produced the best results overall, and that the OGP configuration delivered results that are extremely competitive also. Logistic regression is clearly the best performing of all of the ML algorithms excluding the GP set-ups, with MLP achieving the next best ranking. OGP performed as well as MLP but produced a much tighter distribution which is a solid endorsement of the method.

The J48 algorithm has delivered very poor results relative to the others and the results for SVM are also disappointing. As previously mentioned, the SVM method is known to under-perform when faced with unbalanced data for which we attempted to compensate by weighting the classes according to their proportions in the training set. Informal experiments demonstrated however, that determining effective weights is not a straightforward task, and it may well be the case that if different values had been applied, better results may have been achieved. In other

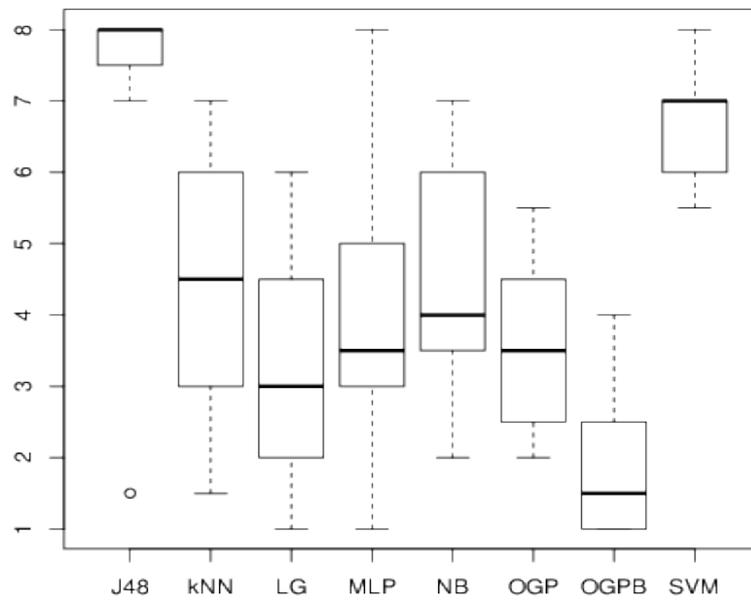


Figure 8.1: Ranked comparison of all methods across the 9 classification datasets.

work, Alfaro-Cid et al. [2008] compared the performance of a strongly typed GP configuration with a cost weighted SVM implementation, on a bankruptcy prediction task. Those researchers observed that the SVM algorithm also performed poorly on that very unbalanced problem.

With regard to MLP, subsequent experiments with some of the larger datasets indicated that better results could be obtained in some cases by increasing the learning time. However, as previously mentioned - the same strategy lead to over-fitting in other cases. In the interest of making a fair empirical comparison, the same learning time was used for all datasets in the study. It is likely that tuning this parameter individually for each dataset would facilitate improved results.

8.5 Summary

In this chapter we have integrated several of the most promising methods proposed by ourselves and others, as discussed in earlier chapters, into one holistic OGP approach. We have applied this optimised GP method to nine binary classification tasks and have compared the results to those achieved using six well known machine learning classification algorithms on the same tasks. Results obtained strongly suggest that an optimised GP learner can deliver competitive results in this scenario.

8.6 Research Outcomes

Our stated aim at the outset of this chapter was to learn if an optimised GP classifier can deliver results on unseen data which are competitive with other state of the art classification algorithms, and, the results obtained would seem to suggest that the answer to that particular question is a definitive yes. The optimised GP configuration was ranked as the best overall performing algorithm across nine datasets when the best AUC was used as the comparison metric. If instead, we choose to take a more conservative approach and eliminate the very best GP solutions from consideration, then the OGP method is very competitively ranked.

Overall, the performance of the OGP set-up is promising, and the

superior results on possibly the most difficult (CAR) dataset are particularly encouraging. Across the range of problems studied, the optimised GP configuration performed competitively on both balanced and unbalanced, large and small, low and high dimensionality datasets.

Chapter 9

Conclusions and Future Directions

9.1 Conclusions

At the beginning of this thesis we set out our central hypothesis, in which we assert that **the reduction of variance error and inappropriate bias in GP will lead to the evolution of more generalisable and robust numerical binary classifiers**. A secondary hypothesis is that dynamic, individualised approaches may have a role to play in reducing the magnitude of error due to bias and variance, thus improving generalisation in GP. In this section we evaluate research outcomes to determine if the research has confirmed the validity of the hypothesis.

9.1.1 Research Objectives

In examining our central hypothesis we identified several core questions which we felt needed to be answered in order to confirm our hypothesis.

CQ.1 **Variance:** Can novel applications of *bootstrapping*, *validation* and *early stopping* strategies be employed to reduce variance (overfitting) in evolved classifiers?

CQ.2 **Search Bias:** Can new *individualised* approaches to selection of *boundary values* and *genetic operator probabilities* lead to the evo-

lution of more generalisable solutions?

CQ.3 Language Bias: Is it possible to improve generalisation performance by implementing heuristics derived from a deeper understanding of the contributions of *program size* and *operator complexity* to language bias?

CQ.4 Selection Bias: Can new sampling and algorithmic methods designed to mitigate the *class imbalance problem*, which is influenced by *inappropriate selection bias*, deliver improvements in generalisation?

In the following sections we summarise the research outcomes for each of the enumerated core question.

9.1.1.1 CQ1: Variance

Given a model that generalises well, we would reasonably expect that its performance on training data would be a reliable indicator on its likely performance on test data, for the same problem. In Chapter 4 we proposed a novel implementation of *bootstrapping* which was designed to produce models having this quality. We carried out an empirical study [Fitzgerald et al., 2013] investigating the performance of the proposed method together with three other similar methods. Results were compared with several benchmark machine learning algorithms.

The investigation indicated that applying individualised bootstrapping improved results on test data, and that in three out of four problems studied, the *performance on training data reliably indicated the performance on the test data*. Also, in Chapter 8 the bootstrapping technique, when combined with several others to form an optimised GP configuration, delivered excellent results on a variety of datasets.

A significant benefit of the bootstrapping approach is that, at least for the problems studied, it produces smaller individuals than standard GP, where the results demonstrate a stronger negative size fitness correlation than is the with case standard GP. We presented the size distributions of best of run bootstrapping individuals and these indicated a clear skew towards smaller sized individuals.

Various researchers in the field of GP and in the wider ML community have investigated using a *validation set* as a means of improving generalisation of predictive models. *Early stopping* has been widely used in ML as a means of *avoiding* over-fitting, although it has not been widely researched as a technique in GP. In Chapter 4 we have reviewed some of the important work on these related topics and proposed a new approach (Chronos) to using validation sets with early stopping.

The Chronos system [Fitzgerald and Ryan, 2011b,c] uses *validation pressure* and *validation elitism* together with an early stopping heuristic with the aim of evolving models which generalise well by avoiding over-fitting. This avoidance is achieved by maintaining the best performing individuals on the validation set in the breeding pool, and by allowing validation performance influence selection, and also through terminating evolution before the system has the opportunity to significantly over-fit.

The results of these experiments indicated that the Chronos system produces statistically significant improvements on test data. Furthermore, results on test data were closer to those achieved on training data than was the case when the standard GP configuration was used - indicating a reduced level of over-fitting. Naturally, due to early stopping, the method also produces smaller programs and uses significantly fewer resources in general, thus confirming Chronos as a reliable approach which may be used to help reduce over-fitting.

9.1.1.2 CQ2: Search Bias

In Chapter 5 we carried out two empirical studies to address an aspect of this core question. Initially, we investigated a simple, individualised, evolved class boundary technique [Fitzgerald and Ryan, 2011a] and compared its performance on three well-known binary classification problems with that of both standard GP and the CDCBD method proposed by Zhang and Smart [2004]. The results indicated that the ECB technique delivered statistically significant improvements in training and test performance and was competitive with a wide range of previously published results for the same problems.

Encouraged by these results, we developed and extended this ap-

proach and combined it with a complementary technique which we call *individualised polarity*. We investigated this combination of techniques (OICB+) [Fitzgerald and Ryan, 2012] together with several other potential boundary determination methods. The results of these two complementary studies strongly suggest that using the proposed techniques to reduce unproductive bias from the classification process offers several significant advantages over traditional GP methods:

- Improved generalisation
- Lower variance
- Smaller programs
- Reduced run times
- Better identification of predictive variables
- Improved interpretability of evolved solutions
- Competitive with other Machine Learning algorithms.

Another interesting observation arising out of this work is that ephemeral random constants are not necessary for good performance on binary classification tasks. Both this finding and OICB+ are used successfully in Chapter 8, together with several other techniques, to construct an optimised GP configuration. In other work Esparcia-Alcázar and Sharman [1999] demonstrated that ERCs were not required to evolve good solutions to several engineering problems. Rather than use ERCs, the researchers augmented GP with numerical parameters known as *node gains* which could be optimised and/or learned during evolution.

Also in Chapter 5 we outlined previous work on self-adaptive genetic operators and went on to explore the relative merits of several individualised and population based approaches [Fitzgerald and Ryan, 2013b]. Our experimental configurations for this investigation included a population based method proposed by Yin et al. [2007]; a set-up which employed individualised self-adaptive crossover and mutation combined with self-adaptive tournament size; an approach using individualised self-adaptive

genetic operators without adaptive tournament selection; an algorithm which used random crossover and mutation, and a tuned GP configuration.

The results of these experiments are consistent with previous findings of researchers such as Spears et al. [1992], Luke and Spector [1997], and White and Poulding [2009] with regard to the sensitivity of the GP system to parameter settings for crossover and mutation, and the self-adaptive approaches performed universally better than the static population based method. Also, the proposed self-adaptive tournament selection method, which adapts the tournament size during evolution such that tournament size becomes increasingly more elitist, generated the best results on test data. Friedman analysis with post-hoc tests showed that the results produced using both of or self-adaptive methods are statistically significant when compared with the tuned GP configuration.

9.1.1.3 CQ4: Language Bias

In Chapter 6 we reviewed previous work concerning the influence of size and complexity on over-fitting and generalisation in GP. Although there has been quite a lot of work on the general topic, it is probably fair to say that the roles that program size and complexity may play in influencing bias and variance components of generalisation are not at all clear at present. Also, looking at previous work, it is not obvious that there is a consensus among researchers as to the meaning of terms such as complexity.

We were interested in looking at this fundamental question from the perspective of binary classification problems, and to this end, we undertook some simple experiments which examined the behaviour of a GP system for binary classification under a range of size and functional complexity constraints. Analysis of experimental results focused on AUC measure, over-fitting and variability of runs across the various configurations. This research generated some very interesting outcomes and also raised some questions regarding the essential relationship between over-fitting and generalisation. Extensive research findings are detailed in Chapter 6 Section 6.2 and lessons learned in that Chapter were applied

in the construction of the optimised GP algorithm which is used to good effect in Chapter 8.

9.1.1.4 CQ3: Selection Bias

In Chapter 7 we explained the problem of class imbalance in binary classification problems and detailed some of the more important work to date on the question. We carried out an empirical study comparing the performance of seven different methods [Fitzgerald and Ryan, 2013a], including those of Bhowan et al. [2009a,b] on the task of classifying five datasets, exhibiting various degrees of class imbalance. Those results indicate that one of the suggested methods, which we call Proportional Individualised Random Sampling (PIRS) combined with a fitness function that is designed to operate well with unbalanced datasets, can deliver superior results on *both* balanced and unbalanced data and produces models which perform significantly better than standard methods on unbalanced data sets.

Because this technique works well for both balanced and unbalanced datasets we chose to use it in the construction of an optimised GP configuration as described in Chapter 8.

This research provides confirmation of previous findings of Bhowan et al. [2009a] and others, that the basic accuracy measure frequently used in GP for binary classification is extremely misleading when data is unbalanced. But more importantly, we believe that a significant research outcome of this study is the observation that standard methods perform poorly on even relatively balanced data sets, implying that researchers should take care to mitigate bias which may be introduced when using data sets which are *even mildly unbalanced*.

9.1.2 Competitiveness

Our final challenge: “Can an optimised GP classifier really deliver results on unseen data which are competitive with other “state of the art” classification algorithms?” is addressed in Chapter 8. There we combined several techniques which had been developed to address our earlier funda-

mental core questions, into a single, optimised GP (OGP) configuration. This was evaluated on nine different binary classification tasks and compared with the performance of several well-known ML algorithms on the same datasets.

The results of that particular study are very encouraging and confirm that an optimised GP classifier can deliver results on unseen data which are competitive with a selection of state of the art classification algorithms.

9.1.3 Conclusion

In the previous sections, we have reviewed research outcomes for each of the fundamental core questions in turn. Here, and in the relevant chapters, the results of the various focused empirical studies show that, at least for the classification problems studied, each of the core questions can be answered in the affirmative. This being the case, we can reasonably conclude that the research detailed in this thesis confirms the original hypothesis: *the reduction of variance and inappropriate bias in GP will lead to the evolution of more generalisable and robust numerical binary classifiers.*

Additionally, as many of the novel adaptations which we believe we have successfully proposed are of an individualised or dynamic nature, we believe that we can reasonably conclude that our secondary, supporting hypothesis *that dynamic, individualised approaches may have a role to play in reducing the magnitude of error due to bias and variance, thus improving generalisation in GP* has also been vindicated in this thesis.

9.2 Future Directions

As the scope of this thesis is relatively broad, covering a range of different aspects under the general heading of improving generalisation of GP for binary classification tasks, there are a number of, possibly disparate, directions in which the work could be developed and extended. We outline some potential avenues in the following subsections:

9.2.1 ECB/OICB+

With regard to the research detailed in Chapter 5, the OICB search algorithm employed in that work does not guarantee to find the optimal boundary for a given individual as the method aimed to balance computational effort with accuracy. Future work could re-examine the algorithm with a view to improving accuracy of the boundaries discovered. Also, as mentioned earlier, at the beginning of evolution fitness scores were much better for the individual boundary methods compared with the static configuration. It would be interesting to investigate ways to leverage the excellent “head start” offered by the proposed method in order to maximise this initial advantage.

It would also be informative to apply other best-in-field GP algorithms such as Probabilistic Multi-class [Smart and Zhang, 2005] and ROC Dominance [Hunter, 2002] to the same datasets and compare OICB+ with those.

9.2.2 The Class Imbalance Problem

The solution proposed in Chapter 7 is effective on imbalanced datasets, but assumes that the proportions of majority and minority instances will remain constant in the future. To facilitate on-line learning, a useful enhancement would be to introduce a real time monitoring and update mechanism that could handle changes in these proportions which may occur over time.

The PIRS technique which we have proposed and studied in the class imbalance experiments may also naturally lend itself to an ensemble approach: where models which have been trained on different data instances and different proportions of minority and majority class instances could be combined.

9.2.3 Bootstrapping

Regarding the research outlined in Chapter 4, we know that further experiments and analysis are required in order to gain a deeper understanding on the reasons for the observed constrained program growth and to

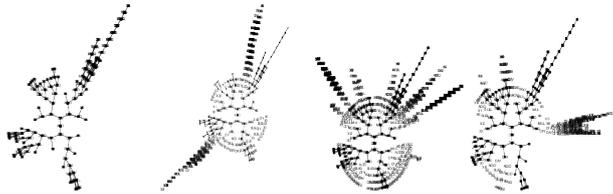


Figure 9.1: Example Bootstrap Tree Lattices.

learn more about the structure and evolution of the evolved programs. On the latter point, we noted an interesting aspect of many of the evolved programs regarding their shape - in that they tended to be much more skewed than was the case with standard GP.

Although average tree size for each of the bootstrap (BS) configurations was dramatically smaller than for GP, on average, the depth of the main bootstrap trees was not significantly shallower. On a preliminary investigation of tree shapes, it seems that those trees generated with BS tend to be both sparser and more asymmetric than those generated by standard GP. In Fig. 9.1 and 9.2 we show some example BS trees and their corresponding GP trees for the corresponding runs (identical random seeds) using “lattice” diagrams as proposed by Daida et al. [2003]. In future work we would like to establish how consistent this behaviour is within and across the experiments and to determine an appropriate measure to express it.

Ramped half and half initialisation (as used in our experiments) tends to produce “bushy” trees which may be more suited to symmetric problems such as parity where all inputs are required, rather than to data mining tasks, where the objective is to identify a subset of influential features [Poli et al., 2008a]. It would be interesting to learn if the BS method is somehow facilitating the evolution of suitably shaped tree structures.

It might also be profitable to investigate the possibility of using the deceleration of program growth associated with our application of bootstrapping as an early stopping mechanism. Although designed to be applied to GP for classification problems, it would be useful to learn if the method could be successfully applied to symbolic regression tasks also.

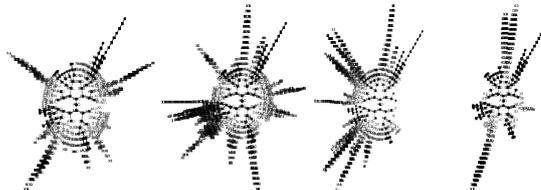


Figure 9.2: Corresponding Standard GP Tree Lattices where identical seeds to the Bootstrap runs were used, i.e. run 0 of Standard GP used the same random seed as run 0 of the Bootstrap configuration.

Models trained using a bootstrapping approach could possibly be combined perhaps based on dissimilarity in BSE or fitness to form ensemble classifiers.

9.2.4 Self Adaptive Operators

In this preliminary work we restricted our experiments to standard subtree mutation and crossover. Future research could investigate the effects of both individualization and self-adaptation on variants of these operators. It might also be productive to examine the effects of permitting inheritability of self-adapted operator probabilities, to determine if at some point an *evolutionarily stable strategy (ESS)* may emerge. Where the notion of ESS was proposed by Smith and Price [1973] and later explored by Dawkins [2006] and is defined as “a strategy, which, if most of the members of a population adopt it, cannot be bettered by an alternative strategy”.

9.2.5 Size and Functional Complexity

Research on the impact of program size and complexity on over-fitting and generalisation produced some intriguing results which warrant more detailed and focused investigation. The notions that a simple function set produces better average results than a more complex one, especially when combined with a relatively large permissible program size; that increased complexity produces increased variability, and that set-ups that exhibit relatively high over-fitting may deliver better performance on test data than configurations that do not over-fit, are all interesting findings

that should be progressed in the future. The latter point about good test results in the presence of over-fitting together with the evidence presented in this thesis regarding bias, also raises what we believe to be an important issue: where the GP community is engaged in research on generalisation the predominating focus is on the variance aspect of generalisation (over-fitting) to the exclusion of the bias component. Perhaps we should keep both in mind with an eye on the bias variance trade-off when we seek to address generalisation in our learning algorithms?

Indeed, this is the area which we intend to explore next; firstly examining key questions as they relate specifically to binary classification, and later extending the research to learn if there are any findings that may apply to the broader GP paradigm.

9.3 Further Remarks

One of the much touted advantages of GP is that it is *Domain Independent* which, in practice means that aside from implementation details concerning loading and processing of input task data, the only components of the system that vary from one type of problem to the next are the terminal set and the *evaluator*, which encapsulates *a fitness function appropriate to the problem*. It is probably fair to say, that for many practitioners, GP parameters and code employed for, say, a classification problem or a symbolic regression problem, will vary only in the data structures used to store and manipulate the training and test data, and the fitness function.

The research outcomes of this thesis could be interpreted to imply that if GP is to be optimised - then perhaps it should not be so domain independent after all? This sentiment is encapsulated in remarks expressed, many years ago, by the famous Chinese General Sun Tzu: “It is said that if you know your enemies and know yourself, you will not be imperilled in a hundred battles; if you do not know your enemies but do know yourself, you will win one and lose one; if you do not know your enemies nor yourself, you will be imperilled in every single battle” – Sun Tzu, *The Art of War*.

Although some of the techniques and observations outlined in this research may readily transfer to other types of problems, we would argue that some are particular to classification. Furthermore, the research shows that without such adaptations the basic GP classifier is a very poor performer when compared with either an optimised GP classifier or almost any other ML classifier. All of which leads to the conclusion that GP researchers may need to pay attention to gaining an understanding of unique features of the type of problem they wish to solve, if the best results possible are to be achieved.

Appendices

Appendix A

Datasets

A.1 Data Sets

In partitioning the data between training, validation and test sets, we discarded several records from each master set where the values did not divide evenly, in order to ensure sets with the correct proportions, preserving the ratio of negative to positive instances. Copies of the data for these tasks were obtained from the UCI Machine Learning Database[Bache and Lichman, 2013] unless otherwise stated.

Table A.1 contains information about the numbers of features and instances of the various numerical datasets used in the thesis, together with the acronym chosen to refer to each dataset and the percentage of minority instances.

Table A.1: Data Sets [Frank and Asuncion, 2010]

Data Set	Acronym	Features	Instances	%Minority
Bio	BIO	42	1082	33
Blood Transfusion	BT	5	768	23
Bupa Liver Disorders	BUPA	7	345	42
Caravan	CAR	85	5946	6
Ecoli	Ecoli10	7	332	10
German Credit	GC	23	1000	30
Habermans Survival	HS	4	306	36
Ionosphere	ION	34	348	36
PIMA Indians Diabetes	PIMA	9	768	35
Wisconsin Breast Cancer	WBC	10	699	46
Yeast	Yeast16	8	1484	16
Yeast(1)	Yeast1.5	8	1484	1.5

A.1.0.1 Biodegradation (BIO)

The QSAR biodegradation dataset was built in the Milano Chemometrics and QSAR Research Group Milan, Italy. The data have been used to develop QSAR (Quantitative Structure Activity Relationships) models for the study of the relationships between chemical structure and biodegradation of molecules. Biodegradation experimental values of 1055

chemicals were collected from the webpage of the National Institute of Technology and Evaluation of Japan (NITE). Classification models were developed in order to discriminate ready (356) and not ready (699) biodegradable molecules.

A.1.0.2 Blood Transfusion (BT)

The BT dataset is a subset of data from a donor database of Blood Transfusion Service Center in Hsin-Chu City in Taiwan. The center passes their blood transfusion service bus to one university in Hsin-Chu City to gather blood donated about every three months. These 748 donor data, each one included R (Recency - months since last donation), F (Frequency - total number of donation), M (Monetary - total blood donated in c.c.), T (Time - months since first donation), and a binary variable representing whether he/she donated blood in March 2007 (1 stand for donating blood; 0 stands for not donating blood).

A.1.0.3 Bupa Liver Disorders (BUPA)

This UCI Dataset was created by BUPA Medical Research Ltd and donated by Richard S. Forsyth in 1990. The dataset contains 345 instances each consisting of seven attributes. The first five attributes are all blood tests which are thought to be sensitive to liver disorders that might arise from excessive alcohol consumption. Each line in the BUPA data file constitutes the record of a single male individual. The dataset does not have any missing values.

A.1.0.4 Caravan (CAR)

This dataset is concerned with caravan insurance and contains information about customers consisting of 86 variables including product usage data and socio-demographic data derived from zip area codes. The data was supplied by the Dutch data mining company Sentient Machine Research and is based on a real world business problem. The training set contains over 5000 descriptions of customers, including the information of whether or not they have a caravan insurance policy.

A.1.0.5 Ecoli (EC)

This data contains information on protein localisation sites and was created by Kenta Nakai at the Institute of Molecular and Cellular Biology at Osaka University Japan.

A.1.0.6 German Credit (GC)

This dataset classifies people described by a set of attributes as good or bad credit risks.

A.1.0.7 Habermans' Survival (HS)

The dataset contains cases from a study that was conducted between 1958 and 1970 at the University of Chicago's Billings Hospital on the survival of patients who had undergone surgery for breast cancer. The class label for this dataset indicates which patients survived for 5 years or more and which died within 5 years of surgery.

A.1.0.8 Ionosphere (ION)

This dataset is used for classification of radar returns from the ionosphere. This radar data was collected by a system in Goose Bay, Labrador. This system consists of a phased array of 16 high-frequency antennas with a total transmitted power on the order of 6.4 kilowatts. The targets were free electrons in the ionosphere. "Good" radar returns are those showing evidence of some type of structure in the ionosphere. "Bad" returns are those that do not; their signals pass through the ionosphere.

A.1.0.9 Pima Indians Diabetes data set

The Pima Indians have been the subject of intensive diabetes research due to the high incidence of the disease in the population, and the Pima Indians Diabetes (PID) data set consists of 768 instances taken from a larger database originally owned by the American National Institute of diabetes. The data in the PID data set refers to female patients at least twenty-one years old of Pima Indian heritage living in Phoenix, Arizona.

There are 500 negative cases and 268 positive cases in this dataset, each consisting of eight numerical attributes plus a class label.

A.1.0.10 Wisconsin Breast Cancer (WBC)

This breast cancer database was originally obtained from the University of Wisconsin Hospitals, Madison from Dr. William H. Wolberg and was donated to the UCI by Olvi Mangasarian in 1992. The dataset contains 699 Instances, each of which has 11 attributes. There are 458 instances of the benign class and 241 of the malignant class. There are 16 instances which contain missing / unavailable attributes, which are denoted by "?".

In the literature, missing attributes are generally handled in one of two ways: either instances which contain missing attributes are removed from the dataset or the missing attributes are replaced with mean or median values. As there are sufficient examples for training and test purposes without the affected instances, we have chosen to remove them from the dataset.

A.1.0.11 Yeast

Similar to the ECO dataset, the yeast dataset is used for the classification task of predicting the cellular localisation sites of proteins.

A.1.0.12 Brodatz Texture (BR)

The Brodatz database is a benchmark dataset used in texture classification research. The Brodatz textures are derived from the Brodatz photographic album (1966) of both natural and man made textures, such as grass, brick, raffia and sand. The version of the database used for our research was obtained from the University of Southern California Signal and Image Processing Institute. We have chosen to work with four textures from the database: brick, bark, grass and sand.

Appendix B

OICB Algorithm

Listing B.1: Boundary Search Algorithm

```
lInitMid = progOuts[mid]
lMax = progOuts[max]
lMin = progOuts[min]
lInitMin = (lInitMid+lMin)/2
lInitMax = (lInitMid+lMax)/2
lRtnError = MaxError;

getBoundary(lMin, lMax, lInitMid, lInitMin, lInitMax, lRtnErrorCount,
            lRtnBoundary);

function getBoundary(inLowerLimit, inUpperLimit, inMid, inBottom, inTop,
                    lRtnErrorCount, lRtnBoundary)
    midError = getClassificationErrors(inMid, progOuts)
    botError = getClassificationErrors(inBottom, progOuts);
    topError = getClassificationErrors(inTop, progOuts);

    IF midError <= botError AND midError <= topError
    THEN
        bestError = midError
        bestBoundary = inMid
        newMid = inMid
        newTop = (inMid+inTop)/2
        newBottom = (inMid+inBottom)/2
    ELSE IF botError <= midError AND botError <= topError
        bestError = botError
        bestBoundary = inBottom
        newMid = inBottom
        newTop = (inBottom+inMid)/2
        newBottom = (inBottom + inLowerLimit)/2
    ELSE
        bestError = topError
        bestBoundary = inTop
        newMid = inTop
        newBottom = (inTop+inMid)/2
        newTop = (inTop+inUpperLimit)/2
    END IF

    IF bestError < lRtnErrorCount then
        lRtnErrorCount = bestError
        lRtnBoundary = bestBoundary
        getBoundary(inLowerLimit, inUpperLimit, newMid, newBottom, newTop,
                    lRtnErrorCount, lRtnBoundary)
    ELSE
        return
    END IF
```

Appendix C

ECB2 Additional Experiments

Table C.1: Comparison of Replacement Strategies

Data	Strategy	Training		Test	
		Base Avg Fitness	ECB Avg Fitness	Base Avg Fitness	ECB Avg Fitness
BUPA	SteadyState	80.56	79.21	64.27	66.80
	Generational	79.43	79.54	65.70	67.43
PIMA	SteadyState	74.36	78.68	65.94	76.81
	Generational	73.40	78.49	66.39	76.90
WBC	SteadyState	98.15	97.33	94.80	95.84
	Generational	97.95	97.86	95.25	95.90

Here are the results of additional experiments carried out to investigate the difference in results in the first and second iterations of boundary experiments. Each Table C.1 to C.4 show the result of an experiment where all other parameters, excluding the one examined, were configured identically.

Table C.2: Comparison of tournament sizes 4 and 5.

Data	Tournament Size	Training		Test	
		Base Avg Fitness	ECB Avg Fitness	Base Avg Fitness	ECB Avg Fitness
BUPA	4	80.70	80.47	66.56	68.36
	5	80.51	80.10	65.20	68.46
PIMA	4	75.30	80.08	66.49	75.66
	5	75.18	79.93	66.19	75.70
WBC	4	98.33	98.30	95.03	95.11
	5	98.43	98.36	94.92	95.19

Table C.3: Comparison of run durations 50 and 60 generations.

Data	Generations	Training		Test	
		Base Avg Fitness	ECB Avg Fitness	Base Avg Fitness	ECB Avg Fitness
BUPA	50	80.51	80.14	65.19	68.46
	60	81.91	80.47	65.49	68.36
PIMA	50	75.30	79.93	66.09	75.70
	60	76.02	80.07	66.48	75.66
WBC	50	98.47	98.38	94.96	95.19
	60	98.34	98.36	95.03	95.19

Table C.4: Comparison with and without ERCs.

Data	Configuration	Training		Test	
		Base Avg Fitness	ECB Avg Fitness	Base Avg Fitness	ECB Avg Fitness
BUPA	ERC	80.77	79.64	62.76	66.73
	No ERC	80.66	80.01	62.66	65.98
PIMA	ERC	75.63	79.94	66.95	75.95
	No ERC	75.31	79.80	66.18	75.66
WBC	ERC	98.36	98.28	94.68	95.77
	No ERC	98.42	98.29	94.94	95.12

Appendix D

Texture Experiments: ECB and Reduced Function Set

Texture Experiments The Brodatz database is a benchmark dataset used in texture classification research. The Brodatz textures are derived from the Brodatz photographic album (1966) of both natural and man made textures, such as grass, brick, raffia and sand. The version of the database used for this work was obtained from the University of Southern California Signal and Image Processing Institute. We have chosen four textures from the database: brick, bark, grass and sand. Each of the texture images are tested in combination to form six binary classification tasks. These texture classification tasks are somewhat different to the previous problems as the system has to perform feature detection as well as feature selection. The terminal inputs consist of raw pixel values in the range 0-256.

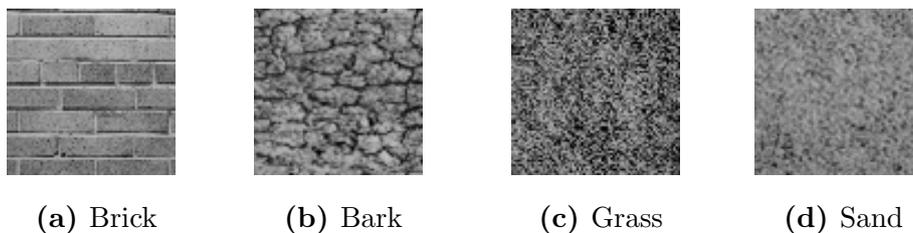


Figure D.1: Texture Images

Outcomes for training accuracy, test accuracy and program size are compared. Results shown are the averages and standard deviation at the final generation in the case of training. Test results represent the average and standard deviation of the fifty test individuals. The best result in each category is in bold text.

The following naming conventions apply:

StaticF/R = Static, zero boundary with full/reduced function set.

ECBF/R = Evolved Class Boundary with full/reduced function set.

The fitness of each program is simply the number of classification errors. Results obtained using a baseline GP configuration are compared with those achieved using an ECB configuration from Chapter 5 combined with either a very simple function set (+-) or a slightly more complex one (+ - /*). Results are shown for each of the function sets in combination with the boundary configurations for each dataset, i.e.

static or **ECB**.

Brodatz Texture Images Table D.1 details the training and test accuracy data for this data set. In a total of 1200 runs 126 perfect classifications occurred. Of these 89 were associated with an ECBR combination and the remainder with ECBF. The results show that in all cases except the brick sand combination the use of ECB resulted in higher training fitnesses. Overall, the best performing configuration was dynamic boundary combined with the reduced function set.

The outcome of these experiments adds weight to the finding regarding the effectiveness of a reduced function set. It is interesting that the effect is most pronounced when the ECB boundary method is employed. Lam and Ciesielski [2004] carried out some experiments on the same dataset, and although they did not have success with the pixel approach, they reported good results using grey level histograms to classify the textures. In that research they noted that a function set consisting solely of the addition operator produced excellent results. Interestingly, in their work, they used a KMeans clustering algorithm to perform the classification, and the ECB algorithm works in much more simplistic but similar fashion.

Statistical Tests Using a standard statistical hypothesis test for large (> 30) independent samples we calculated sample statistics for the differences in means for test fitness between StaticR and ECBR for a subset of the BROD experiments. Assuming the null hypothesis, the population means are equal. i.e. $\mu_1 = \mu_2$.

$$z = \frac{(\bar{X}_1 - \bar{X}_2) - (\mu_1 - \mu_2)}{\sqrt{\frac{\sigma_1^2}{n_1} + \frac{\sigma_2^2}{n_2}}} \quad (\text{D.1})$$

P-Values of 0.002 are achieved for each test. Applying confidence intervals, we are 95% confident that the difference achieved using the ECB method is statistically significant.

Table D.1: Brodatz Texture Training and Test Fitness

Problem	Method	Training				Test	
		Best Fitness	StdDev	Average Fitness	StdDev	Average Fitness	StdDev
Bark/Brick	<i>StaticF</i>	89.13	4.51	83.86	3.8	72.65	4.79
	<i>StaticR</i>	87.09	4.05	79.09	3.04	69.94	2.9
	<i>ECBF</i>	88.25	4.4	86.05	4.6	82.18	3.32
	<i>ECBR</i>	94.19	2.79	93.15	2.9	83.5	2.88
Bark/Grass	<i>StaticF</i>	86.47	4.11	81.23	3.33	69.31	3.04
	<i>StaticR</i>	84.73	6.11	78.65	5.65	69.06	2.98
	<i>ECBF</i>	88.56	3.42	74.62	3.81	78.06	3.92
	<i>ECBR</i>	96.97	2.06	94.78	2.19	81.63	3.05
Bark/Sand	<i>StaticF</i>	87.94	3.97	82.32	4.04	72.15	4.01
	<i>StaticR</i>	87.88	4.57	81.3	5.36	70.06	2.91
	<i>ECBF</i>	93.4	4.1	91.27	4.25	82.28	3.59
	<i>ECBR</i>	94.97	2.08	94.44	1.9	82.63	3.0
Brick/Grass	<i>StaticF</i>	90.69	3.92	84.51	3.8	76.03	4.57
	<i>StaticR</i>	87.94	4.22	80.27	4.53	73.75	3.48
	<i>ECBF</i>	97.5	2.16	93.39	2.97	95.81	2.05
	<i>ECBR</i>	99.79	0.64	94.13	3.14	97.22	1.56
Brick/Sand	<i>StaticF</i>	85.38	4.92	78.6	4.01	69.18	3.14
	<i>StaticR</i>	80.81	6.69	73.8	7.26	68.28	2.49
	<i>ECBF</i>	78.69	4.93	74.85	5.03	68.5	2.3
	<i>ECBR</i>	88.59	3.56	85.05	3.89	68.88	2.53
Grass/Sand	<i>StaticF</i>	90.88	4.19	85.21	4.02	75.65	2.42
	<i>StaticR</i>	88.81	3.49	81.68	4.44	73.66	5.06
	<i>ECBF</i>	98.77	1.49	97.65	1.02	94.37	1.89
	<i>ECBR</i>	99.81	0.51	97.59	0.94	95.47	2.09

Appendix E

Selection Bias Results

Table E.1: Replacement Strategy Experiments: Performance of best-of-run Trained Individuals, BIO

Method	AUC	Best AUC	Avg. Train	StdDev	Best Train	Avg. Test	StdDev	Best Test	Avg Size	OverFit	
BIO	S2	0.82	0.90	82.55	2.26	90.02	79.89	1.99	84.81	171.20	3.26
	S3	0.79	0.90	84.38	2.61	88.42	78.91	2.53	83.71	179.48	5.46
	S6	0.82	0.91	85.94	1.99	89.90	79.91	1.95	84.81	234.12	6.03
	S9	0.80	0.90	85.65	3.61	90.88	79.80	2.20	84.81	214.68	5.84
	S12	0.81	0.90	85.58	2.53	89.65	79.57	1.67	83.33	200.56	6.00
	S15	0.79	0.91	82.55	2.26	87.06	79.29	1.66	83.33	211.60	6.01
	S2E	0.81	0.90	85.06	2.52	87.81	79.33	2.62	84.44	150.88	4.72
	S3E	0.82	0.91	85.37	2.22	88.79	79.87	2.27	83.70	175.60	5.50
	S6E	0.80	0.90	85.64	1.97	89.53	80.01	1.88	84.44	184.12	5.62
	S9E	0.81	0.91	85.78	1.83	89.66	80.01	2.30	84.44	183.44	5.76
	S12E	0.82	0.91	85.53	2.98	89.04	79.44	2.57	84.81	188.60	6.09
	S15E	0.81	0.93	85.83	2.34	89.53	79.75	2.73	85.93	197.84	6.18
	G2	0.78	0.88	82.00	2.50	83.33	77.96	2.01	81.48	135.24	4.03
	G3	0.79	0.91	83.58	1.95	86.95	79.19	2.36	85.56	155.92	4.39
	G6	0.81	0.90	85.48	1.61	88.55	80.37	1.71	83.70	180.12	5.10
G9	0.82	0.91	85.78	1.56	88.30	79.91	2.05	84.44	172.48	5.86	
G12	0.82	0.90	85.78	2.02	89.29	79.59	2.15	84.07	164.20	6.20	
G15	0.82	0.91	82.27	1.42	89.04	79.85	2.34	84.44	172.24	6.42	
G2E	0.83	0.93	84.23	1.71	87.07	79.85	2.02	86.66	116.52	4.37	
G3E	0.82	0.90	85.37	1.54	88.79	80.13	2.08	84.81	147.00	5.23	
G6E	0.82	0.89	85.55	1.74	89.16	79.91	1.67	83.33	134.80	5.64	
G9E	0.83	0.91	85.95	1.85	90.02	80.24	2.07	84.07	154.24	5.71	
G12E	0.83	0.90	86.26	1.92	90.27	80.13	2.02	84.44	162.40	6.12	
G15E	0.83	0.92	86.00	2.01	89.03	79.70	2.06	84.44	160.36	6.30	

Table E.2: Replacement Strategy Experiments: Performance of best-of-run Trained Individuals, BT

	Method	AUC	Best AUC	Avg. Train	StdDev	Best Train	Avg. Test	StdDev	Best Test	Avg Size	OverFit
BT	S2	0.72	0.77	73.42	2.40	77.86	73.01	2.32	79.57	225.76	0.47
	S3	0.73	0.78	74.80	1.88	78.93	73.96	2.07	78.49	263.28	0.85
	S6	0.72	0.79	74.93	2.42	80.71	73.74	2.83	79.57	314.20	1.87
	S9	0.73	0.78	75.75	2.45	79.64	74.18	2.61	79.03	315.72	0.97
	S12	0.71	0.78	74.72	2.23	80.19	73.14	2.76	79.57	313.36	1.59
	S15	0.72	0.79	75.39	2.36	79.82	74.37	2.72	79.03	331.44	1.02
	S2E	0.73	0.78	74.41	1.84	78.21	74.23	2.33	80.65	210.60	0.02
	S3E	0.73	0.79	74.70	1.61	77.50	74.02	1.94	77.96	232.87	0.67
	S6E	0.73	0.79	74.90	2.14	79.46	74.20	2.54	79.03	262.78	0.69
	S9E	0.72	0.79	75.14	2.11	79.28	74.06	2.75	79.03	262.36	1.07
	S12E	0.72	0.77	75.07	2.40	80.18	73.68	2.82	78.49	284.88	1.39
	S15E	0.72	0.78	75.63	2.37	78.93	73.68	2.76	78.50	294.76	1.36
	G2	0.72	0.79	72.49	2.34	77.14	72.42	2.63	80.12	173.64	0.07
	G3	0.73	0.78	73.42	1.96	76.96	73.23	2.40	79.03	219.88	0.19
	G6	0.73	0.78	74.80	1.69	77.68	74.29	2.00	77.40	208.96	0.51
G9	0.73	0.78	74.99	2.11	78.21	74.42	2.57	80.11	231.67	0.57	
G12	0.72	0.78	75.20	1.83	79.65	74.30	2.49	79.03	240.32	0.90	
G15	0.73	0.78	75.10	1.84	79.64	74.39	2.37	79.03	225.84	0.72	
G2E	0.73	0.79	73.55	2.09	78.75	73.05	2.54	79.03	158.12	0.49	
G3E	0.73	0.78	74.24	1.77	78.40	73.88	2.51	79.57	193.88	0.36	
G6E	0.73	0.78	74.42	2.87	78.57	73.34	2.46	77.42	207.72	1.08	
G9E	0.72	0.77	74.40	1.93	78.39	73.45	2.11	77.24	192.96	0.95	
G12E	0.72	0.78	74.61	1.80	78.39	74.13	2.06	77.96	214.76	0.48	
G15E	0.72	0.79	75.10	2.38	78.93	74.55	2.73	80.64	202.32	0.56	

Table E.3: Replacement Strategy Experiments: Performance of best-of-run Trained Individuals, BUPA

	Method	AUC	Best AUC	Avg.Train	StdDev	Best Train	Avg. Test	StdDev	Best Test	Avg Size	OverFit
BUPA	S2	0.71	0.85	73.19	1.67	76.56	70.60	5.22	80.23	158.64	2.58
	S3	0.69	0.82	74.65	1.80	78.52	68.09	5.12	79.07	221.56	6.56
	S6	0.68	0.87	75.66	2.30	80.07	68.07	5.93	82.55	282.72	7.59
	S9	0.70	0.86	76.17	3.04	82.03	69.21	5.54	81.39	261.41	6.98
	S12	0.68	0.82	76.19	2.72	84.77	67.58	5.21	80.23	274.24	8.60
	S15	0.67	0.86	76.54	2.92	81.64	66.77	5.89	81.39	283.24	9.77
	S2E	0.69	0.82	74.59	1.84	78.13	68.12	4.71	76.74	197.20	6.46
	S3E	0.71	0.86	74.67	2.29	78.3	69.84	5.35	79.06	228.12	4.86
	S6E	0.68	0.82	75.81	2.38	81.25	68.02	3.91	74.74	240.00	7.79
	S9E	0.68	0.79	75.27	2.45	81.25	67.16	5.10	77.91	245.64	8.10
	S12E	0.69	0.82	74.70	3.12	80.86	68.00	5.87	80.23	244.60	6.70
	S15E	0.68	0.84	75.37	3.18	83.20	67.25	5.31	82.55	259.72	8.12
	G2	0.77	0.89	71.46	2.33	75.00	75.40	10.97	85.96	147.44	-3.93
	G3	0.73	0.89	73.53	2.65	78.07	70.26	14.03	87.72	176.06	3.26
	G6	0.71	0.89	75.26	2.80	79.82	69.85	15.04	86.84	203.00	5.42
G9	0.72	0.88	75.93	2.90	82.01	71.92	13.74	87.72	228.04	4.00	
G12	0.72	0.87	75.75	2.73	80.70	70.67	14.18	85.96	227.60	5.07	
G15	0.73	0.89	75.89	2.95	82.02	71.09	13.24	86.84	213.20	4.80	
G2E	0.73	0.90	74.00	2.39	77.63	72.23	13.90	86.84	158.64	1.78	
G3E	0.78	0.89	75.18	2.17	78.95	74.09	11.59	87.72	156.08	1.08	
G6E	0.74	0.90	75.45	2.28	79.82	73.54	12.69	86.84	192.72	1.90	
G9E	0.74	0.88	75.07	2.85	80.26	71.22	13.33	84.21	191.84	3.84	
G12E	0.71	0.89	75.42	2.10	79.38	70.65	14.35	86.84	176.64	4.77	
G15E	0.71	0.88	74.86	2.53	80.26	70.63	13.93	85.08	190.52	4.23	

Table E.4: Replacement Strategy Experiments: Performance of best-of-run Trained Individuals, CAR

	Method	AUC	Best AUC	Avg. Train	StdDev	Best Train	Avg. Test	StdDev	Best Test	Avg Size	OverFit
CAR	S2	0.66	0.78	67.26	3.45	72.72	67.72	3.30	74.06	139.12	-0.47
	S3	0.65	0.78	69.42	2.91	74.31	69.53	2.91	75.04	182.88	-0.11
	S6	0.64	0.76	67.95	5.97	81.04	68.25	5.44	80.06	159.64	-0.31
	S9	0.63	0.76	66.28	8.30	81.34	66.91	7.51	80.26	133.40	-0.63
	S12	0.61	0.76	63.31	10.06	79.03	64.20	8.96	78.01	121.84	-0.89
	S15	0.65	0.76	66.32	7.38	80.83	67.17	6.78	79.80	130.96	-0.85
	S2E	0.65	0.76	68.97	4.29	75.42	69.28	4.25	75.84	134.04	-0.30
	S3E	0.66	0.76	67.51	5.13	76.73	67.78	4.64	76.30	142.92	-0.22
	S6E	0.65	0.77	68.46	5.01	76.94	68.75	4.87	76.57	149.64	-0.28
	S9E	0.65	0.77	65.84	6.99	76.50	66.41	6.30	77.03	134.64	-0.57
	S12E	0.61	0.76	63.31	10.07	79.03	64.20	8.96	78.02	121.84	-0.90
	S15E	0.65	0.76	66.32	7.38	80.84	67.40	6.78	79.80	130.96	-0.85
	G2	0.65	0.76	66.31	3.87	74.75	67.55	3.76	75.45	108.32	-0.92
	G3	0.66	0.77	67.35	3.97	77.88	67.85	3.94	77.89	139.52	-0.50
	G6	0.66	0.77	69.06	4.74	78.60	69.38	4.46	77.69	146.28	-0.32
G9	0.66	0.79	68.94	5.09	79.50	69.12	4.63	77.89	145.96	-0.37	
G12	0.65	0.78	68.79	4.83	75.08	68.96	4.34	74.85	133.36	-0.17	
G15	0.66	0.77	67.54	5.06	76.28	68.00	4.41	74.59	127.60	-0.46	
G2E	0.66	0.75	68.48	3.53	75.94	68.83	3.59	77.49	102.04	-0.35	
G3E	0.67	0.79	68.48	3.53	75.94	69.02	3.25	75.38	116.00	-0.27	
G6E	0.65	0.79	67.61	5.24	78.31	68.19	4.95	78.42	112.84	-0.58	
G9E	0.65	0.78	67.85	6.06	78.49	68.18	5.37	77.69	107.44	-0.34	
G12E	0.64	0.77	67.58	6.06	77.91	67.92	5.46	77.10	129.44	-0.34	
G15E	0.64	0.76	67.75	5.60	75.20	68.29	5.15	75.70	107.61	-0.53	

Table E.5: Replacement Strategy Experiments: Performance of best-of-run Trained Individuals, GC

Method	AUC	Best AUC	Avg.Train	StdDev	Best Train	Avg. Test	StdDev	Best Test	Avg Size	OverFit	
GC	S2	0.59	0.75	67.84	3.42	73.85	59.94	4.76	71.20	153.20	7.89
	S3	0.61	0.69	69.14	3.10	74.53	60.89	3.88	68.40	209.12	8.25
	S6	0.59	0.69	70.25	2.98	76.13	60.12	4.25	68.40	239.90	10.13
	S9	0.60	0.72	70.80	2.88	77.06	60.80	3.46	67.0	232.48	9.99
	S12	0.61	0.71	69.82	3.44	74.53	59.76	4.88	68.40	202.52	10.06
	S15	0.59	0.71	70.47	3.52	78.67	60.41	4.27	68.80	220.80	10.18
	S2E	0.63	0.74	69.83	3.35	77.60	61.51	4.89	69.20	162.68	8.32
	S3E	0.62	0.72	70.24	2.79	74.93	61.32	3.94	68.00	211.40	8.91
	S6E	0.60	0.68	69.34	2.90	77.33	59.43	4.46	68.40	196.43	9.90
	S9E	0.59	0.72	70.20	3.07	76.00	59.72	4.28	68.80	208.80	10.49
	S12E	0.60	0.73	69.95	3.41	76.13	59.82	4.41	70.80	192.92	10.13
	S15E	0.58	0.71	69.41	3.61	75.07	58.67	4.79	68.80	197.20	10.72
	G2	0.61	0.70	67.09	3.09	75.20	60.73	4.44	70.00	148.60	6.36
	G3	0.64	0.71	68.59	2.88	73.20	60.90	3.90	68.80	188.84	7.69
	G6	0.62	0.72	69.88	2.98	76.40	60.62	4.49	68.80	211.64	9.26
G9	0.61	0.71	70.36	3.39	76.00	60.98	3.80	69.20	182.20	9.37	
G12	0.62	0.72	71.13	2.99	76.27	61.37	4.43	70.00	190.12	9.76	
G15	0.62	0.73	70.88	2.99	77.33	60.69	4.55	70.80	188.92	10.19	
G2E	0.63	0.72	69.35	2.99	75.20	61.15	4.13	71.20	138.52	8.19	
G3E	0.63	0.74	69.91	2.40	74.80	61.58	4.58	70.00	162.08	8.32	
G6E	0.63	0.71	70.54	2.72	75.67	61.42	3.79	70.80	172.56	9.11	
G9E	0.63	0.72	70.54	2.84	75.60	61.10	4.20	74.40	193.40	9.44	
G12E	0.62	0.72	70.90	2.66	76.27	61.63	4.04	69.60	183.44	9.27	
G15E	0.61	0.71	70.03	2.60	75.20	60.03	3.92	68.40	179.64	10.00	

Table E.6: Replacement Strategy Experiments: Performance of best-of-run Trained Individuals, HS

Method	AUC	Best AUC	Avg. Train	StdDev	Best Train	Avg. Test	StdDev	Best Test	Avg Size	OverFit	
HS	S2	0.76	0.87	75.14	1.99	79.65	80.92	2.58	84.21	223.04	-5.14
	S3	0.75	0.88	76.21	2.18	80.08	80.18	2.91	85.52	256.00	-3.96
	S6	0.76	0.88	76.99	3.14	83.55	79.63	2.67	84.21	280.60	-2.64
	S9	0.73	0.86	77.80	3.29	83.98	78.71	3.96	85.52	273.88	-0.91
	S12	0.73	0.87	77.62	3.30	83.11	78.66	3.57	85.52	303.80	-1.03
	S15	0.74	0.86	77.16	3.51	85.71	78.07	4.55	84.21	299.80	-0.91
	S2E	0.76	0.88	74.13	1.73	77.92	77.21	4.80	84.21	210.28	-3.08
	S3E	0.73	0.91	75.93	2.55	82.55	75.47	4.94	85.53	222.40	0.46
	S6E	0.74	0.90	77.03	2.89	84.42	76.92	4.80	85.53	228.21	0.10
	S9E	0.76	0.90	76.88	2.92	82.88	77.05	5.41	85.53	256.68	-0.17
	S12E	0.74	0.87	76.07	2.62	88.34	77.47	4.15	85.53	272.00	-1.40
	S15E	0.75	0.86	77.22	3.09	85.71	76.84	4.78	85.53	256.48	0.37
	G2	0.78	0.90	73.00	1.72	77.06	77.18	3.51	85.53	147.36	-4.68
	G3	0.76	0.90	74.59	2.20	80.52	77.55	3.55	84.21	193.64	-2.96
	G6	0.75	0.87	76.45	3.11	82.68	77.00	4.81	84.21	230.04	-0.54
G9	0.76	0.88	76.80	2.43	82.25	76.76	4.94	85.53	214.36	0.04	
G12	0.73	0.89	77.15	2.78	83.98	77.44	5.01	84.21	225.28	-0.30	
G15	0.76	0.87	76.60	3.09	83.55	77.39	4.98	84.21	226.36	-0.79	
G2E	0.78	0.89	74.43	2.00	79.65	78.55	3.66	85.52	155.76	-4.11	
G3E	0.77	0.91	75.83	2.45	81.81	77.86	4.71	84.21	164.24	-2.04	
G6E	0.76	0.89	76.13	2.59	80.52	76.86	4.64	84.21	173.40	-0.73	
G9E	0.78	0.88	76.19	2.90	83.11	77.99	4.60	86.84	190.44	-1.80	
G12E	0.78	0.89	76.00	2.52	81.38	77.81	4.01	84.21	189.80	-1.85	
G15E	0.75	0.88	76.78	2.34	83.12	77.39	3.79	82.89	210.04	-0.61	

Table E.7: Replacement Strategy Experiments: Performance of best-of-run Trained Individuals, ION

Method	AUC	Best AUC	Avg. Train	StdDev	Best Train	Avg. Test	StdDev	Best Test	Avg Size	OverFit	
ION	S2	0.83	0.95	91.30	1.80	95.00	87.19	3.73	95.41	163.62	4.10
	S3	0.83	0.93	92.82	2.12	96.54	87.20	4.02	96.55	211.04	5.63
	S6	0.83	0.94	94.45	2.48	98.07	88.21	3.70	93.10	215.72	6.24
	S9	0.85	0.95	93.61	2.5	97.31	88.64	3.70	95.40	207.52	4.97
	S12	0.83	0.98	93.72	2.64	97.70	87.15	4.18	96.55	171.00	6.57
	S15	0.84	0.95	93.55	2.74	97.69	87.22	4.60	94.25	182.60	6.34
	S2E	0.84	0.94	92.97	1.78	96.53	87.88	3.69	95.40	137.45	5.08
	S3E	0.85	0.96	93.93	1.51	96.15	89.31	3.38	95.40	171.08	4.62
	S6E	0.85	0.96	94.07	2.15	97.31	88.76	3.23	95.40	177.12	5.31
	S9E	0.84	0.93	93.77	2.73	98.08	87.61	4.06	95.64	173.48	6.16
	S12E	0.83	0.95	94.36	1.33	97.69	87.65	3.49	94.25	171.48	6.70
	S15E	0.85	0.97	94.11	2.29	98.84	87.79	4.61	95.40	164.32	6.31
	G2	0.80	0.94	89.33	1.77	93.07	86.71	3.76	74.25	122.40	2.61
	G3	0.85	0.95	93.61	1.41	96.92	89.22	2.67	94.25	160.48	4.39
	G6	0.84	0.95	93.89	1.92	97.69	88.85	3.76	95.40	179.32	5.04
G9	0.85	0.94	94.52	1.53	97.31	89.12	2.86	94.25	193.36	5.39	
G12	0.84	0.95	94.54	1.76	97.69	88.55	3.86	95.40	164.02	5.98	
G15	0.86	0.97	94.98	1.77	98.46	89.56	3.04	95.10	182.24	5.42	
G2E	0.85	0.95	93.60	1.40	96.91	89.21	2.67	94.25	160.48	5.08	
G3E	0.85	0.95	93.60	1.41	96.92	89.21	2.67	94.25	160.48	4.38	
G6E	0.84	0.95	94.11	2.22	97.69	88.88	4.10	95.40	149.72	5.44	
G9E	0.84	0.96	94.27	1.70	98.07	88.71	2.65	93.10	149.20	5.55	
G12E	0.85	0.95	94.23	2.12	97.69	88.36	4.17	94.25	136.84	5.86	
G15E	0.86	0.98	94.31	2.14	96.92	89.43	3.36	97.70	124.87	4.88	

Table E.8: Replacement Strategy Experiments: Performance of best-of-run Trained Individuals, PIMA

	Method	AUC	Best AUC	Avg.Train	StdDev	Best Train	Avg. Test	StdDev	Best Test	Avg Size	OverFit
PIMA	S2	0.65	0.74	70.26	2.66	75.00	63.00	3.32	68.75	186.44	7.26
	S3	0.62	0.73	71.81	2.59	77.65	62.13	4.05	68.75	234.08	9.68
	S6	0.63	0.73	72.99	2.78	80.38	62.26	3.56	71.87	271.68	10.74
	S9	0.62	0.73	72.66	3.15	78.64	61.56	3.89	69.27	258.88	11.10
	S12	0.62	0.72	73.44	2.20	78.22	62.24	3.50	70.83	238.12	10.80
	S15	0.63	0.71	72.87	2.83	80.56	61.69	3.34	68.75	263.00	11.17
	S2E	0.64	0.75	72.18	2.41	77.26	63.79	2.76	69.27	180.24	8.38
	S3E	0.62	0.73	72.57	2.46	79.51	62.99	4.11	71.88	199.48	9.58
	S6E	0.62	0.72	72.29	2.72	80.56	61.32	3.70	68.75	214.92	10.76
	S9E	0.62	0.73	73.15	3.22	79.16	62.22	4.43	71.35	252.72	10.92
	S12E	0.62	0.70	71.93	3.42	78.64	60.87	4.19	68.75	234.44	11.05
	S15E	0.63	0.72	72.60	2.67	77.08	61.82	4.21	68.22	242.80	10.79
	G2	0.64	0.74	68.48	2.61	74.13	62.22	3.65	69.27	156.80	6.26
	G3	0.64	0.77	71.11	2.32	76.04	62.81	3.57	72.92	184.84	8.30
	G6	0.63	0.75	72.59	2.40	76.21	62.83	4.00	71.35	216.88	9.76
G9	0.63	0.74	73.01	2.40	79.00	62.61	3.29	70.31	217.84	10.39	
G12	0.64	0.74	72.94	2.40	77.60	62.54	4.13	69.27	195.04	10.40	
G15	0.63	0.72	73.29	2.81	81.59	61.64	4.27	70.31	195.84	11.65	
G2E	0.65	0.75	71.92	1.87	76.38	63.70	3.15	73.43	126.20	8.21	
G3E	0.64	0.72	72.58	2.47	78.29	63.02	3.81	74.44	154.96	9.56	
G6E	0.64	0.72	72.75	2.38	77.78	62.91	3.63	69.27	164.00	9.83	
G9E	0.65	0.74	73.17	2.28	78.47	63.68	3.86	70.31	173.81	9.49	
G12E	0.63	0.71	73.09	2.68	78.30	62.53	3.42	68.23	161.64	10.55	
G15E	0.63	0.70	72.88	2.29	78.65	62.23	3.57	69.37	172.92	10.64	

Table E.9: Replacement Strategy Experiments: Performance of best-of-run Trained Individuals, WBC

	Method	AUC	Best AUC	Avg. Train	StdDev	Best Train	Avg. Test	StdDev	Best Test	Avg Size	OverFit
WBC	S2	0.98	1	97.57	0.51	98.43	98.56	0.61	100	189.09	-0.98
	S3	0.95	0.99	97.95	0.52	98.62	98.84	0.89	99.59	193.24	-0.39
	S6	0.97	1	98.05	0.43	99.02	98.48	0.96	100	198.72	-0.43
	S9	0.95	0.99	98.09	0.30	98.62	98.40	0.77	99.56	191.48	-0.37
	S12	0.95	1	97.96	0.54	98.62	98.22	0.83	100	195.20	-0.24
	S15	0.98	0.99	97.99	0.41	99.62	98.64	0.58	99.56	176.84	-0.65
	S2E	0.93	0.99	97.96	0.30	98.62	98.52	0.77	99.56	150.92	-0.55
	S3E	0.92	0.99	98.06	0.36	98.82	98.49	0.71	99.56	159.48	-0.43
	S6E	0.96	0.99	98.08	0.29	98.82	98.46	0.56	99.56	161.28	-0.38
	S9E	0.98	1	98.07	0.33	99.01	98.37	0.93	99.56	167.09	-0.29
	S12E	0.93	1	98.05	0.43	98.82	98.42	0.67	100	192.80	-0.37
	S15E	0.95	0.99	97.92	0.55	99.01	98.18	0.87	99.56	179.24	-0.26
	G2	0.96	0.99	97.21	0.45	98.03	98.44	0.61	99.55	142.00	-1.23
	G3	0.93	1	97.03	0.38	98.62	98.63	0.69	100	175.52	-0.79
	G6	0.90	0.99	98.07	0.41	98.82	98.32	0.81	99.56	183.52	-0.24
G9	0.92	0.99	98.08	0.30	99.01	98.58	0.64	99.56	179.16	-0.46	
G12	0.96	0.99	98.18	0.26	98.82	98.59	0.56	99.50	163.48	-0.39	
G15	0.97	1	98.11	0.29	98.81	98.64	0.62	100	175.80	-0.52	
G2E	0.93	0.99	97.88	0.21	98.22	98.66	0.62	99.56	124.08	-0.77	
G3E	0.95	1	97.98	0.28	98.62	98.53	0.76	100	139.72	-0.57	
G6E	0.93	0.99	98.09	0.31	98.82	98.54	0.62	99.56	152.49	-0.43	
G9E	0.93	1	98.09	0.28	98.82	98.62	0.64	100	130.64	-0.53	
G12E	0.96	1	98.06	0.32	99.21	98.53	0.65	100	134.72	-0.47	
G15E	0.96	0.99	98.05	0.38	98.62	98.48	0.60	99.56	132.32	-0.42	

- (2004). Special issue on learning from imbalanced datasets. *SIGKDD Explorer Newsletter*, 6(1).
- Abraham, A., Nedjah, N., and de Macedo Mourelle, L. (2006). *Evolutionary computation: from genetic algorithms to genetic programming*. Springer.
- Agrawal, R., Hedge, M., and Teneketzis, D. (1988). Asymptotically efficient adaptive allocation rules for the multi-armed bandit problem with switching cost. *Automatic Control, IEEE Transactions on*, 33(10):899–906.
- Akbani, R., Kwek, S., and Japkowicz, N. (2004). Applying support vector machines to imbalanced datasets. In *Machine Learning: ECML 2004*, pages 39–50. Springer.
- Al-Madi, N. and Ludwig, S. A. (2012). Adaptive genetic programming applied to classification in data mining. In *Nature and Biologically Inspired Computing (NaBIC), 2012 Fourth World Congress on*, pages 79–85. IEEE.
- Alfaro-Cid, E., Merelo, J., de Vega, F. F., Esparcia-Alcázar, A. I., and Sharman, K. (2010). Bloat control operators and diversity in genetic programming: A comparative study. *Evolutionary Computation*, 18(2):305–332.
- Alfaro-Cid, E., Sharman, K., and Esparcia-Alcázar, A. I. (2007). A genetic programming approach for bankruptcy prediction using a highly unbalanced database. In *Applications of Evolutionary Computing*, pages 169–178. Springer.
- Alfaro-Cid, E., Sharman, K., Esparcia-Alcázar, A. I., et al. (2008). Strong typing, variable reduction and bloat control for solving the bankruptcy prediction problem using genetic programming. In *Natural Computing in Computational Finance*, pages 161–185. Springer.
- Angeline, P. J. (1994). Genetic programming and emergent intelligence. *Advances in genetic programming*, 1:75–98.

- Angeline, P. J. (1995). Adaptive and self-adaptive evolutionary computations. In *Computational intelligence: a dynamic systems perspective*.
- Archetti, F., Lanzeni, S., Messina, E., and Vanneschi, L. (2007). Genetic programming and other machine learning approaches to predict median oral lethal dose (ld50) and plasma protein binding levels (% ppb) of drugs. In *Evolutionary Computation, Machine Learning and Data Mining in Bioinformatics*, pages 11–23. Springer.
- Azad, R. M. A. and Ryan, C. (2010). Abstract functions and lifetime learning in genetic programming for symbolic regression. In Branke, J., Pelikan, M., Alba, E., Arnold, D. V., Bongard, J., Brabazon, A., Branke, J., Butz, M. V., Clune, J., Cohen, M., Deb, K., Engelbrecht, A. P., Krasnogor, N., Miller, J. F., O’Neill, M., Sastry, K., Thierens, D., van Hemert, J., Vanneschi, L., and Witt, C., editors, *GECCO ’10: Proceedings of the 12th annual conference on Genetic and evolutionary computation*, pages 893–900, Portland, Oregon, USA. ACM.
- Azad, R. M. A. and Ryan, C. (2011b). Variance based selection to improve test set performance in genetic programming. In Krasnogor et al., editors, *GECCO ’11: Proceedings of the 13th annual conference on Genetic and evolutionary computation*, pages 1315–1322, Dublin, Ireland. ACM.
- Bache, K. and Lichman, M. (2013). UCI machine learning repository.
- Back, T., Hammel, U., and Schwefel, H.-P. (1997). Evolutionary computation: Comments on the history and current state. *Evolutionary computation, IEEE Transactions on*, 1(1):3–17.
- Badran, K. and Rockett, P. I. (2010). The influence of mutation on population dynamics in multi-objective genetic programming. *Genetic Programming and Evolvable Machines*, 11(1):5–33.
- Baesens, B., Egmont-Petersen, M., Castelo, R., and Vanthienen, J. (2002). Learning bayesian network classifiers for credit scoring using

- markov chain monte carlo search. In *Proceedings of the 16 th International Conference on Pattern Recognition (ICPR'02) Volume 3*, ICPR '02, pages 30049–, Washington, DC, USA. IEEE Computer Society.
- Banzhaf, W., Francone, F. D., and Nordin, P. (1996). The effect of extensive use of the mutation operator on generalization in genetic programming using sparse data sets. In *Parallel Problem Solving from NaturePPSN IV*, pages 300–309. Springer.
- Batista, G. E., Prati, R. C., and Monard, M. C. (2004). A study of the behavior of several methods for balancing machine learning training data. *ACM SIGKDD Explorations Newsletter*, 6(1):20–29.
- Bentley, J. L. (1975). Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517.
- Bhowan, U., Johnston, M., and Zhang, M. (2009a). Differentiating between individual class performance in genetic programming fitness for classification with unbalanced data. In *Evolutionary Computation, 2009. CEC'09. IEEE Congress on*, pages 2802–2809. IEEE.
- Bhowan, U., Johnston, M., and Zhang, M. (2011). Evolving ensembles in multi-objective genetic programming for classification with unbalanced data. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, pages 1331–1338. ACM.
- Bhowan, U., Zhang, M., and Johnston, M. (2009b). Genetic programming for image classification with unbalanced data. In *Proceeding of the 24th International Conference Image and Vision Computing New Zealand, IVCNZ '09*, pages 316–321, Wellington. IEEE.
- Bishop, C. M. (1995). Training with noise is equivalent to tikhonov regularization. *Neural computation*, 7(1):108–116.
- Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. M. Jordan, J. Kleinberg and B. Scholkopf Editors. Springer New York. 2006.

- Blickle, T. and Thiele, L. (1995). A mathematical analysis of tournament selection. In *ICGA*, pages 9–16. Citeseer.
- Bojarczuk, C. C., Lopes, H. S., and Freitas, A. A. (1999). Discovering comprehensible classification rules by using genetic programming: a case study in a medical domain. In Banzhaf, W., Daida, J., Eiben, A. E., Garzon, M. H., Honavar, V., Jakiela, M., and Smith, R. E., editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 2, pages 953–958, Orlando, Florida, USA. Morgan Kaufmann.
- Boser, B. E., Guyon, I. M., and Vapnik, V. N. (1992). A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 144–152. ACM.
- Bot, M. C. and Langdon, W. B. (2000). Application of genetic programming to induction of linear classification trees. In *Genetic Programming*, pages 247–258. Springer.
- Box, G. E. (1957). Evolutionary operation: A method for increasing industrial productivity. *Applied Statistics*, pages 81–101.
- Brameier, M. (2005). *On linear genetic programming*. PhD thesis, Dissertation, University of Dortmund.
- Breiman, L. (1996a). Bagging predictors. *Machine Learning.*, 24(2):123–140.
- Breiman, L. (1996b). Bias, variance, and arcing classifiers. Technical report, Statistics Department, University of California, Berkeley, CA, USA.
- Breiman, L. (2001). Random forests. In *Machine Learning*, pages 5–32.
- Breiman, L., Friedman, J. H., Olshen, R. A., and Stone, C. J. (1984). Classification and regression trees. *Monterey, CA*.
- Bremermann, H. J. (1962). Optimization through evolution and recombination. *Self-organizing systems*, pages 93–106.

- Brown, C. D. and Davis, H. T. (2006). Receiver operating characteristics curves and related decision measures: A tutorial. *Chemometrics and Intelligent Laboratory Systems*, 80(1):24–38.
- Castelli, M., Manzoni, L., Silva, S., and Vanneschi, L. (2010). A comparison of the generalization ability of different genetic programming frameworks. In *IEEE Congress on Evolutionary Computation (CEC 2010)*, Barcelona, Spain. IEEE Press.
- Castelli, M., Manzoni, L., Silva, S., and Vanneschi, L. (2011). A quantitative study of learning and generalization in genetic programming. In et al., S. S., editor, *Proceedings of the 14th European Conference on Genetic Programming, EuroGP 2011*, volume 6621 of *LNCS*, pages 25–36, Turin, Italy. Springer Verlag.
- Cavaretta, M. J. and Chellapilla, K. (1999). Data mining using genetic programming: the implications of parsimony on generalization error. In *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*, volume 2. IEEE.
- Cetisli, B. (2010). Development of an adaptive neuro-fuzzy classifier using linguistic hedges: Part 1. *Expert Syst. Appl.*, 37:6093–6101.
- Chang, C.-C. and Lin, C.-J. (2011). LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27.
- Chawla, N. V., Bowyer, K. W., Hall, L. O., and Kegelmeyer, W. P. (2011). Smote: synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, 16, 341–378.
- Chong, S. Y., Tino, P., and Yao, X. (2009). Relationship between generalization and diversity in coevolutionary learning. *Computational Intelligence and AI in Games, IEEE Transactions on*, 1(3):214–232.
- Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Machine learning*, 20(3):273–297.

- Costelloe, D. and Ryan, C. (2009). On improving generalisation in genetic programming. In Vanneschi, L., Gustafson, S., Moraglio, A., De Falco, I., and Ebner, M., editors, *Proceedings of the 12th European Conference on Genetic Programming, EuroGP 2009*, volume 5481 of *LNCS*, pages 61–72, Tuebingen. Springer.
- Cramer, N. L. (1985). A representation for the adaptive generation of simple sequential programs. In *ICGA*, pages 183–187.
- Da Costa, L. E. and Landry, J.-A. (2006). Relaxed genetic programming. In Keijzer, M., Cattolico, M., Arnold, D., Babovic, V., Blum, C., Bosman, P., Butz, M. V., Coello Coello, C., Dasgupta, D., Ficici, S. G., Foster, J., Hernandez-Aguirre, A., Hornby, G., Lipson, H., McMinn, P., Moore, J., Raidl, G., Rothlauf, F., Ryan, C., and Thierens, D., editors, *GECCO 2006: Proceedings of the 8th annual conference on Genetic and evolutionary computation*, volume 1, pages 937–938, Seattle, Washington, USA. ACM Press.
- Daida, J. M., Hilss, A. M., Ward, D. J., and Long, S. L. (2003). Visualizing tree structures in genetic programming. In Cantú-Paz, E., Foster, J. A., Deb, K., Davis, D., Roy, R., O’Reilly, U.-M., Beyer, H.-G., Standish, R., Kendall, G., Wilson, S., Harman, M., Wegener, J., Dasgupta, D., Potter, M. A., Schultz, A. C., Dowsland, K., Jonoska, N., and Miller, J., editors, *Genetic and Evolutionary Computation – GECCO-2003*, volume 2724 of *LNCS*, pages 1652–1664, Chicago. Springer-Verlag.
- Darwiche, M., Feuilloy, M., Bousaleh, G., and Schang, D. (2010). Prediction of blood transfusion donation. In *Research Challenges in Information Science (RCIS), 2010 Fourth International Conference on*, pages 51–56.
- Dawkins, R. (2006). *The selfish gene*. Oxford university press.
- Demšar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *J. Mach. Learn. Res.*, 7:1–30.

- Deng, H., Runger, G., and Tuv, E. (2011).x Bias of importance measures for multi-valued attributes and solutions. In *Proceedings of the 21st international conference on Artificial neural networks - Volume Part II*, ICANN'11, pages 293–300, Berlin, Heidelberg. Springer-Verlag.
- Dennett, D. C. (2003). The Baldwin effect: A crane not a skyhook. *Evolution and Learning, the Baldwin effect reconsidered*. Pages 69–79. MIT Press. 2003.
- Dietterich, T. G. (1998). Approximate statistical tests for comparing supervised classification learning algorithms. *Neural Computation*, 10(7):1895–1923.
- Dietterich, T. G. and Kong, E. B. (1995). Machine learning bias, statistical bias, and statistical variance of decision tree algorithms. *ML-95*, 255.
- Domingos, P. (1999a). Metacost: a general method for making classifiers cost-sensitive. In *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 155–164. ACM.
- Domingos, P. (1999b). The role of Occam’s razor in knowledge discovery. *Data mining and knowledge discovery*, 3(4):409–425.
- Domingos, P. (2000). A unified bias-variance decomposition. In *Proceedings of 17th International Conference on Machine Learning*. Stanford CA Morgan Kaufmann, pages 231–238.
- Domingos, P. (2012). A few useful things to know about machine learning. *Communications of the ACM*, 55(10):78–87.
- Dorigo, M. (1992). Optimization, learning and natural algorithms. *Ph. D. Thesis, Politecnico di Milano, Italy*.
- Doucette, J. and Heywood, M. (2008). GP classification under imbalanced data sets: Active sub-sampling and auc approximation. In O'Neill, M., Vanneschi, L., Gustafson, S., Esparcia-Alcázar, A., Falco,

- I., Cioppa, A., and Tarantino, E., editors, *Genetic Programming*, volume 4971 of *Lecture Notes in Computer Science*, pages 266–277. Springer Berlin Heidelberg.
- Eberhart, R. and Kennedy, J. (1995). A new optimizer using particle swarm theory. In *Micro Machine and Human Science, 1995. MHS'95., Proceedings of the Sixth International Symposium on*, pages 39–43. IEEE.
- Efron, B. and Tibshirani, R. (1994). *An Introduction to the Bootstrap*. Monographs on Statistics and Applied Probability. Taylor & Francis.
- Eggermont, J., Kok, J. N., and Kusters, W. A. (2004a). Genetic programming for data classification: Partitioning the search space. In *Proceedings of the 2004 Symposium on Applied Computing (ACM SAC'04)*, pages 1001–1005, Nicosia, Cyprus.
- Eggermont, J., Kok, J. N., and Kusters, W. A. (2004b). Genetic programming for data classification: Partitioning the search space. In *Proceedings of the 2004 Symposium on Applied Computing (ACM SAC'04)*, pages 1001–1005, Nicosia, Cyprus.
- Eiben, A. and Schippers, C. (1998). On evolutionary exploration and exploitation. *Fundamenta Informaticae*, 35(1-4):35–50.
- Eiben, A. E., Hinterding, R., and Michalewicz, Z. (1999). Parameter control in evolutionary algorithms. *Evolutionary Computation, IEEE Transactions on*, 3(2):124–141.
- Eiben, Á. E. and Jelasity, M. (2002). A critical note on experimental research methodology in ec. In *Evolutionary Computation, 2002. CEC'02. Proceedings of the 2002 Congress on*, volume 1, pages 582–587. IEEE.
- Eichler, M. (1981). The inadequacy of the monolithic model of the family. *Canadian Journal of Sociology/Cahiers canadiens de sociologie*, pages 367–388.

- Elkan, C. (2001). Magical thinking in data mining: lessons from coil challenge 2000. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 426–431. ACM.
- Esparcia-Alcázar, A. and Sharman, K. (1999). Phenotype plasticity in Genetic Programming a Comparison of Darwinian and Lamarckian Inheritance Schemes. In Poli R., Nordin P., Langdon W., and Fogarty Terence C., editors *Genetic Programming* volume 1598 of *Lecture Notes in Computer Science* pages 49–64. Springer Berlin Heidelberg.
- Esparcia-Alcázar, A. and Moravec, J. (2013). Fitness approximation for bot evolution in genetic programming. *Soft Comput.*, 17(8):1479–1487.
- Espejo, P. G., Ventura, S., and Herrera, F. (2010). A Survey on the Application of Genetic Programming to Classification. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 40(2):121–144.
- Estebanez, C., Aler, R., and Valls, J. M. (2007). A method based on genetic programming for improving the quality of datasets in classification problems. *International Journal of Computer Science and Applications*, 4(1):69–80.
- Fagan, D., Hemberg, E., Nicolau, M., O’Neill, M., and McGarraghy, S. (2012). Towards adaptive mutation in grammatical evolution. In Soule, T., Auger, A., Moore, J., Pelta, D., Solnon, C., Preuss, M., Dorin, A., Ong, Y.-S., Blum, C., Silva, D. L., Neumann, F., Yu, T., Ekart, A., Browne, W., Kovacs, T., Wong, M.-L., Pizzuti, C., Rowe, J., Friedrich, T., Squillero, G., Bredeche, N., Smith, S., Motsinger-Rei, A., Lozano, J., Pelikan, M., Meyer-Nienber, S., Igel, C., Hornby, G., Doursat, R., Gustafson, S., Olague, G., Yoo, S., Clark, J., Ochoa, G., Pappa, G., Lobo, F., Tauritz, D., Branke, J., and Deb, K., editors, *GECCO Companion ’12: Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference companion*, pages 1481–1482, Philadelphia, Pennsylvania, USA. ACM.

- Fan, W., Stolfo, S. J., Zhang, J., and Chan, P. K. (1999). Adacost: misclassification cost-sensitive boosting. In *MACHINE LEARNING-INTERNATIONAL WORKSHOP THEN CONFERENCE-*, pages 97–105. Citeseer.
- Fang, Y. and Li, J. (2010b). A review of tournament selection in genetic programming. In Cai, Z., Hu, C., Kang, Z., and Liu, Y., editors, *Advances in Computation and Intelligence*, volume 6382 of *Lecture Notes in Computer Science*, pages 181–192. Springer Berlin Heidelberg.
- Fawcett, T. (2006). An introduction to roc analysis. *Pattern Recogn. Lett.*, 27:861–874.
- Ferreira, C. (2001). Gene expression programming: A new adaptive algorithm for solving problems. *Complex Systems*, 13(2):87–129.
- Fialho, Á., Da Costa, L., Schoenauer, M., and Sebag, M. (2008). Extreme value based adaptive operator selection. In *Parallel Problem Solving from Nature: PPSN*, pages 175–184. Springer.
- Fitzgerald, J., Azad, R. M. A., and Ryan, C. (2013). A bootstrapping approach to reduce over-fitting in genetic programming. In *GECCO:Proceeding of the fifteenth annual conference on Genetic and evolutionary computation conference companion*, Amsterdam, The Netherlands. ACM.
- Fitzgerald, J. and Ryan, C. (2011a). Drawing boundaries: using individual evolved class boundaries for binary classification problems. In Krasnogor, N. and Lanzi, P. L., editors, *GECCO11:Proceedings of the 13th annual conference on Genetic and evolutionary computation*, pages 1347–1354. ACM.
- Fitzgerald, J. and Ryan, C. (2011b). Validation sets for evolutionary curtailment with improved generalisation. In *ICHIT (1)*, pages 282–289.
- Fitzgerald, J. and Ryan, C. (2011c). Validation sets, genetic programming and generalisation. In *Research and Development in Intelligent*

- Systems XXVIII. Proceedings of AI-2011 the 31st SGAI international conference on Innovative Techniques and Applications of Artificial Intelligence*, SGAI'11, pages 79–93. Springer.
- Fitzgerald, J. and Ryan, C. (2012). Exploring boundaries: optimising individual class boundaries for binary classification problem. In Soule, T. and Auger, A., editors, *Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference*, GECCO '12, pages 743–750, New York, NY, USA. ACM.
- Fitzgerald, J. and Ryan, C. (2013a). A hybrid approach to the problem of class imbalance. In *International Conference on Soft Computing*, Brno, Czech Republic.
- Fitzgerald, J. and Ryan, C. (2013b). Individualized self-adaptive genetic operators with adaptive selection in genetic programming. In *Nature and Biologically Inspired Computing (NaBIC), 2013 World Congress on*, pages 232–237. IEEE.
- Fix, E. and Hodges, J. L. (1989). Discriminatory analysis. nonparametric discrimination: Consistency properties. *International Statistical Review/Revue Internationale de Statistique*, 57(3):238–247.
- Fogel, D. (2009). *Artificial intelligence through simulated evolution*. Wiley-IEEE Press.
- Fonseca, C. M., Fleming, P. J., et al. (1993). Genetic algorithms for multiobjective optimization: Formulation discussion and generalization. In *ICGA*, volume 93, pages 416–423.
- Foreman, N. and Evett, M. (2005). Preventing overfitting in GP with canary functions. In Beyer, H.-G., O'Reilly, U.-M., Arnold, D. V., Banzhaf, W., Blum, C., Bonabeau, E. W., Cantu-Paz, E., Dasgupta, D., Deb, K., Foster, J. A., de Jong, E. D., Lipson, H., Llorca, X., Manicoridis, S., Pelikan, M., Raidl, G. R., Soule, T., Tyrrell, A. M., Watson, J.-P., and Zitzler, E., editors, *GECCO 2005: Proceedings of the 2005 conference on Genetic and evolutionary computation*, volume 2, pages 1779–1780, Washington DC, USA. ACM Press.

- Fortmann-Roe, S. (2012). Understanding the bias variance tradeoff. <http://scott.fortmann-roe.com/docs/BiasVariance.html>.
- Francone, F. D., Nordin, P., and Banzhaf, W. (1996). Benchmarking the generalization capabilities of a compiling genetic programming system using sparse data sets. In *Proceedings of the First Annual Conference on Genetic Programming*, pages 72–80, Stanford University, USA. MIT Press.
- Frank, A. and Asuncion, A. (2010). UCI machine learning repository.
- Freund, Y. and Schapire, R. E. (1996). Experiments with a new boosting algorithm. In *Thirteenth International Conference on Machine Learning*, pages 148–156, San Francisco. Morgan Kaufmann.
- Friedberg, R. M. (1958). A learning machine: Part i. *IBM Journal of Research and Development*, 2(1):2–13.
- Friedman, J. H. (1997). On bias, variance, 0/1loss, and the curse-of-dimensionality. *Data mining and knowledge discovery*, 1(1):55–77.
- Friedman, M. (1937). The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of American Statistical Assoc.*, 32(200):675–701.
- Gagné, C. and Parizeau, M. (2002). Open beagle: A new c++ evolutionary computation framework. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO '02*, pages 888–, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.
- Gagné, C., Schoenauer, M., Parizeau, M., and Tomassini, M. (2006). Genetic programming, validation sets, and parsimony pressure. In Collet, P., Tomassini, M., Ebner, M., Gustafson, S., and Ekárt, A., editors, *Proceedings of the 9th European Conference on Genetic Programming*, volume 3905 of *Lecture Notes in Computer Science*, pages 109–120, Budapest, Hungary. Springer.

- Gathercole, C. and Ross, P. (1994). Dynamic training subset selection for supervised learning in genetic programming. In *Parallel Problem Solving from Nature PPSN III*, pages 312–321. Springer.
- Gathercole, C., Ross, P., et al. (1997). Small populations over many generations can beat large populations over few generations in genetic programming. *Genetic programming*, 97.
- Geman, S., Bienenstock, E., and Doursat, R. (1992). Neural networks and the bias/variance dilemma. *Neural Computation*, 4(1):1–58.
- Gerevini, A., Saetti, A., and Serina, I. (2005). An experimental study based on Friedman’s test of some local search techniques for planning. In *Proceedings of the Workshop "Analisi Sperimentale e Benchmark di Algoritmi per l’Intelligenza Artificiale"*, pages 59–68, Associazione Italiana per l’Intelligenza Artificiale, Dipartimento di Ingegneria, Università di Ferrara, Italy.
- Goldberg, D. E. and Deb, K. (1991). A comparative analysis of selection schemes used in genetic algorithms. *Urbana*, 51:61801–2996.
- Goldman, B. W. and Tauritz, D. R. (2011). Self-configuring crossover. In Pappa, G. L., Freitas, A. A., Swan, J., and Woodward, J., editors, *GECCO 2011 1st workshop on evolutionary computation for designing generic algorithms*, pages 575–582, Dublin, Ireland. ACM.
- Gomez, F. J., Togelius, J., and Schmidhuber, J. (2009). Measuring and optimizing behavioral complexity for evolutionary reinforcement learning. In *Artificial Neural Networks–ICANN 2009*, pages 765–774. Springer.
- Gonçalves, I. and Silva, S. (2011). Experiments on controlling overfitting in genetic programming. In *15th Portuguese Conference on Artificial Intelligence (to appear)*.
- Gonçalves, I. and Silva, S. (2013). Balancing learning and overfitting in genetic programming with interleaved sampling of training data. In *Genetic Programming*, pages 73–84. Springer.

- Goncalves, I., Silva, S., Melo, J. B., and Carreiras, J. M. B. (2012). Random sampling technique for overfitting control in genetic programming. In Moraglio, A., Silva, S., Krawiec, K., Machado, P., and Cotta, C., editors, *Proceedings of the 15th European Conference on Genetic Programming, EuroGP 2012*, Volume 7244 of *LNCS*, pages 218–229, Malaga, Spain. Springer Verlag.
- Grünwald, P. (1997). The minimum description length principle and non-deductive inference. In *Proceedings of the IJCAI Workshop on Abduction and Induction in*, page <http://www.mpeg.org/>.
- Gunanidhi Pradhan, Vishal Korimilli, et al., (2009). Design of simple ann (sann) model for data classification and its performance comparison with flann (functional link ann). *International Journal of Computer Science and Network Security*, Vol. 9 No. 10:105–115.
- Gustafson, S. and Burke, E.K. and Krasnogor, N., On improving genetic programming for symbolic regression. The 2005 IEEE Congress on Evolutionary Computation, pages 912–919 Vol.1, Sept 2005.
- Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., and Witten, I. H. (2009). The weka data mining software: an update. *SIGKDD Explor. Newsl.*, 11(1):10–18.
- Harper, P. R. (2005). A review and comparison of classification algorithms for medical decision making. *Health Policy*, 71(3):315–331.
- Harper, R. (2010). Genetic programming -to much P and not enough G? In *IEEE Congress on Evolutionary Computation (CEC 2010)*, Barcelona, Spain. IEEE Press.
- Harper, R. and Blair, A. (2006). A self-selecting crossover operator. In Yen, G. G., Wang, L., Bonissone, P., and Lucas, S. M., editors, *Proceedings of the 2006 IEEE Congress on Evolutionary Computation*, pages 5569–5576, Vancouver. IEEE Press.
- He, H. and Garcia, E. A. (2009). Learning from imbalanced data. *Knowledge and Data Engineering, IEEE Transactions on*, 21(9):1263–1284.

- Hoaglin, D. C., Mosteller, F., and Tukey, J. W., editors (1983). *Understanding robust and exploratory data analysis*. Wiley series in probability and mathematical statistics. Wiley-Interscience.
- Hoffmeister, F. (1991). Scalable parallelism by evolutionary algorithms. In *Parallel Computing and Mathematical Optimization*, pages 177–198. Springer.
- Holland, J. H. (1975). *Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence*. U Michigan Press.
- Hosmer Jr, D. W., Lemeshow, S., and Sturdivant, R. X. (2013). *Applied logistic regression*. Wiley. com.
- Hsu, C.-W., Chang, C.-C., Lin, C.-J., et al. (2003). A practical guide to support vector classification.
- Hunt, R., Johnston, M., Browne, W., and Zhang, M. (2011). Sampling methods in genetic programming for classification with unbalanced data. In *AI 2010: Advances in Artificial Intelligence*, pages 273–282. Springer.
- Hunt, R., Neshatian, K., and Zhang, M. (2012). Scalability analysis of genetic programming classifiers. In Li, X., editor, *Proceedings of the 2012 IEEE Congress on Evolutionary Computation*, pages 509–516, Brisbane, Australia.
- Hunter, A. (2002). Using multiobjective genetic programming to infer logistic polynomial regression models. In *19th European Conference on Artificial Intelligence*, pages 193–197, Lyon, France.
- Hyafil, L. and Rivest, R. L. (1976). Constructing optimal binary decision trees is np-complete. *Information Processing Letters*, 5(1):15 – 17.
- Iba, H. (1999). Bagging, boosting, and bloating in genetic programming. In Banzhaf, W., Daida, J., Eiben, A. E., Garzon, M. H., Honavar, V., Jakiela, M., and Smith, R. E., editors, *Proceedings of the Genetic and*

Evolutionary Computation Conference, Volume 2, pages 1053–1060, Orlando, Florida, USA. Morgan Kaufmann.

Jabeen, H. and Baig, A. (2010a). A Framework for Optimization of Genetic Programming Evolved Classifier Expressions Using Particle Swarm Optimization. In Graña, Romay, M., Corchado, E., and Garcia, Sebastian, ., editors, *Hybrid Artificial Intelligence Systems*, Volume 6076 of *Lecture Notes in Computer Science*, chapter 7, pages 56–63–63. Springer Berlin / Heidelberg, Berlin, Heidelberg.

Jabeen, H. and Baig, A. R. (2010b). Review of classification using genetic programming. *International Journal of Engineering Science and Technology*, 2(2):94–103.

Jackson, D. (2008). The generalisation ability of a selection architecture for genetic programming. In *Parallel Problem Solving from Nature—PPSN X*, pages 468–477. Springer.

Jackson, D. (2010). Promoting phenotypic diversity in genetic programming. In Schaefer, R., Cotta, C., Kolodziej, J., and Rudolph, G., editors, *PPSN 2010 11th International Conference on Parallel Problem Solving From Nature*, Volume 6239 of *Lecture Notes in Computer Science*, pages 472–481, Krakow, Poland. Springer.

Jensen, D. D. and Cohen, P. R. (2000). Multiple comparisons in induction algorithms. *Machine Learning*, 38(3):309–338.

Johnston, M., Liddle, T., and Zhang, M. (2009). A linear regression approach to numerical simplification in tree-based genetic programming. Research report 09-7, School of Mathematics Statistics and Operations Research, Victoria University of Wellington, New Zealand.

Jones, J. and Soule, T. (2006). Comparing genetic robustness in generational vs. steady state evolutionary algorithms. In *Proceedings of the 8th annual conference on genetic and evolutionary computation*, pages 143–150. ACM.

- Joshi, M. V., Kumar, V., and Agarwal, R. C. (2001). Evaluating boosting algorithms to classify rare classes: Comparison and improvements. In *Data Mining, 2001. ICDM 2001, Proceedings IEEE International Conference on*, pages 257–264. IEEE.
- Juile, H. and Pollack, J. (1998). Coevolutionary arms race improves generalization. In Koza, J. R., editor, *Late Breaking Papers at the Genetic Programming 1998 Conference*, pages 92–100, University of Wisconsin, Madison, Wisconsin, USA. Stanford University Bookstore.
- Keijzer, M. (2003). Improving symbolic regression with interval arithmetic and linear scaling. In *Genetic Programming*, pages 70–82. Springer.
- Keijzer, M. (2004). Scaled symbolic regression. *Genetic Programming and Evolvable Machines*, 5:259–269.
- Keijzer, M. and Babovic, V. (2000a). Genetic programming, ensemble methods and the bias/variance tradeoff - introductory investigations. In Poli, R., Banzhaf, W., Langdon, W. B., Miller, J. F., Nordin, P., and Fogarty, T. C., editors, *Genetic Programming, Proceedings of EuroGP'2000*, Volume 1802 of *LNCS*, pages 76–90, Edinburgh. Springer-Verlag.
- Keijzer, M. and Babovic, V. (2000b). Genetic programming, ensemble methods and the bias/variance tradeoff—introductory investigations. In *Genetic Programming: European Conference, EuroGP 2000 Edinburgh, Scotland, UK, April 15-16, 2000 Proceedings*, number 1802, page 76. Springer.
- Kim, M., McKay, R. I. B., Hoai, N. X., and Kim, K. (2011). Operator self-adaptation in genetic programming. In Silva, S., Foster, J. A., Nicolau, M., Giacobini, M., and Machado, P., editors, *Proceedings of the 14th European Conference on Genetic Programming, EuroGP 2011*, Volume 6621 of *LNCS*, pages 215–226, Turin, Italy. Springer Verlag.
- Kohavi, R. and John, G. H. (1997). Wrappers for feature subset selection. *Artificial intelligence*, 97(1):273–324.

- Kotsiantis, S., Kanellopoulos, D., Pintelas, P., et al. (2006a). Handling imbalanced datasets: A review. *GESTS International Transactions on Computer Science and Engineering*, 30(1):25–36.
- Kotsiantis, S., Zaharakis, I., and Pintelas, P. (2007). Supervised machine learning: A review of classification techniques. *Frontiers in Artificial Intelligence and Applications*, 160:3.
- Kotsiantis, S. B., Zaharakis, I. D., and Pintelas, P. E. (2006b). Machine learning: a review of classification and combining techniques. *Artificial Intelligence Review*, 26(3):159–190.
- Kötzing, T., Neumann, F., and Spöhel, R. (2011). Pac learning and genetic programming. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, pages 2091–2096. ACM.
- Kovacs, T. and Wills, A. J. (2012). *Encyclopedia of the Sciences of Learning*, chapter Generalization Versus Discrimination in Machine Learning.
- Koza, J. R. (1990). Genetic programming: A paradigm for genetically breeding populations of computer programs to solve problems. Technical report.
- Koza, J. R. (1992). *Genetic Programming: vol. 1, On the programming of computers by means of natural selection*. MIT press.
- Koza, J. R. (1995). A response to the ml-95 paper entitled “hill climbing beats genetic search on a boolean circuit synthesis of Koza’s”. Distributed 11 July 1995 at the 1995 International Machine Learning Conference in Tahoe City, California, USA.
- Koza, J. R. (2010). Human-competitive results produced by genetic programming. *Genetic Programming and Evolvable Machines*, 11(3-4):251–284.
- Koza, J. R., Keane, M. A., and Streeter, M. J. (2004). Routine automated synthesis of five patented analog circuits using genetic programming. *Soft Computing*, 8(5):318–324.

- Kramer, O. (2010). Evolutionary self-adaptation: a survey of operators and strategy parameters. *Evolutionary Intelligence*, 3:51–65.
- Kubat, M., Matwin, S., et al. (1997). Addressing the curse of imbalanced training sets: one-sided selection. In *MACHINE LEARNING-INTERNATIONAL WORKSHOP THEN CONFERENCE-*, pages 179–186. MORGAN KAUFMANN PUBLISHERS, INC.
- Kullback, S. and Leibler, R. A. (1951). On information and sufficiency. *Annals of Mathematical Statistics*, 22:49–86.
- Kushchu, I. (2002a). An evaluation of evolutionary generalisation in genetic programming. *Artificial Intelligence Review*, 18(1):3–14.
- Kushchu, I. (2002b). Genetic programming and evolutionary generalization. *IEEE Transactions on Evolutionary Computation*, 6(5):431–442.
- Lam, B. and Ciesielski, V. (2004). Discovery of human-competitive image texture feature extraction using genetic programming. In K. Debs, et al., editors, *LNCS*, Volume 3103, pages 1114–1125. GECCO, Springer-Verlag Berlin Heidelberg.
- Langdon, W. B. (1999). Size fair tree crossovers. In Postma, E. and Gyssen, M., editors, *Proceedings of the Eleventh Belgium/Netherlands Conference on Artificial Intelligence (BNAIC'99)*, pages 255–256, Kasteel Vaeshartelt, Maastricht, Holland.
- Langdon, W. B. (2000). Size fair and homologous tree crossovers for tree genetic programming. *Genetic programming and evolvable machines*, 1(1-2):95–119.
- Langdon, W. B. (2011). Generalisation in genetic programming. In Krasnogor, N., Lanzi, P. L., Engelbrecht, A., Pelta, D., Gershenson, C., Squillero, G., Freitas, A., Ritchie, M., Preuss, M., Gagne, C., Ong, Y. S., Raidl, G., Gallagher, M., Lozano, J., Coello-Coello, C., Silva, D. L., Hansen, N., Meyer-Nieberg, S., Smith, J., Eiben, G., Bernado-Mansilla, E., Browne, W., Spector, L., Yu, T., Clune, J., Hornby, G., Wong, M.-L., Collet, P., Gustafson, S., Watson, J.-P., Sipper, M.,

- Poulding, S., Ochoa, G., Schoenauer, M., Witt, C., and Auger, A., editors, *GECCO '11: Proceedings of the 13th annual conference companion on Genetic and evolutionary computation*, page 205, Dublin, Ireland. ACM.
- Langdon, W. B. and Poli, R. (1998). Why ants are hard. Technical Report CSRP-98-4, University of Birmingham, School of Computer Science. Presented at GP-98.
- Langton, C. G. (1986). Studying artificial life with cellular automata. *Physica D: Nonlinear Phenomena*, 22(1):120–149.
- Li, Y., Ma, J., and Zhao, Q. (2008). Two improvements in genetic programming for image classification. In Wang, J., editor, *2008 IEEE World Congress on Computational Intelligence*, pages 2492–2497, Hong Kong. IEEE Computational Intelligence Society, IEEE Press.
- Lim, T.-S., LOH, W.-Y., and Cohen, W. (2000). A comparison of prediction accuracy, complexity, and training time of thirty-three old and new classification algorithms.
- Ling, C. X., Huang, J., and Zhang, H. (2003). Auc: a better measure than accuracy in comparing learning algorithms. In *Advances in Artificial Intelligence*, pages 329–341. Springer.
- Liu, Y. and Khoshgoftaar, T. (2004). Reducing overfitting in genetic programming models for software quality classification. In *High Assurance Systems Engineering, 2004. Proceedings. Eighth IEEE International Symposium on*, pages 56–65. IEEE.
- Lotte, F., Congedo, M., Lécuyer, A., Lamarche, F., Arnaldi, B., et al. (2007). A review of classification algorithms for EEG based brain-computer interfaces. *Journal of neural engineering*, 4.
- Loveard, T. (2003). *Genetic Programming for Classification Learning Problems*. PhD thesis, School of Computer Science and Information Technology, Royal Melbourne Institute of Technology.

- Loveard, T. and Ciesielski, V. (2001). Representing classification problems in genetic programming. In *Proceedings of the Congress on Evolutionary Computation*, Volume 2, pages 1070–1077, COEX, World Trade Center, 159 Samseong-dong, Gangnam-gu, Seoul, Korea. IEEE Press.
- Loveard, T. and Ciesielski, V. (2002). Employing nominal attributes in classification using genetic programming. In *Conference on Simulated Evolution And Learning (SEAL02)*, pages 487–491, Singapore. IEEE.
- Luke, S. and Panait, L. (2002). Lexicographic parsimony pressure. In Langdon, W. B., Cantú-Paz, E., Mathias, K., Roy, R., Davis, D., Poli, R., Balakrishnan, K., Honavar, V., Rudolph, G., Wegener, J., Bull, L., Potter, M. A., Schultz, A. C., Miller, J. F., Burke, E., and Jonoska, N., editors, *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 829–836, New York. Morgan Kaufmann Publishers.
- Luke, S. and Panait, L. (2006). A comparison of bloat control methods for genetic programming. *Evolutionary Computation*, 14(3):309–344.
- Luke, S. and Spector, L. (1997). A comparison of crossover and mutation in genetic programming. *Genetic Programming*, 97:240–248.
- Mahler, S., Robilliard, D., and Fonlupt, C. (2005). Tarpeian bloat control and generalization accuracy. In *Genetic Programming*, pages 203–214. Springer.
- Mansouri, K., Ringsted, T., Ballabio, D., Todeschini, R., and Consonni, V. (2013). Quantitative structure–activity relationship models for ready biodegradability of chemicals. *Journal of chemical information and modeling*, 53(4):867–878.
- Maturana, J. and Saubion, F. (2008). A compass to guide genetic algorithms. In *Parallel Problem Solving from Nature–PPSN X*, pages 256–265. Springer.
- McCarthy, J. (1987). Generality in artificial intelligence. *Communications of the ACM*, 30(12):1030–1035.

- McKay, R. I. (2000). Fitness sharing in genetic programming. In Whitley, D., Goldberg, D., et al., editors, *Genetic Evolutionary Computation Conference, GECCO*, pages 435–442, Las Vegas, Nevada, USA.
- Meyer, D. (2003). The support vector machine under test. *Neurocomputing*, 55(1-2):169–186.
- Michie, D., Spiegelhalter, D. J., Taylor, C. C., and Campbell, J., editors (1994). *Machine learning, neural and statistical classification*. Ellis Horwood, Upper Saddle River, NJ, USA.
- Mierswa, I., Wurst, M., Klinkenberg, R., Scholz, M., and Euler, T. (2006). Yale: Rapid prototyping for complex data mining tasks. In Ungar, L., Craven, M., Gunopulos, D., and Eliassi-Rad, T., editors, *KDD '06: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 935–940, New York, NY, USA. ACM.
- Miller, J. F. (1999). An empirical study of the efficiency of learning boolean functions using a cartesian genetic programming approach. In *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 2, pages 1135–1142.
- Miller, J. F. and Thomson, P. (1998). Aspects of digital evolution: Geometry and learning. In *Proceedings of the Second International Conference on Evolvable Systems*, pages 25–35. Springer-Verlag.
- Mitchell, T. M. (1980). *The need for biases in learning generalizations*. Department of Computer Science, Laboratory for Computer Science Research, Rutgers Univ.
- Mitchell, T. M. (1997). *Machine learning*. McGraw-Hill, New York, NY.
- Montana, D. J. (1995). Strongly typed genetic programming. *Evolutionary computation*, 3(2):199–230.
- Morgan, N. and Bourlard, H. (1989). Generalization and parameter estimation in feedforward nets: Some experiments. In *NIPS*, pages 630–637.

- Mulder, H. A., Rönnegård, L., Fikse, W. F., Veerkamp, R. F., and Strandberg, E. (2013). Estimation of genetic variance for macro- and micro-environmental sensitivity using double hierarchical generalized linear models. *Genetics Selection Evolution*, 45(1):23.
- Muni, D. P., Pal, N. R., and Das, J. (2004). A novel approach to design classifier using genetic programming. *IEEE Transactions on Evolutionary Computation*, 8(2):183–196.
- Murphy, G. and Ryan, C. (2008). A simple powerful constraint for genetic programming. In *Genetic Programming: 11th European Conference, EuroGP 2008, Naples, Italy, March 26-28, 2008, Proceedings*, volume 4971, page 146. Springer.
- Naik, T. R. and Dabhi, V. K. (2013). Improving generalization ability of genetic programming: Comparative study. *Journal of Bioinformatics and Intelligent Control*, 2(4):243–252.
- Nguyen, T. H., Nguyen, X. H., McKay, B., and Nguyen, Q. U. (2012). Where should we stop? An investigation on early stopping for GP learning. In *Simulated Evolution and Learning*, pages 391–399. Springer.
- Nikolaev, N., de Menezes, L. M., and Iba, H. (2002). Overfitting avoidance in Genetic Programming of Polynomials. In *Evolutionary Computation, 2002. CEC'02. Proceedings of the 2002 Congress on*, volume 2, pages 1209–1214. IEEE.
- Nikolaev, N. Y. and Iba, H. (2001). Regularization approach to inductive Genetic Programming. *Evolutionary Computation, IEEE Transactions on*, 5(4):359–375.
- Nordin, P. and Banzhaf, W. (1995). Complexity compression and evolution. In *Proceedings of the 6th International Conference on Genetic Algorithms*, pages 310–317, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

- Oakley, E. H. N. (1996). Genetic programming, the reflection of chaos, and the bootstrap: Towards a useful test for chaos. In Koza, J. R., Goldberg, D. E., Fogel, D. B., and Riolo, R. L., editors, *Genetic Programming 1996: Proceedings of the First Annual Conference*, pages 175–181, Stanford University, CA, USA. MIT Press.
- O’Neill, M. and Ryan, C. (2003). *Grammatical Evolution: Evolutionary automatic programming in an arbitrary language*, volume 4. Springer.
- O’Neill, M., Vanneschi, L., Gustafson, S., and Banzhaf, W. (2010). Open issues in Genetic Programming. *Genetic Programming and Evolvable Machines*, 11(3/4):339–363. Tenth Anniversary Issue: Progress in Genetic Programming and Evolvable Machines.
- Owens, A. J., Walsh, M. J., and Fogel, L. J. (1966). *Artificial intelligence through simulated evolution*.
- ONEill, M., Vanneschi, L., Gustafson, S., and Banzhaf, W. (2010). Open issues in Genetic Programming. *Genetic Programming and Evolvable Machines*, 11(3-4):339–363.
- OReilly, U.-M. and Hemberg, M. (2007). Integrating generative growth and evolutionary computation for form exploration. *Genetic Programming and Evolvable Machines*, 8(2):163–186.
- Panait, L. and Luke, S. (2003). Methods for evolving robust programs. In Cantú-Paz, E., Foster, J. A., Deb, K., Davis, D., Roy, R., O’Reilly, U.-M., Beyer, H.-G., Standish, R., Kendall, G., Wilson, S., Harman, M., Wegener, J., Dasgupta, D., Potter, M. A., Schultz, A. C., Dowsland, K., Jonoska, N., and Miller, J., editors, *Genetic and Evolutionary Computation – GECCO-2003*, volume 2724 of *LNCS*, pages 1740–1751, Chicago. Springer-Verlag.
- Paris, G., Robilliard, D., and Fonlupt, C. (2003). Exploring over-fitting in Genetic Programming. In Liardet, P., Collet, P., Fonlupt, C., Lutton, E., and Schoenauer, M., editors, *Evolution Artificielle, 6th International Conference*, volume 2936 of *Lecture Notes in Computer Science*, pages 267–277, Marseilles, France. Springer. Revised Selected Papers.

- Parker, S. (2003). *McGraw-Hill dictionary of scientific and technical terms*. MCGRAW HILL DICTIONARY OF SCIENTIFIC AND TECHNICAL TERMS. McGraw-Hill.
- Parrott, D., Li, X., and Ciesielski, V. (2005). Multi-objective techniques in Genetic Programming for evolving classifiers. In Corne, D., Michalewicz, Z., Dorigo, M., Eiben, G., Fogel, D., Fonseca, C., Greenwood, G., Chen, T. K., Raidl, G., Zalzal, A., Lucas, S., Paechter, B., Willies, J., Guervos, J. J. M., Eberbach, E., McKay, B., Channon, A., Tiwari, A., Volkert, L. G., Ashlock, D., and Schoenauer, M., editors, *Proceedings of the 2005 IEEE Congress on Evolutionary Computation*, volume 2, pages 1141–1148, Edinburgh, UK. IEEE Press.
- Patterson, G. and Zhang, M. (2007). Fitness functions in Genetic Programming for classification with unbalanced data. In Orgun, M. A. and Thornton, J., editors, *Proceedings of the 20th Australian Joint Conference on Artificial Intelligence*, Lecture Notes in Computer Science, pages 769–775, Gold Coast, Australia. Springer.
- Pham, D., Ghanbarzadeh, A., Koc, E., Otri, S., Rahim, S., and Zaidi, M. (2005). The bees algorithm. technical note. *Manufacturing Engineering Centre, Cardiff University, UK*, pages 1–57.
- Polat, K. and Güneş, S. (2008). Artificial immune recognition system with fuzzy resource allocation mechanism classifier, principal component analysis and fft method based new hybrid automated identification system for classification of eeg signals. *Expert Syst. Appl.*, 34:2039–2048.
- Poli, R. (2003). A simple but theoretically-motivated method to control bloat in Genetic Programming. In *Genetic Programming: 6th European Conference, EuroGP 2003, Essex, UK, April 14-16, 2003. Proceedings*, volume 6, page 204. Springer.
- Poli, R. (2005). Tournament selection, iterated coupon-collection problem, and backward-chaining evolutionary algorithms. In Wright, A., Vose, M., Jong, K., and Schmitt, L., editors, *Foundations of Genetic*

- Algorithms*, volume 3469 of *Lecture Notes in Computer Science*, pages 132–155. Springer Berlin Heidelberg.
- Poli, R., Langdon, W. B., and McPhee, N. F. (2008a). *A Field Guide to Genetic Programming*. Lulu.com.
- Poli, R., McPhee, N. F., and Vanneschi, L. (2008b). Elitism reduces bloat in Genetic Programming. In Keijzer, M., Antoniol, G., Congdon, C. B., Deb, K., Doerr, B., Hansen, N., Holmes, J. H., Hornby, G. S., Howard, D., Kennedy, J., Kumar, S., Lobo, F. G., Miller, J. F., Moore, J., Neumann, F., Pelikan, M., Pollack, J., Sastry, K., Stanley, K., Stoica, A., Talbi, E.-G., and Wegener, I., editors, *GECCO '08: Proceedings of the 10th annual conference on Genetic and evolutionary computation*, pages 1343–1344, Atlanta, GA, USA. ACM.
- Poli, R., Vanneschi, L., Langdon, W. B., and McPhee, N. F. (2010). Theoretical results in Genetic Programming: the next ten years? *Genetic Programming and Evolvable Machines*, 11(3-4):285–320.
- Powers, D. (2011). Evaluation: From precision, recall and f-measure to roc., informedness, markedness & correlation. *Journal of Machine Learning Technologies*, 2(1):37–63.
- Prechelt, L. (1998). Automatic early stopping using cross validation: quantifying the criteria. *Neural Networks*, 11(4):761–767.
- Prechelt, L. (1998). Early stopping: But when? *Lecture notes in computer science*, pages 55–69.
- Provost, F. (2000). Learning with imbalanced data sets. In *Invited paper for the AAAI2000 Workshop on Imbalanced Data Sets*.
- Provost, F. J. and Buchanan, B. G. (1995). Inductive policy: The pragmatics of bias selection. *Machine Learning*, 20(1-2):35–61.
- Quang, U. N., Nguyen, T. H., Nguyen, X. H., and O’Neill, M. (2010). Improving the generalisation ability of Genetic Programming with semantic similarity based crossover. In Esparcia-Alcazar, A. I., Ekart,

- A., Silva, S., Dignum, S., and Uyar, A. S., editors, *Proceedings of the 13th European Conference on Genetic Programming, EuroGP 2010*, volume 6021 of *LNCS*, pages 184–195, Istanbul. Springer.
- Quinlan, J. R. (1986). Induction of decision trees. *Mach. Learn.*, 1(1):81–106.
- Quinlan, J. R. (1996). Bagging, boosting, and c4.5. In *AAAI/IAAI, Vol. 1*, pages 725–730.
- R Development Core Team (2008). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0.
- Rechenberg, I. (1973). Evolution strategy: Optimization of technical systems by means of biological evolution. *Fromman-Holzboog, Stuttgart*, 104.
- Riekert, M., Malan, K. M., and Engelbrecht, A. P. (2009). Adaptive Genetic Programming for dynamic classification problems. In Tyrrell, A., editor, *2009 IEEE Congress on Evolutionary Computation*, pages 674–681, Trondheim, Norway. IEEE Computational Intelligence Society, IEEE Press.
- Rish, I. (2001). An empirical study of the naive bayes classifier. In *IJCAI 2001 workshop on empirical methods in artificial intelligence*, volume 3, pages 41–46.
- Rissanen, J. (1978). Modeling by shortest data description. *Automatica*, 14(5):465–471.
- Robilliard, D. and Fonlupt, C. (2001). Backwarding : An over-fitting control for Genetic Programming in a remote sensing application. In Collet, P., Fonlupt, C., Hao, J.-K., Lutton, E., and Schoenauer, M., editors, *Artificial Evolution 5th International Conference, Evolution Artificielle, EA 2001*, volume 2310 of *LNCS*, pages 245–254, Creusot, France. Springer Verlag.

- Rodriguez-Vazquez, K., Fonseca, C. M., and Fleming, P. J. (1997). Multiobjective Genetic Programming: A nonlinear system identification application. In *Late Breaking Papers at the 1997 Genetic Programming Conference*, pages 207–212.
- Rosca, J. (1996). Generality versus size in Genetic Programming. In Koza, J. R., Goldberg, D. E., Fogel, D. B., and Riolo, R. L., editors, *Genetic Programming 1996: Proceedings of the First Annual Conference*, pages 381–387, Stanford University, CA, USA. MIT Press.
- Rosenblatt, F. (1957). The perceptron—a perceiving and recognizing automaton. Technical report, Cornell Aeronautical Laboratory.
- Ryan, C. (1994). Pygmies and civil servants. In Kinnear, Jr., K. E., editor, *Advances in Genetic Programming*, chapter 11, pages 243–263. MIT Press.
- Ryan, C., Collins, J., and Neill, M. O. (1998). Grammatical evolution: Evolving programs for an arbitrary language. In *Genetic Programming*, pages 83–96. Springer.
- Ryan, C., Keijzer, M., and Cattolico, M. (2005). Favourable biasing of function sets using run transferable libraries. In *Genetic Programming Theory and Practice II*, pages 103–120. Springer.
- Sayed, S. (2012). An introduction to data mining.
- Schmiedle, F., Grosse, D., Drechsler, R., and Becker, B. (2001). Too much knowledge hurts: Acceleration of genetic programs for learning heuristics. In Reusch, B., editor, *Computational Intelligence : Theory and Applications*, volume 2206 of *LNCS*, pages 479–491, Dortmund, Germany. Springer-Verlag.
- Schmutter, P. (2002). *Object Oriented Ontogenetic Programming: Breeding Computer Programs that Work Like Multicellular Creatures*. Univ., Systems Analysis Research Group.
- Schwefel, H. (1977). Numerische optimierung von computermodellen mittels der evolutionsstrategie. 1977. *Basel, Stuttgart*, page 192.

- Shavlik, J. W. and Dietterich, T. G. (1990). *Readings in machine learning*. Morgan Kaufmann.
- Silva, S. and Costa, E. (2009). Dynamic limits for bloat control in Genetic Programming and a review of past and current bloat theories. *Genetic Programming and Evolvable Machines*, 10(2):141–179.
- Smart, W. and Zhang, M. (2005). Using Genetic Programming for multi-class classification by simultaneously solving component binary classification problems. Technical Report CS-TR-05-1, Computer Science, Victoria University of Wellington, New Zealand.
- Smart, W. R. and Zhang, M. (2003). Classification strategies for image classification in genetic programming. In Bailey, D., editor, *Proceeding of Image and Vision Computing NZ International Conference*, pages 402–407, Palmerston North, New Zealand. Massey University.
- Smith, J. M. and Price, G. (1973). The logic of animal conflict. *Nature*, 246:15.
- Sokolov, A. and Whitley, D. (2005). Unbiased tournament selection. In *Proceedings of the 2005 Conference on Genetic and Evolutionary Computation, GECCO '05*, pages 1131–1138, New York, NY, USA. ACM.
- Song, A., Loveard, T., and Ciesielski, V. (2001). Towards Genetic Programming for texture classification. In Stumptner, M., Corbett, D., and Brooks, M., editors, *Proceedings of the 14th International Joint Conference on Artificial Intelligence AI 2001: Advances in Artificial Intelligence*, volume 2256 of *Lecture Notes in Computer Science*, pages 461–472, Adelaide, Australia. Springer-Verlag.
- Song, D., Heywood, M. I., and Zincir-Heywood, A. N. (2005). Training Genetic Programming on half a million patterns: an example from anomaly detection. *IEEE Trans. Evolutionary Computation*, 9(3):225–239.

- Soule, T. and Foster, J. A. (1998). Effects of code growth and parsimony pressure on populations in Genetic Programming. *Evolutionary Computation*, 6(4):293–309.
- Spears, W. M. et al. (1992). Crossover or mutation. *Foundations of genetic algorithms*, 2:221–237.
- Stober, P. and Yeh, S.-T. (2007). An explicit functional form specification approach to estimate the area under a receiver operating characteristic (roc) curve. Available at <http://www2.sas.com/proceedings/sugi27/p226-227.pdf> Accessed March, 7.
- Storn, R. and Price, K. (1997). Differential evolution—a simple and efficient heuristic for global optimization over continuous spaces. *Journal of global optimization*, 11(4):341–359.
- Tacket, W. and Carmi, A. (1994). The donut problem: scalability, generalization and breeding policies in Genetic Programming. *Advances in Genetic Programming*, pages 143–176.
- Tackett, W. A. (1993). Genetic Programming for feature discovery and image discrimination. In Forrest, S., editor, *Proceedings of the 5th International Conference on Genetic Algorithms, ICGA-93*, pages 303–309, University of Illinois at Urbana-Champaign. Morgan Kaufmann.
- Thammano, A. and Moolwong, J. (2007). Classification algorithm based on human social behavior. In *Proceedings of the 7th IEEE International Conference on Computer and Information Technology*, pages 105–109, Washington, DC, USA. IEEE Computer Society.
- Thierens, D. (1998). Selection schemes, elitist recombination, and selection intensity. *UU-CS*, (1998-48).
- Thierens, D. (2005). An adaptive pursuit strategy for allocating operator probabilities. In *Proceedings of the 2005 conference on Genetic and evolutionary computation*, pages 1539–1546. ACM.
- Thomas, J. D. and Sycara, K. (1999). The importance of simplicity and validation in Genetic Programming for data mining in financial data.

- In Freitas, A. A., editor, *Data Mining with Evolutionary Algorithms: Research Directions*, pages 7–11, Orlando, Florida. AAAI Press. Technical Report WS-99-06.
- Triola, M. F. (1995). *Elementary Statistics*. Addison-Wesley, 6 edition.
- Tsakonas, A. (2006). A comparison of classification accuracy of four genetic programming-evolved intelligent structures. *Inf. Sci.*, 176:691–724.
- Tuite, C., Agapitos, A., O’Neill, M., and Brabazon, A. (2011). A preliminary investigation of overfitting in evolutionary driven model induction: Implications for financial modelling. In Di Chio, C., Brabazon, A., Di Caro, G., Drechsler, R., Ebner, M., Farooq, M., Grahl, J., Greenfield, G., Prins, C., Romero, J., Squillero, G., Tarantino, E., Tettamanzi, A. G. B., Urquhart, N., and Uyar, A. S., editors, *Applications of Evolutionary Computing, EvoApplications 2011: EvoCOMNET, EvoFIN, EvoHOT, EvoMUSART, EvoSTIM, EvoTRANSLOG*, volume 6625 of *LNCS*, pages 121–130, Turin, Italy. Springer Verlag.
- Urbanowicz, R. J. and Moore, J. H. (2009). Learning classifier systems: a complete introduction, review, and roadmap. *Journal of Artificial Evolution and Applications*, 2009:1.
- Valentini, G. and Dietterich, T. G. (2004). Bias-variance analysis of support vector machines for the development of svm-based ensemble methods. *The Journal of Machine Learning Research*, 5:725–775.
- Valiant, L. G. (1984). A theory of the learnable. volume 27, pages 1134–1142, New York, NY, USA. ACM.
- Valiant, L. G. (2009). Evolvability. *Journal of the ACM (JACM)*, 56(1):3.
- Van Der Putten, P. and Van Someren, M. (2004). A bias-variance analysis of a real world learning problem: The coil challenge 2000. *Machine Learning*, 57(1-2):177–195.
- Vanneschi, L., Castelli, M., and Silva, S. (2010a). Measuring bloat, overfitting and functional complexity in genetic programming. In Branke,

- J., Pelikan, M., Alba, E., Arnold, D. V., Bongard, J., Brabazon, A., Branke, J., Butz, M. V., Clune, J., Cohen, M., Deb, K., Engelbrecht, A. P., Krasnogor, N., Miller, J. F., O'Neill, M., Sastry, K., Thierens, D., van Hemert, J., Vanneschi, L., and Witt, C., editors, *GECCO '10: Proceedings of the 12th annual conference on Genetic and evolutionary computation*, pages 877–884, Portland, Oregon, USA. ACM.
- Vanneschi, L., Castelli, M., and Silva, S. (2010b). Measuring bloat, overfitting and functional complexity in genetic programming. In *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, GECCO '10, pages 877–884, New York, NY, USA. ACM.
- Vanneschi, L. and Gustafson, S. (2009). Using crossover based similarity measure to improve genetic programming generalization ability. In Raidl, G., Rothlauf, F., Squillero, G., Drechsler, R., Stuetzle, T., Bittanti, M., Congdon, C. B., Middendorf, M., Blum, C., Cotta, C., Bosman, P., Grahl, J., Knowles, J., Corne, D., Beyer, H.-G., Stanley, K., Miller, J. F., van Hemert, J., Lenaerts, T., Ebner, M., Bacardit, J., O'Neill, M., Di Penta, M., Doerr, B., Jansen, T., Poli, R., and Alba, E., editors, *GECCO '09: Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 1139–1146, Montreal. ACM.
- Vladislavleva, E. J., Smits, G. F., and Den Hertog, D. (2009a). Order of nonlinearity as a complexity measure for models generated by symbolic regression via pareto Genetic Programming. *Trans. Evol. Comp*, 13:333–349.
- Vladislavleva, E. J., Smits, G. F., and Den Hertog, D. (2009b). Order of nonlinearity as a complexity measure for models generated by symbolic regression via pareto Genetic Programming. *Trans. Evol. Comp*, 13:333–349.
- Wah, B. W. (1999). Generalization and generalizability measures. *Knowledge and Data Engineering, IEEE Transactions on*, 11(1):175–186.

- Weihls, C. and Gaul, W., editors (2005). *Classification - the Ubiquitous Challenge, Proceedings of the 28th Annual Conference of the Gesellschaft für Klassifikation e.V., University of Dortmund, March 9-11, 2004*, Studies in Classification, Data Analysis, and Knowledge Organization. Springer.
- Whigham, P. A. (1996). Search bias, language bias and Genetic Programming. In *Proceedings of the First Annual Conference on Genetic Programming*, 1996, pages 230–237, Cambridge, MA, USA. MIT Press.
- White, D. R. and Poulding, S. (2009). A rigorous evaluation of crossover and mutation in genetic programming. In *Proceedings of the 12th European Conference on Genetic Programming, EuroGP '09*, pages 220–231, Berlin, Heidelberg. Springer-Verlag.
- Whitley, D., Richards, M., Beveridge, R., and da Motta Salles Barreto, A. (2006). Alternative evolutionary algorithms for evolving programs: Evolution strategies and steady state gp. In *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation, GECCO '06*, pages 919–926, New York, NY, USA. ACM.
- Wilke, C. O., Wang, J. L., Ofria, C., Lenski, R. E., and Adami, C. (2001). Evolution of digital organisms at high mutation rates leads to survival of the flattest. *Nature*, 412(6844):331–333.
- Williams, N., Zander, S., and Armitage, G. (2006). A preliminary performance comparison of five machine learning algorithms for practical ip traffic flow classification. *ACM SIGCOMM Computer Communication Review*, 36(5):5–16.
- Winkler, S. M., Affenzeller, M., and Wagner, S. (2006). Using enhanced Genetic Programming techniques for evolving classifiers in the context of medical diagnosis - an empirical study. In Smith, S. L., Cagnoni, S., and van Hemert, J., editors, *MedGEC 2006 GECCO Workshop on Medical Applications of Genetic and Evolutionary Computation*, Seattle, WA, USA.

- Wolpert, D. H. and Macready, W. G. (1995). No free lunch theorems for search. Technical report, Technical Report SFI-TR-95-02-010, Santa Fe Institute.
- Wolpert, D. H. and Macready, W. G. (1997). No free lunch theorems for optimization. *Evolutionary Computation, IEEE Transactions on*, 1(1):67–82.
- Wright, S. (1932). The roles of mutation, inbreeding, crossbreeding and selection in evolution. In *Proceedings of the sixth international congress on genetics*, volume 1, pages 356–366.
- Xie, H. (2008). *An Analysis of Selection in Genetic Programming*. PhD thesis, Computer Science, Victoria University of Wellington, New Zealand.
- Xie, H. and Zhang, M. (2009). *Tuning Selection Pressure in Tournament Selection*. School of Engineering and Computer Science, Victoria University of Wellington.
- Xie, H., Zhang, M., and Andrae, P. (2006). Automatic selection pressure control in Genetic Programming. In Yang, B. and Chen, Y., editors, *6th International Conference on Intelligent System Design and Applications*, pages 435–440, Jinan Nanjiao Hotel, Jinan, China. IEEE.
- Xie, H., Zhang, M., and Andrae, P. (2007). Not-sampled issue and round-replacement tournament selection. Technical report CS-TR-07-2, Computer Science, Victoria University of Wellington, New Zealand.
- Yan, L., Dodier, R., Mozer, M. C., and Wolniewicz, R. (2003). Optimizing classifier performance via the wilcoxon-mann-whitney statistics. In *Proceedings of the 20th international conference on machine learning*, pages 848–855. Citeseer.
- Yang, X.-S. and Deb, S. (2009). Cuckoo search via levy flights. In *Nature & Biologically Inspired Computing, 2009. NaBIC 2009. World Congress on*, pages 210–214. IEEE.

- Yang, X.-S. and Deb, S. (2010). Eagle strategy using levy walk and firefly algorithms for stochastic optimization. In *Nature Inspired Cooperative Strategies for Optimization (NICSO 2010)*, pages 101–111. Springer.
- Yin, Z., Brabazon, A., and O’Sullivan, C. (2007). Adaptive Genetic Programming for option pricing. In Bosman, P. A. N., editor, *Late breaking paper at Genetic and Evolutionary Computation Conference (GECCO’2007)*, pages 2588–2594, London, United Kingdom. ACM Press.
- Zhang, B.-T. and Mühlenbein, H. (1995). Balancing accuracy and parsimony in Genetic Programming. *Evolutionary Computation*, 3(1):17–38.
- Zhang, M. and Ciesielski, V. (1999). Genetic Programming for multiple class object detection. In Foo, N., editor, *12th Australian Joint Conference on Artificial Intelligence*, volume 1747 of *LNAI*, pages 180–192, Sydney, Australia. Springer-Verlag.
- Zhang, M. and Smart, W. (2004). Multiclass object classification using Genetic Programming. Technical Report CS-TR-04-2, Computer Science, Victoria University of Wellington, New Zealand.
- Zhou, C., Nelson, P. C., Xiao, W., and Tirpak, T. M. (2002). Discovery of classification rules by using gene expression programming. In *Proceedings of the International Conference on Artificial Intelligence (IC-AI’02)*, pages 1355–1361, Las Vegas, U.S.A.
- Zikopoulos, P., Eaton, C., et al. (2011). *Understanding big data: Analytics for enterprise class hadoop and streaming data*. McGraw-Hill Osborne Media.
- Zweig, M. H. and Campbell, G. (1993). Receiver-operating characteristic (roc) plots: a fundamental evaluation tool in clinical medicine. *Clinical chemistry*, 39(4):561–577.