Nova Southeastern University
NSUWorks

CEC Theses and Dissertations | College of Engineering and Computing

2016

# Automatically Defined Templates for Improved Prediction of Non-stationary, Nonlinear Time Series in Genetic Programming

David Moskowitz
*Nova Southeastern University*, dave@infoblazer.com

This document is a product of extensive research conducted at the Nova Southeastern University College of Engineering and Computing. For more information on research and degree programs at the NSU College of Engineering and Computing, please click here.

Follow this and additional works at: http://nsuworks.nova.edu/gscis_etd

Part of the Artificial Intelligence and Robotics Commons, Finance and Financial Management Commons, Software Engineering Commons, and the Theory and Algorithms Commons

Share Feedback About This Item

NSUWorks Citation
David Moskowitz. 2016. *Automatically Defined Templates for Improved Prediction of Non-stationary, Nonlinear Time Series in Genetic Programming.* Doctoral dissertation. Nova Southeastern University. Retrieved from NSUWorks, College of Engineering and Computing. (953)
http://nsuworks.nova.edu/gscis_etd/953.

This Dissertation is brought to you by the College of Engineering and Computing at NSUWorks. It has been accepted for inclusion in CEC Theses and Dissertations by an authorized administrator of NSUWorks. For more information, please contact nsuworks@nova.edu.

# Automatically Defined Templates for Improved Prediction of Non-stationary, Nonlinear Time Series in Genetic Programming

by

David Moskowitz

A dissertation submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy
in
Computer Science

College of Engineering and Computing
Nova Southeastern University

2016

We hereby certify that this dissertation, submitted by David Moskowitz, conforms to acceptable standards and is fully adequate in scope and quality to fulfill the dissertation requirements for the degree of Doctor of Philosophy.

| | |
|---|---|
| Sumitra Mukherjee, Ph.D.<br>Chairperson of Dissertation Committee | Date |
| Francisco J. Mitropoulos, Ph.D.<br>Dissertation Committee Member | Date |
| Michael J. Laszlo, Ph.D.<br>Dissertation Committee Member | Date |

Approved:

| | |
|---|---|
| Amon B. Seagull, Ph.D.<br>Interim Dean, College of Engineering and Computing | Date |

SIGNED ORIGINAL ON FILE AT NSU

College of Engineering and Computing
Nova Southeastern University

2016

An Abstract of a Dissertation Submitted to Nova Southeastern University
in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy

# Automatically Defined Templates for Improved Prediction of Non-stationary, Nonlinear Time Series in Genetic Programming

by
David Moskowitz
2016

Soft methods of artificial intelligence are often used in the prediction of non-deterministic time series that cannot be modeled using standard econometric methods. These series, such as occur in finance, often undergo changes to their underlying data generation process resulting in inaccurate approximations or requiring additional human judgment and input in the process, hindering the potential for automated solutions.

Genetic programming (GP) is a class of nature-inspired algorithms that aims to evolve a population of computer programs to solve a target problem. GP has been applied to time series prediction in finance and other domains. However, most GP-based approaches to these prediction problems do not consider regime change.

This paper introduces two new genetic programming modularity techniques, collectively referred to as automatically defined templates, which better enable prediction of time series involving regime change. These methods, based on earlier established GP modularity techniques, take inspiration from software design patterns and are more closely modeled after the way humans actually develop software. Specifically, a regime detection branch is incorporated into the GP paradigm. Regime specific behavior evolves in a separate program branch, implementing the template method pattern.

A system was developed to test, validate, and compare the proposed approach with earlier approaches to GP modularity. Prediction experiments were performed on synthetic time series and on the S&P 500 index. The performance of the proposed approach was evaluated by comparing prediction accuracy with existing methods.

One of the two techniques proposed is shown to significantly improve performance of time series prediction in series undergoing regime change. The second proposed technique did not show any improvement and performed generally worse than existing methods or the canonical approaches. The difference in relative performance was shown to be due to a decoupling of reusable modules from the evolving main program population. This observation also explains earlier results regarding the inferior performance of genetic programming techniques using a similar, decoupled approach. Applied to financial time series prediction, the proposed approach beat a buy and hold return on the S&P 500 index as well as the return achieved by other regime aware genetic programming methodologies. No approach tested beat the benchmark return when factoring in transaction costs.

## Acknowledgements

I would like to thank my dissertation committee members, Dr. Michael Laszlo and Dr. Francisco Mitropoulos, for their review and feedback of this work. I would especially like to acknowledge the advice, input, and guidance I received from my dissertation advisor, Dr. Sumitra Mukherjee, for the past several years.

Most importantly, to my wife Anna and son Gabriel, for their support, sacrifices, and inspiration over the past six years of being a "PhD family".

This dissertation was made possible by copious amounts of energy drinks and large doses of heavy metal.

# Table of Contents

List of Figures ... vii
List of Tables ... xvi
Chapter 1: Introduction ... 1
Background ... 2
Problem Statement ... 6
Dissertation Goals ... 7
Research Questions ... 8
Relevance and Significance ... 9
Barriers and Issues ... 9
Assumptions, Limitations, and Delimitations ... 11
Definitions of Terms ... 12
Summary ... 13
Chapter 3: Review of the Literature ... 15
Time Series Prediction and the Stock Market ... 15
Econometric Methods for Time Series Prediction ... 18
Artificial Intelligence Methods for Time Series Prediction ... 25
Neural networks. ... 25
Genetic algorithms. ... 27
Genetic programming. ... 29
DyFor GP. ... 38
Modularity in genetic programming. ... 42
Patterns in Software Engineering ... 44
Chapter 3: Methodology ... 48
Overview ... 48
Fitness and Selection ... 55
Decoupled approach to selection. ... 55
Coupled approach to selection. ... 57
Fitness calculation. ... 58
Automatically defined templates ... 59
Crossover. ... 60

Automatically acquired templates. 60
Crossover. 61
Sufficiency 62
Specific Methodology 62
Automatically defined template example. 63
Automatically acquired templates example. 65
Implementation. 69
Optimizations. 73
Omniscient regime detection. 75
Experiments. 76
Measuring performance. 78
Resource Requirements 81
Summary 82
Chapter 4: Results 83
Synthetic Series 83
Data analysis. 84
Findings. 93
Summary. 108
Market Data Series 109
Data analysis. 109
Findings. 118
Summary. 125
Summary of Results 126
Chapter 5: Conclusions, Implications, Recommendations, and Summary 127
Conclusions 127
Implications 129
Recommendations for Future Work 130
Adaptive training and dynamic training generations. 130
Optimization of parameters. 131
Computational optimization. 131
Improving market data prediction. 133
Summary 134
Appendix A 137

Design Pattern Example ... 137
Appendix B ... 146
Stock Prediction Example ... 146
Java implementation. ... 147
Clojure implementation. ... 150
Appendix C ... 152
Primitives ... 152
Functions. ... 152
Terminals. ... 154
Appendix D ... 156
Program Parameters ... 156
Parameter descriptions. ... 156
Parameter files. ... 162
Appendix E ... 178
Best Symbolic Regression Programs ... 178
Appendix F ... 185
MGHENMG Initial Random Parameters ... 185
Appendix G ... 186
Full Results ... 186
SINCOS. ... 186
LGOZLG ... 190
MGHENMG ... 200
S&P 500 market data prediction. ... 205
Appendix H ... 215
Best S&P 500 Prediction Programs ... 215
Transaction costs considered. ... 215
No transaction costs. ... 230
Appendix I ... 248
Raw Data Files ... 248
References ... 249

## List of Figures

**Figures**

Figure 1. Series created by superimposing a Hénon map on a Mackey-Glass series. ........ 4

Figure 2. S&P 500 index close price during the stock market crash of 2008 .................... 4

Figure 3. Overlapping training period approach used by Yu et al. .................................. 17

Figure 4. VIX index ........................................................................................................ 21

Figure 5. Markov state transition diagram. ..................................................................... 24

Figure 6. Sample genetic algorithm chromosome used by Canelas et al. ......................... 28

Figure 7. Average historical NYSE transaction costs ..................................................... 34

Figure 8. The Backus-Naur grammar used in EDDIE 7 ................................................. 35

Figure 9. Nonlinear equations analyzed by Chen & Ye .................................................. 36

Figure 10. DyFor GP sliding window example, 1 of 5. ................................................... 38

Figure 11. DyFor GP sliding window example, 2 of 5. ................................................... 39

Figure 12. DyFor GP sliding window example, 3 of 5. ................................................... 39

Figure 13. DyFor GP sliding window example, 4 of 5. ................................................... 39

Figure 14. DyFor GP sliding window example, 5 of 5. ................................................... 39

Figure 15. Template method pattern class diagram ......................................................... 46

Figure 16. Strategy pattern class diagram ....................................................................... 47

Figure 17. Canonical genetic programming flow. ........................................................... 49

Figure 18. Program tree including an ADF branch .......................................................... 50

Figure 19. Regime determining program branch tree structure. ...................................... 51

Figure 20. ADT tree structure. ......................................................................................... 52

Figure 21. ADT algorithm flow chart. ............................................................................. 54

Figure 22. Tree structure for coupled ADT. .................................................................. 58

Figure 23. Sample regime determining program tree. ..................................................... 63

Figure 24. Program tree incorporating an abstract method.............................................. 64

Figure 25. Regime specific template implementations.................................................... 65

Figure 26. Module acquisition compression operation.................................................... 66

Figure 27. Program tree prior to AAT extraction. ........................................................... 67

Figure 28. Program tree after AAT extraction................................................................. 67

Figure 29. Function tree extracted by AAT. .................................................................... 68

Figure 30. Regime specific template implementations created by AAT extraction. ........ 69

Figure 31. Primitives set used in GP example. ................................................................. 70

Figure 32. Tree structure representing the sample Clojure expression (+ (+ 5 x) 5). ....... 71

Figure 33. Java implementation of a sample S-expression tree........................................ 71

Figure 34. LG-OZ-LG synthetic time series used by Wagner & Michalewicz. ............... 77

Figure 35. MG-HEN-MG synthetic time series used by Wagner & Michalewicz. .......... 77

Figure 36. SINCOS synthetic time series. ........................................................................ 85

Figure 37. LGOZLG synthetic time series. ...................................................................... 87

Figure 38. MGHENMG synthetic time series. ................................................................. 89

Figure 39. Average fitness for SINCOS symbolic regression experiment. ...................... 95

Figure 40. Relative performance of ADT vs. DyFor GP in LGOZLG experiment. ......... 98

Figure 41. Relative performance of ADT vs. DyFor GP in MGHENMG experiment... 102

Figure 42. Relative performance of ADT vs DyFor GP in LGOZLG validation tests. 105

Figure 43. Relative performance of ADT vs. DyFor GP in MGHENMG experiment... 106

Figure 44. S&P 500 index.............................................................................................. 110

Figure 45. S&P 500 index during study period. .......... 110

Figure 46. S&P 500 index during prediction period. .......... 111

Figure 47. S&P 500 normalized series .......... 112

Figure 48. Overlapping training, validation, and testing periods used by Chen et al. .... 113

Figure 49. 3-month Treasury bill historical yield .......... 114

Figure 50. Total return calculation algorithm used in market data experiments. .......... 115

Figure 51. 95% CI for market experiments. .......... 120

Figure 52. 95% CI for sliding window market experiments. .......... 121

Figure 53. 95% CI for market experiments without transaction cost .......... 123

Figure 54. 95% CI for sliding window experiments without transaction costs. .......... 124

Figure 55. Program tree containing an intron. .......... 132

Figure A1. Sample racing domain. .......... 137

Figure A2. Initial implementation of Automobile class in autocross example. .......... 138

Figure A3. Initial implementation of Motorcycle class in autocross example. .......... 139

Figure A4. Turn method for Automobile class in autocross example. .......... 139

Figure A5. Turn method for Motorcycle class in autocross example. .......... 140

Figure A6. Race method for Automobile class in autocross example. .......... 140

Figure A7. Race method for Motorcycle class in autocross example. .......... 140

Figure A8. Object model showing one parent and two subclasses. .......... 141

Figure A9. Methods in abstract parent class. .......... 141

Figure A10. Additional methods in abstract parent class. .......... 141

Figure A11. Race method in abstract class. .......... 141

Figure A12. Full class diagram for abstract and concrete classes. ................................ 142

Figure A13. Java code for abstract parent class........................................................... 142

Figure A14. Java code for Automobile concrete class. ................................................. 143

Figure A15. Java code for Motorcycle concrete class. .................................................. 143

Figure A16. Object model for strategy pattern implementation. ................................... 144

Figure A17. Java code for applying strategy pattern in concrete Automobile class....... 145

Figure B1. Example investment model logic. ............................................................... 147

Figure B2. Java method for example investment decision ........................................... 148

Figure B3. Class diagram for template method pattern implementation ....................... 148

Figure B4. Regime indicator implementation. .............................................................. 148

Figure B5. Investment decision Java interface. ............................................................. 149

Figure B6. Template method Java implementation. ..................................................... 149

Figure B7. Regime specific logic implementing abstract template methods................. 149

Figure B8. Final program using template method pattern. ........................................... 150

Figure B9. Clojure implementation of investment decision example............................ 150

Figure B10. Clojure implementation incorporating modularity features....................... 151

Figure C1. Primitive functions class diagram. .............................................................. 152

Figure C2. Primitive terminals class diagram. ............................................................... 154

Figure D1. ADT LGOZLG parameters ......................................................................... 162

Figure D2. ADT Omni LGOZLG parameters. ............................................................. 163

Figure D3. AAT LGOZLG parameters. ........................................................................ 164

Figure D4. AAT Omni LGOZLG parameters. ............................................................ 165

Figure D5. DyFor GP LGOZLG parameters. ............................................................ 165

Figure D6. ADT MGHENMG parameters. ............................................................ 166

Figure D7. ADT Omni MGHENMG parameters. ............................................................ 167

Figure D8. AAT MGHENMG parameters. ............................................................ 168

Figure D9. AAT Omni MGHENMG parameters. ............................................................ 169

Figure D10. DyFor GP MGHENMG parameters. ............................................................ 170

Figure D11. GP SINCOS parameters. ............................................................ 170

Figure D12. ADF SINCOS Parameters ............................................................ 171

Figure D13. ADT SINCOS parameters. ............................................................ 172

Figure D14. AAT SINCOS parameters. ............................................................ 172

Figure D15. ADF S&P 500 prediction sample parameter file ....................................... 173

Figure D16. ADT S&P 500 prediction sample parameter file ....................................... 174

Figure D17. GP S&P 500 prediction sample parameter file ....................................... 175

Figure D18. ADT sliding window S&P 500 prediction sample parameter file ............. 176

Figure D19. DyFor GP S&P 500 prediction sample parameter file .............................. 177

Figure E1. Best Regression Program recorded by GP approach. .................................. 178

Figure E2. Best symbolic regression program recorded by ADF approach. .................. 179

Figure E3. Best symbolic regression program recorded by ADT approach. .................. 180

Figure E4. Best symbolic regression program recorded by AAT approach. .................. 181

Figure E5. Best symbolic regression program recorded by coupled ADT approach. .... 183

Figure E6. Best symbolic regression program recorded by coupled AAT approach. .... 184

Figure G1. Best fitness in symbolic regression SINCOS experiments. .......................... 186

Figure G2. Best regressions in SINCOS experiments. ................................................. 188

Figure G3. Best regime regressions in SINCOS experiments. ...................................... 189

Figure G4. Average training fitness in LGOZLG experiments. .................................... 190

Figure G5. Average training fitness in LGOZLG 1-100 experiments. .......................... 190

Figure G6. Average error in LGOZLG experiments. ................................................... 192

Figure G7. Average error in LGOZLG 1-100 experiments. ......................................... 193

Figure G8. LGOZLG regime determination by best performing individuals. ................ 194

Figure G9. LGOZLG 1-100 regime determination by best performing individuals. ..... 195

Figure G10. LGOZLG best prediction. ......................................................................... 198

Figure G11. LGOZLG 1-100 best prediction. .............................................................. 199

Figure G12. MGHENMG average training fitness. ...................................................... 200

Figure G13. MGHENMG  30-130 average training fitness. ......................................... 200

Figure G14. MGHENMG regime determination by best performing individuals. ......... 201

Figure G15. MGHEMMG 1-100 regime determination best performing individuals. .. 202

Figure G16. MGHENMG best prediction. .................................................................... 203

Figure G17. MGHENMG 1-100 best prediction. ......................................................... 204

Figure G18. S&P 500 investment performance. ........................................................... 207

Figure G19. Investment performance in S&P 500 sliding window experiments. .......... 208

Figure G20. S&P 500 investment performance with transaction costs ignored. ............ 212

Figure G21. Performance in S&P 500 sliding window tests w/o transaction costs ....... 213

Figure H1. Best performing approach for 1999-2000 period with transaction costs...... 216

Figure H2. Regime predicted by ADT for 1999-2000 period with transaction costs..... 216

Figure H3. Regime predicted by ADT for 1999-2000 period with transaction costs plotted against normalized S&P 500 index. ........................................................................... 217

Figure H4. Best Performing GP Program for 1999-2000 period with transaction costs. 217

Figure H5. Best ADT program for 1999-2000 period with transaction costs. ............... 218

Figure H6. Best ADT regime program for 1999-2000 period with transaction costs. ... 220

Figure H7. Best performing approach for 2001-2002 period with transaction costs...... 221

Figure H8. Regime predicted by best approach for 2001-2002 with transaction costs. . 221

Figure H9. Regime predicted by best approach for 2001-2002 with transaction costs plotted against normalized target series. ......................................................................... 222

Figure H10. Best performing ADT program for 2001-2002 with transaction costs....... 223

Figure H11. Regime predicted by ADT for 2001-2002 period with transaction costs... 224

Figure H12. Best performing approach for 2003-2004 period with transaction costs.... 225

Figure H13. Regime predicted by best approach for 2003-2004 with transaction costs. 225

Figure H14. Regime predicted by best approach for 2003-2004 with transaction costs plotted against normalized target series. ......................................................................... 226

Figure H15. Best performing ADF program for 2003-2004 with transaction costs. ...... 227

Figure H16. Best performing ADT program for 2003-2004 with transaction costs....... 227

Figure H17. Regime program for ADT for 2003-2004 with transaction costs.............. 228

Figure H18. Best returns for 1999-2014 period with transaction costs. ......................... 229

Figure H19. Regime predicted by ADT for 1999-2004 with transaction costs. ............ 229

Figure H20. Regime predicted by ADT for 1999-2004 with transaction costs, plotted against normalized target series. .......... 230

Figure H21. Best performing approach for 1999-2000 period w/o transaction costs..... 231

Figure H22. Regime predicted by ADT for 1999-2000 period w/o transaction costs.... 231

Figure H23. Regime predicted by ADT for 1999-2000 period w/o transaction costs plotted against normalized target series .......... 232

Figure H24. Best Performing ADF Program for 1999-2000 without transaction costs. 233

Figure H25. Best performing ADT program for 1999-2000 without transaction costs.. 235

Figure H26. Best ADT regime program for 1999-2000 without transaction costs. ....... 235

Figure H27. Best performing approach for 2001-2002 period w/o transaction costs..... 237

Figure H28. Regime predicted by best approach for 2001-2002 w/o transaction costs. 237

Figure H29. Regime predicted by best approach for 2001-2002 w/o transaction costs, plotted against normalized target series. .......... 238

Figure H30. Best ADF program for 2001-2002 period without transaction costs. ........ 238

Figure H31. Best ADT program for 2001-2002 period without transaction costs. ........ 240

Figure H32. Regime predicted by ADT for 2001-2002 period w/o transaction costs.... 241

Figure H33. Best performing approach for 2003-2004 period w/o transaction costs..... 243

Figure H34. Regime predicted by best approach for 2003-2004 w/o transaction costs. 243

Figure H35. Regime predicted by best approach for 2003-2004 without transaction costs, plotted against normalized target series. .......... 244

Figure H36. Best performing ADT program for 2003-2004 without transaction costs.. 245

Figure H37. Regime program for ADT for 2003-2004 without transaction costs.......... 246

Figure H38. Best returns for 1999-2014 period without transaction costs. .................... 246

Figure H39. Regime predicted by ADT for 1999-2004 without transaction costs. ........ 247

Figure H40. Regime predicted by ADT for 1999-2004 without transaction costs, plotted against normalized series. ........................................................................................................ 247

# List of Tables

**Tables**


Table 1. Regime Determination Using Two Indicators ... 63

Table 2. Sample Fitness Evaluation Using Mean Error ... 72

Table 3. Common Parameters Used in Experiments ... 90

Table 4. ADT Parameters ... 90

Table 5. AAT Parameters ... 90

Table 6. DyFor GP Parameters ... 91

Table 7. SINCOS Symbolic Regression Results ... 94

Table 8. LGOZLG Prediction Results ... 95

Table 9. LGOZLG Total Evaluations ... 99

Table 10. MGHENMG Prediction Results ... 100

Table 11. MGHENMG Total Evaluations ... 102

Table 12. Results for LGOZLG and MGHENMG Validation Tests ... 104

Table 13. Results Reported in Wagner & Michalewicz Benchmark Experiment ... 107

Table 14. Average Population Node Counts ... 108

Table 15. Experimental Parameters Used in Market Prediction Tests ... 117

Table 16. DyFor GP Parameters Used in Market Data Experiments ... 118

Table 17. Market Data Experiment Results Including Transaction Costs ... 119

Table 18. Sliding Window Market Data Experiment Results With Transaction Costs .. 120

Table 19. Findings Reported by Chen, Kuo, & Hoi ... 121

Table 20. Market Data Experiment Results Not Including Transaction Costs ... 122

Table 21. Sliding Window Market Data Experiment Results w/o Transaction Costs .... 124

Table C1. Function Primitives ........................................................................................ 153

Table C2. Terminal Primitives........................................................................................ 155

Table D1. Common Parameters Used in Experiments .................................................. 156

Table D2. ADT/AAT Parameters .................................................................................... 160

Table D3. ADT Parameters............................................................................................. 161

Table D4. AAT Parameters............................................................................................. 161

Table D5. DyFor GP Parameters .................................................................................... 161

Table F1. MGHENMG Initialization Parameters.......................................................... 185

Table G1. Number of Trades Executed, Including Transaction Costs .......................... 209

Table G2. Number of Trades Executed, Not Including Transaction Costs ................... 214

Table H1. Best Return, 1999-2000, With Transaction Costs ........................................ 215

Table H2. Best Return, 2001-2002, With Transaction Costs ........................................ 220

Table H3. Best Return, 2003-2004, With Transaction Costs ........................................ 224

Table H4. Best Return, 1999-2004, With Transaction Costs ........................................ 228

Table H5. Best Return, 1999-2000, Without Transaction Costs ................................... 230

Table H6. Best Return, 2001-2002, Without Transaction Costs ................................... 236

Table H7. Best Return, 2003-2004, Without Transaction Costs ................................... 242

Table H8. Best Return, 1999-2004, Without Transaction Costs ................................... 246

# Chapter 1

# Introduction

A stated aim of artificial intelligence is "to get machines to exhibit behavior, which if done by humans would be assumed to involve the use of intelligence"- Arthur Samuel, quoted in (Keane, Streeter, Mydlowec, Lanza, & Yu, 2006, p. 3). Samuel's view of artificial intelligence appears to promote a black box approach, where only the external results of a process are considered. An example of such an approach is the iconic Turing Test, where a human interrogator queries a human and a computer (without knowing which is which), trying to ascertain the correct identities of each (Turing, 1950). However, a peek behind the curtain would reveal that the intelligent machine that just successfully mimicked a human is only a simple program regurgitating the interrogator's questions or offering canned responses. The equally iconic ELIZA (Weizenbaum, 1966) is an example of one such "intelligent" machine. Inarguably, Turing's question in his famous paper (Turing, 1950), "can machines think", would certainly be answered "not in this case". But this realization is often only evident when removing the curtain and taking a white box approach to artificial intelligence—examining the internals as well as the external results of the process.

Genetic programming (GP), a subset of a broader class of evolutionary algorithms, is a method for automatically generating computer programs using biological evolution inspired operators with little thought to structure of the solutions, other than their fitness (Koza, 1992). Though this description seems to imply a black box approach, progress in the field of genetic programming since its inception has shown this is not necessarily the case.

A widely acknowledged advantage of evolutionary algorithms over other artificial intelligence based search techniques, such as neural networks, is the ease of interpreting and

understanding the result solutions. Koza, Streeter, & Keane (2008), describes numerous results from genetic programming that have replicated existing patents in the area of controller design, indicating the transparent nature of the solutions. While the earliest examples of genetically evolved programs were, though syntactically correct, far from what a human programmer would produce, subsequent research incorporated more human-like behavior. Many of the advances in the field have been achieved by incorporating additional elements often associated with language design, such as functions (Koza, 1994), shared code modules (Angeline, 1994) , and recursion (Yu & Clack, 1998).

Modeling genetic programming internals on how humans actually program has been an ongoing research trend (Woodward, 2003). Incorporating additional software engineering methods into the genetic programming paradigm should further improve performance of this artificial intelligence technique and move further towards the true goal of AI as stated by Samuel. While past research has focused on incorporating features of programming languages into genetic programming, there is potential for further improvements by incorporating non-language elements such as software design patterns.

**Background**

Evolutionary and other artificial intelligence approaches are often used in time series prediction problems. In many cases, no deterministic solution to these problems exists, so non-deterministic methods, such as artificial neural network or genetic algorithms, are often applied (Srinivas & Patnaik, 1994). Furthermore, many time series, especially financial time series, involve regime change, where the underlying data generation processes may abruptly change. Financial time series regimes can change due to political or economic events, such as the US

housing crisis of the late 2000s, or Federal Reserve policy changes, such as the current low interest rate environment.

Forecasting algorithms often adopt an autoregressive approach, generating prediction rules by analyzing historical values of the time series to be predicted. The entire series history, or only a portion of that history, can be analyzed. The choice of analysis window size is often made due to data volume considerations. As time series can involve a large number of data points, analyzing the entire history may prove unfeasible. In either approach, most algorithms will produce a single prediction, hopefully applicable to any past or future period. As time series can involve distinct periods when underlying data generation processes change, a “one size fits all" result may be insufficient and only produce average results across multiple regimes. Most existing time series forecasting methods do not consider regime change situation (Wagner, Khouja, Michalewicz, & McGregor, 2008). Those methods that do consider regime change generally require additional human input and judgment, forcing the analyst to choose the appropriate time window for analysis and limiting potential automated solutions (Wagner et al., 2007, p. 2).

A fabricated example of a change in the underlying data generation process presented in (Wagner & Michalewicz, 2008) and is reproduced in Figure 1. This series is created by superimposing  a Hénon map series (Hénon, 1976) on a Mackey-Glass series (Mackey & Glass, 1977).

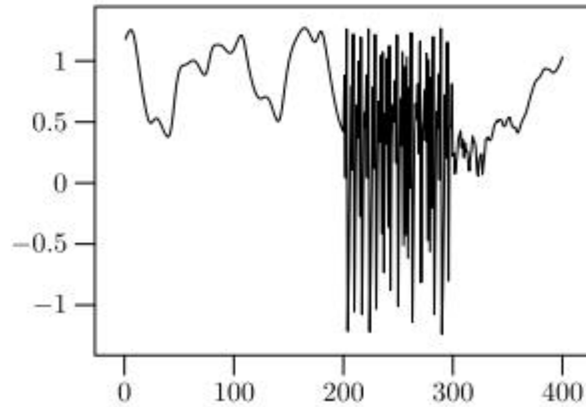



*Figure 1. S*ynthetic series created by superimposing a Hénon map on a Mackey-Glass series. Taken from (Wagner & Michalewicz, 2008).

A real world example of regime change is the stock market crash of 2008 which resulted in a 50% drop followed by a 50% gain in the S&P 500 index, as shown in Figure 2.

*Figure 2.* S&P 500 index close price during the stock market crash of 2008 (Yahoo, 2013).

The underlying factors influencing the drop from mid-2007 through early 2009 likely differ from those factors influencing the subsequent bull market over the following years.

This dissertation examines applying genetic programming to predict non-linear time series, such as those shown in the examples above, and discusses how performance can be improved by incorporating software design patterns that enable regime specific behavior in the

evolutionary framework. The following section provides additional background and explanation of these and other technologies addressed in this document.

A genetic programming approach to handling changing regimes in time series forecasting was developed by Wagner, Michalewicz, Khouja, McGregor (2007). Recognizing that most existing forecasting methods assumed a static data generating environment that may not apply to nonlinear data such as stock series, Wagner et al. developed the Dynamic Forecasting Genetic Program (DyFor GP) model. DyFor GP uses an automated, online regime handling methodology. Two sliding analysis windows are used, instead of a more typically used single window. Two program populations are independently evolved based on the data in each window. Fitness is determined by calculating forecasting accuracy of future points (those points immediately after the sliding window). Regime changes are inferred by comparing the accuracy of the best individuals from the two program populations.

DyFor GP does not explicitly detect regimes; it only determines if a regime is stable or changing and fine tunes the analysis windows to favor the current regime. DyFor GP assumes that during stable regimes a larger analysis window will yield better results as it includes more data points within the same regime. If prediction accuracy decreases, DyFor GP assumes the regime is changing and the sliding windows are shrunk to include less data from the prior regime. If prediction accuracy increases, DyFor GP assumes a new regime has stabilized and the windows are increased to include more data points from the current regime. Following each comparison, the resultant windows are slid forward along the time series with additional iterations performed, using the same approach over the new data set.

DyFor GP has several limitations. DyFor GP is only applicable as an online method that has been trained on prior data. A generated DyFor solution could not be used at an arbitrary

future point, as regime detection is based only on performance differentials over time. This dissertation incorporates a regime discovery step into the process where programs are evolved to determine the current regime using all the data available at any point in time. This approach also allows pluggable regime detection algorithms. DyFor GP could even be used as a regime detection implementation, as could domain knowledge or other statistical methods.

**Problem Statement**

Most evolutionary methods for predicting time series do not consider regime change: changes to the underlying data generating process. Even when training occurs across different regimes, most existing algorithms do not explicitly consider regime change in testing or in actual prediction. One methodology that addresses this concern is DyFor GP, which attempts to automatically detect changing regimes and steer the algorithm towards training data primarily in the current regime (Wagner et al., 2007). Performance, however, can be improved by addressing several limitations of the DyFor GP methodology.

DyFor GP does not specifically address modularity; Entire programs are evolved as a single unit. Crossover between program code specific to different regimes can occur and may negatively affect the performance of the overall solution. Regime specific code is only used as "hints" for current period, or as its creators state "this knowledge allows for faster convergence to current conditions by giving the model searching process a 'head-start'" (Wagner et al., 2007, p. 433). In addition, code optimized for prior regimes is injected into the current program population whenever a new regime is discovered with the hope that the injected code will be applicable to the new regime, or evolved out if not. It may be more optimal to explicitly detect regimes and isolate regime specific code in a separate evolvable program branch. Besides the performance implications, the explanatory benefit of genetic programming, a key advantage of

evolutionary algorithms, is limited, as prior regime specific rules are lost as they are evolved out in favor of code more fit for the current period.

Modularity is listed as an open issue in genetic programming by O'Neill, Vanneschi, Gustafson, & Banzhaf (2010). They state that "Adopting practices from other computer science methods may be one source of tools" for achieving greater modularity (ibid., p. 12). Specifically addressing automatically defined functions (ADF), an early GP approach to evolving reusable functions (Koza, 1992), the paper further asks:

> Are ADFs necessary/sufficient as a formalism to help solve grand-challenge problems i.e. to provide scalability? And even more ambitiously: Can we achieve modularity, hierarchy and reuse in a more controlled and principled manner? Can Software Engineering provide insights and metrics on how to achieve this? (ibid., p. 13)

This dissertation argues that software design patterns can facilitate the "controlled and principled manner" needed to achieve greater modularity and formalism.

**Dissertation Goals**

The goal of this dissertation was to improve upon the performance of DyFor GP for time series prediction in the presence of regime change. This goal was achieved by incorporating a regime detection branch into the genetic programming paradigm. The results of the regime detection are used in the prediction process through two new modularity techniques: automatically defined templates (ADT), a variation on automatically defined functions (Koza, 1994), and automatically acquired templates (AAT), a variation on module acquisition (Angeline & Pollack, 1993). These new modularity techniques enable regime specific behavior in the canonical genetic programming algorithm. The dissertation goal was measured by comparing results achieved by the proposed approach to an implementation of DyFor GP created for this

study. Performance was measured by comparing the prediction accuracy and computational cost of each approach.

The dissertation goal was realized by the development of a genetic programming system incorporating and enhancing prior approaches to improved modularity (automatically defined function and module acquisition) and regime change (DyFor GP). The system, though flexible enough to address varied domains, focused primarily on financial prediction. Therefore, a sufficient set of financial functions to enable realistic predictions was incorporated into the final system.

**Research Questions**

This document presents a new approach for improving modularity in genetic programming using software design patterns. This new approach must be compared to earlier efforts at improving GP modularity. Automatically defined functions (Koza, 1994) and module acquisition (Angeline & Pollack, 1993) are considered canonical approaches to modularity as these were the first solutions addressing this problem and are widely referenced in academic literature. Several related questions were addressed:

1. How do ADT and AAT compare with DyFor GP in performance accuracy and computational cost? What is the benefit of these approaches over automatically defined functions, with and without the presence of regime change?
2. Can the necessary computational performance be achieved using Clojure as the target representational language? As a LISP dialect, Clojure can conveniently represent and evaluate program trees directly. Though not an explicit goal of the project, its implementation will determine the feasibility of this approach and whether an

alternate approach, such as another language like Scala, or alternate evaluation method, such as direct expression tree evaluation in Java, is necessary.

**Relevance and Significance**

Genetic programming is only beginning to achieve the fundamental goals of artificial intelligence, as stated by Arthur Samuel and quoted earlier in this paper. Koza et al. (2008) list 38 “human competitive” results produced through genetic programming. This number is only a small fraction of the achievements realized by other automated or non-automated areas of human research and discovery. In addition, programs generated by computers do not currently resemble the highly structured code produced by humans. Incorporating additional software engineering principles into the genetic programming model should help further this goal. There is no theoretical limit to the quality of programs that can be automatically generated by genetic programming. Producing code “which if done by humans would be assumed to involve the use of intelligence” would prove a significant advancement.

Though the research presented primarily addresses the domain of finance, this research more broadly applies to predicting any time series. Such data is prevalent in many scientific domain, such as medical monitoring and especially in more random series, such as climate modeling, where governing influencers may not be known beforehand.

**Barriers and Issues**

Predicting financial time series is difficult. That this problem has not been widely accepted as being solved is proof of this difficulty. Many researchers believe, as described by efficient market theory (Fama, 1965), that all available information and future expectations are factored into the current price of any financial instrument and any future price movement is caused only by unexpected events. If this theory is true, it would be impossible to make

predictions consistently superior to a buy and hold strategy. Some researchers, often subscribing to technical analysis, hold that markets are not entirely efficient and are moved by human, potentially irrational, factors and beliefs. This is evidently true as human decision making does not use a single algorithm for determining when to buy or sell. No matter which side is correct, there exists a multi-billion-dollar industry built around making correct investment decisions and beating the market. While the primary goal of the proposed research is not to develop new market beating strategies, adequate performance in such as task is necessary to justify the approach and results. In addition, the stochastic approach used in genetic programming may not always yield a successful outcome.

Genetic programming has its own set of complexities. A prime concern in GP is bloat, or the tendency of generated programs to grow larger and larger, after a period of stability (Poli, Vanneschi, Langdon, & McPhee, 2010). Bloat can complicate solutions and make them harder to understand or slow down processing, allowing fewer and less optimum solutions to be discovered (Jong, Watson, & Pollack, 2001). Often, limits on the size or depth of generated solutions are applied. Modularity techniques, such as ADF, can also help combat bloat (Bleuler, Brack, Thiele, & Zitzler, 2001).

Finally, though GP packages exist in many languages, extending GP in the manner necessary required custom development to facilitate techniques not available in existing systems. This undertaking was not trivial but was needed to achieve the desired goals. Alternative approaches to genetic programming were implemented for comparison to the new methodology presented. As no canned software exists or is generally available for these alternative approaches, they were built based solely on descriptions in the literature.

**Assumptions, Limitations, and Delimitations**

The primary focus of this dissertation was on improving genetic programming performance through enhanced modularity. While modularity in GP is an established research area, the volume of existing research does not compare to the abundance of studies, academic and industrial, on stock market prediction. Due to the potential financial reward, a large number of studies on market prediction have been done. Many of these studies are not public, but kept hidden behind the walls of hedge funds, as sharing this research would not necessarily be beneficial. Regarding the widely known approaches to market prediction, many of the more successful approaches often lose their performance edge at later times, pointing to regime dependent factors. Even if regimes are considered in the prediction process, efficient markets likely apply to some markets at some times, so a prediction approach consistently superior to a random walk model is not likely achievable.

Another limitation of this study was the choice to restrict the number of predictor series considered and focus development on the core algorithm, potentially at the expense of better predictions. However, this methodology is easily extended to include additional predictors. Another aspect of the presented approach that could affect prediction accuracy is the limitation to a predetermined number of regimes. This constraint may limit performance accuracy if the choice is incorrect. Typical of first implementations of new methodologies, future work can extend this approach and incorporate regime number determination. This was the case with ADF, which originally required declaration of a fixed number of ADF branches and was later extended to discover the optimal number (Koza, 1994).

A core requirement in genetic programming is sufficiency, which holds that the primitive operators available should be capable of solving the problem at hand (Koza, 1992). This

requirement demands a certain level of domain specific knowledge. In the case of non-deterministic prediction problems, such as financial time series forecasting, the set of operators sufficient for prediction is not agreed upon or even known. There exists a multitude of technical indicators and possible predictor series available. There is no guarantee that the set of primitives chosen by the analyst is indeed sufficient for the problem, though a good set of primitives can be discovered via trial and error over multiple runs. However, comparative performance of the proposed approach versus other GP approaches can still be measured.

There is no guarantee that either of proposed approaches will improve on existing methods—there is no guarantee in any scientific endeavor. However, in the domain of financial prediction, even a small improvement in performance will be extremely beneficial.

**Definitions of Terms**

*Design Pattern* – Common solutions to recurring problems. In software design, this term often refers to patterns cataloged by the Gang of Four (Gamma, Helm, Johnson, & Vlissides, 1995) but later expanded by other researchers.

*Efficient Market Hypothesis/ Efficient Market Theory (EMT/EMH)* – The theory that consistently predicting stock prices is not possible since all currently known information is already factored into the current price and future price movement is exclusively due to unforeseen events.

*Data Generating Process (DGP)* – The actual process producing the data observed in a time series. This process is often unknown and only inferable through data observations.

*Individual* – A single program in a population of evolving programs.

*Nonlinear Time Series* – A series that cannot be modeled as a simple linear equation of dependent variables. Structural breaks/regime change and chaotic series are examples on nonlinear behavior.

*Predictor Series* – A time series that may be used to help predict another, target time series (see Target Series below). An autoregressive approach implies the target series is itself a predictor.

*Regime Change* –A distinct shift in the appearance and behavior of a time series, perhaps due to a fundamental change in the DGP (see above).

*Stationary Time Series* – A time series with a mean and variance that does not changes over time. Informally, the series looks the same at any time period.

*Stock Return* – Percent change in price over a given period. Many studies use this metric instead of absolute price as it is normalized to take into account varying magnitudes. Log return is also frequently used in econometric studies, depending on the expected distribution of returns.

*Target Series* – The time series being predicted.

*Technical Analysis* - Financial series prediction approach based on historical pricing and volume metrics of the target series. This approach often involves the search for visual patterns thought to predict future trends.

*Template* – Related to the template method pattern (see Design Patterns above), an algorithm where certain steps are deferred to multiple, context specific implementations.

**Summary**

A shortcoming of statistical methods of time series prediction is their non-realistic assumptions, such as stationary and linearity. Soft methods of AI have been applied to prediction problems to overcome some of these limitations or when the underlying data generation process is unknown and cannot be modeled. However, most AI approaches do not address regime change. Such a situation was shown to be common in time series such as occur in finance.

This dissertation presents a methodology to better predict time series in the present of regime change using genetic programming. An existing GP methodology, DyFor GP, was

discussed and various ways to improve on this approach were presented. Most critically, DyFor does not incorporate modularity features. Such features were shown to be a common path of development in GP research and may hold promise for improved prediction performance. Two modularity approaches, automatically defined templates (ADT) and automatically acquired templates (AAT) are presented to address this problem.

# Chapter 2
# Review of the Literature

This paper presents automatically defined templates (ADT), a new approach to genetic programming modularity that may improve time series prediction, especially in the presence of regime change or other nonlinearities. Rather than an exhaustive review, this section highlights the most prevalent methods of time series prediction from the econometric and AI literature. Domain specific techniques that may be incorporated into the prediction are not. For example, numerous stock prediction approaches exist in the academic and trade financial literature such as put/call ratios, investment sentiment, etc. These techniques will only be discussed where applicable to the methodology or a specific experiment. The actual methods used are independent from the ADT methodology. The framework is able to choose and incorporate any method specified as part of the set of primitives, as can most genetic programming approaches.

## Time Series Prediction and the Stock Market

How to beat the stock market? This question motivates countless amateur and professional financial researchers and traders. To beat the market, or at least trade profitably, the trader must make appropriate buy and sell decisions. Such actions require a general idea of the direction of the market.  As this is a potentially lucrative problem to solve, numerous approaches have been tried. These include classical time series forecasting approaches, such as such as autoregressive methods, as well as “soft” methods of artificial intelligence, such as neural networks and evolutionary programming. There is a long history of attempts at stock market prediction. An exhaustive list of AI approaches are given in (Atsalakis & Valavanis, 2009). Those most relevant to the proposed research is briefly described in this section.

Hellstrom compared the time series prediction approach with the model-driven approach (Hellstrom & Holmstrom, 1999). In the former, epitomized by an autoregressive model, the value of a time series at a future point is predicted using a root mean square error calculation. The input is often a fixed-sized, prior slice of that time series. Investment decisions are taken based on the predicted future value. In the latter model-driven approach, called by Hellstrom the trading simulation approach, the prediction and investment decisions are separated. The input to the model is a set of data available at the current time. The input can be a variety of indicators—not only prior series values. The output of the model function provides a value that can be used for investment decisions. The advantage of the model-driven approach is greater flexibility. The model building task is described as "The task for the learning process in the trading simulation approach is to find the function g to maximize the profit, when applying the rule on real data (ibid., p. 4)". However, though individual model parameters can be optimized via search, this approach applied predefined models which were limited and static. The description of the model approach is consistent with the techniques proposed in this paper. The proposed research however takes a model discovery approach instead using of predefined trading rules.

Forecasting of time series is useful in many domains such as sales, marketing, and finance. Wagner et al. (2007) discuss classical methods of time series prediction, which include both linear and nonlinear regression methods, and non-deterministic AI-based approaches. They note that all these methods "assume that the underlying data generating process of the time series is constant" (ibid., p. 434). In addition, existing methods do not automatically handle regime change and require human judgment in how and where to apply forecasting techniques in such an environment (ibid.).

Often, attempts to reconcile this problem are handled by setting up training periods to include [what are known to be] different regimes. In a study on profitable trading pattern discovery using genetic algorithms, Canelas, Neves, & Horta (2012) make a stated choice to use data from 1998 to 2010 , specifically to include both a bull and bear market in the single testing period and "to include the instabilities and crisis of the year 2008 and beyond, to test our algorithm on the worst market conditions of the last years (ibid., p. 1060)". Even if training occurs across different time periods and regimes, any approach which yields a single, static result is limiting. It is likely that the US financial crisis of 2008 exhibit different underlying patterns than the bull market of the late 1990s and those may differ from patterns seen in earlier decades.

Another attempt at addressing this problem was made by Yu, Chen, & Kuo (2005). This study incorporated multiple, overlapping training/testing periods. Different GP runs were alternatively processed against different periods, as shown in Figure 3.

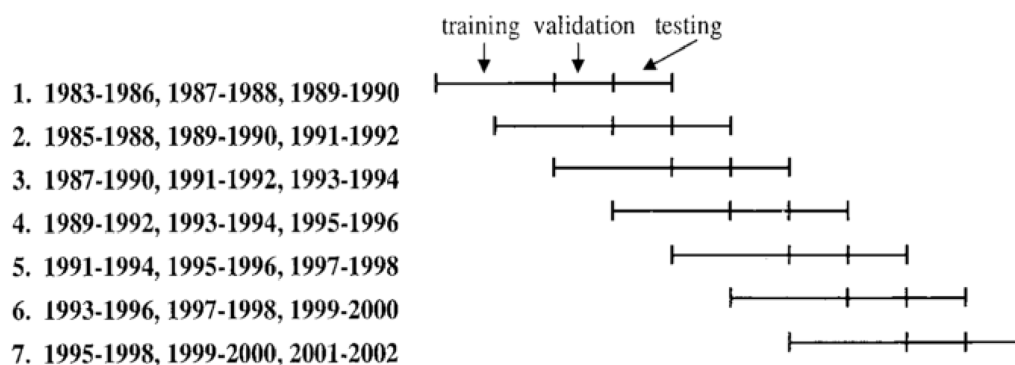



*Figure 3.* Overlapping training period approach used by Yu et al. (2005, p. 18).

This approach, while perhaps more valid that those that employ single training and testing periods, still has the disadvantage of discovering only a single set of prediction rules to be used in all future periods and may not be optimal across regimes.

**Econometric Methods for Time Series Prediction**

The following methods are the most common and established in the time series prediction literature. These are described with the intent of capturing these approaches in a GP model, either as explicit functions or constructively from lower level building blocks. These methods can also be extended for more general time series model building, where additional factors other than the time series under analysis are considered.

The most basic statistical prediction method is linear regression. Linear regression was also used as a basic example in (Koza, 1992).[1] This technique attempts to explain a dependent variable, such as the future value of a time series, via a series of observed independent variables. The relationship is constant over of time and can be modeled in Equation (1), representing a line with slope β and y intercept α.

$$y = \alpha + \beta x \tag{1}$$

For relationships with linear coefficients, a best fit line can be determined via the ordinary least squares (OLS) method. As the observed values will not generally fall on the best fit line for all but synthetic series, an error term is added to the model to yield the general Equation (2):

$$y_t = \alpha + \beta x_t + \mu_t \tag{2}$$

Since $\mu_t$ is by definition unknown, the error term does not factor into any forecast.

Linear regression requires certain assumptions. Ordinary least squares estimation assumes that the error terms follow a normal distribution with zero mean and constant variance

---

[1] Koza called this symbolic regression and generalized this to an arbitrary, non-linear curve.

with no correlation between error terms. Real world time series do not follow this assumption due to changing mean or variance (volatility), autocorrelation of error terms, or structural breaks/regime change. The following econometric models address these limitations in the attempt to provide more realistic and accurate predictions.

Equation (1) uses independent variables to predict a single dependent variable. However, there is often no independent variable or explanatory model available, so the approach uses lagged values of the dependent variable are used as predictors. This approach, called an autoregressive model, is described by Equation (3):

$$y_t = \mu + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \cdots + \phi_p y_{t-p} + u_t \tag{3}$$

Where: $u_t$ is a random shock occurring at period t
μ is the process mean
$\phi_1 \ldots \phi_p$ are unknown coefficients weighing prior lagged values
P is the order of the process

(Brooks, 2014, Section 6.3)

The above model is considered an AR(p) process, incorporating p lagged values. This model ignores shocks from earlier periods (i.e. $u_{i<t}$). Coefficients can be estimated using ordinary least squares on sample data, as all the independent variables are observed.

In the linear regression and autoregressive models, the random shock at time t is assumed to immediately decay to 0 at time t+1. However, a slower decaying noise can also be modeled, assuming that such a shock takes several periods to be fully incorporated by the dependent variable. Such a process is called a moving average process and can be modeled Equation (4):

$$y_t = \mu + u_t + \theta_1 u_{t-1} + \theta_2 u_{t-2} + \cdots + \theta_q u_{t-q} \tag{4}$$

(Brooks, 2014, Section 6.3)

Unknown coefficients $\theta_1$ through $\theta_q$ can be estimated using maximum likelihood or other numeric methods. Least squares cannot be used, as the error terms u are not directly observable.

A common variation on the moving average process is exponential moving average. An advantage of this approach is that only the prior period smoothed value is needed to predict the current value, as illustrated in Equation (5):

$$S = \alpha y_t + (1 - \alpha) S_{t-1} \tag{5}$$

(Brooks, 2014, Section 6.10)

$S_t$ is the smoothed value at time t. $\alpha$ ($0 <= \alpha <= 1$) is a smoothing constant weighing the current value to past values. In forecasting, $S_t$ is taken as the predicted value for any future period.

The above two methods are combined into an ARMA (autoregressive moving average) process, where both the influence of the random shocks and autoregressive effects are modeled. An ARMA (p, q) process is composed of an AR(p) process and an MA(q) process. The combination of Equations *(3)* and *(4)* yield Equation (6). Future forecasts are made by extrapolating the model forward in time.

$$y_t = \mu + \acute{\emptyset} y_{t-1} + \acute{\emptyset}_2 u_{t-2} + \cdots + \acute{\emptyset}_q u_{t-p} + \theta_1 u_{t-1} + \theta_2 u_{t-2} + \cdots + \theta_q u_{t-q} + u_t \tag{6}$$

(Brooks, 2014, Section 6.6)

Box and Jenkins (1976) popularized an iterative method for determining the parameters of the ARMA process, making this arguably the most popular approach to time series prediction (Zhang, 2003). The steps in this methodology are:

1. Identify the order of the process via graphical inspection.
2. Estimate the parameters (coefficients).
3. Test the model to determine errors.
4. Repeat until an appropriate model is determined.

ARMA assumes a stationary process. Often a non-stationary process can be made stationary through techniques such as differences. A differencing factor is added into the model description to yield an ARIMA (p, d, q) process, where d is the number of times the target variable is differenced.

The econometric models previously described in this section assume stationary and linear processes. It has been observed that many financial time series are nonlinear. This can be seen in looking at stock volatility, as evidenced by the VIX index (Chicago Board Options Exchange, 2014), a measure of the volatility of the S&P 500 index, shown in Figure 4. Changing volatility violates the constant variance condition of stationarity.

*Figure 4.* VIX index (Chicago Board Options Exchange, 2014).

Non-stationary series can often be made stationary by differencing methods, such as using change in stock price (return) instead of absolute stock price. However, other nonlinearities, such as structural breaks and changing volatility, can invalidate techniques such as ARIMA (Zhang, 2003).

The ARCH (Autoregressive conditional heteroskedasticity) model applies an autoregressive model to the series variance. Such an approach is widely used to financial series

modelling where the variance is seen to change over time in a correlated manner. The variance of the ARCH model is governed by Equation (7):

$$\sigma_t^2 = \alpha_0 + \alpha_1 u_{t-1}^2 \tag{7}$$

(Brooks, 2014, Section 6.6)

The variance is assumed to follow an autoregressive AR(1) trend. The model for the dependent variable is not restricted and may be linear, autoregressive, or other model. However, the variance of the error is no longer assumed constant. Maximum likelihood numeric estimation can be used to determine ARCH parameters from sample data.

Many observers have noticed that financial time series tend to make sudden shifts, such as abrupt changes in mean or variance. Two models that address this concern are the threshold autoregressive model and the Markov regime switching model. These methods are the primary econometric alternative to the regime determination GP approach proposed in this paper. Both approaches break the time series into two or more separate series, one for each regime, and apply traditional modeling techniques, such as ARMA, to each independently. These methods differ in how they determine regimes when the regime breaks are unknown. In cases when the breaks are known, a more traditional seasonality approach using dummy variables can be used. The widespread acceptance of these econometric methods also support the notion proposed in this paper of using separate functions to determine regime breaks, independent of the primary time series model.

The threshold autoregressive model uses a variation of a piecewise linear approach. Regimes are determined by examining a time series of a chosen, observable variable, such as a lagged value of the target series or another independent variable. Different models may then be applied to each series. In Equation (*8*), two separate AR(1) models are applied depending on the

state variable in relation to a threshold r. Parameters can be estimated by maximum likelihood method.

$$y_t = \begin{cases} \mu_1 + \phi_1 y_{t-1} + u_{1t} \;\; if \; s_{t-k} < r \\ \mu_2 + \phi_2 y_{t-1} + u_{2t} \;\; if \; s_{t-k} \geq r \end{cases} \quad (8)$$

(Brooks, 2014, Section 10.23)

By contrast, in Markov switching models, the regime boundaries are determined based on a probabilistic model of an unobservable variable. This model was proposed by Hamilton (1989) as an attempt to better estimate GNP (US gross national product) growth. Prior studies assumed GNP "followed a stationary linear process", often using standard first order differencing of the return. Hamilton attempted to model a nonlinear stationary process governed by "discrete regime shifts"

A simple example is an AR(1) model subject to a regime dependent movement of the y intercept, modeled in Equation (9).

$$y_t = c_{s_t} + \phi y_{t-1} + \varepsilon_t \quad (9)$$

(Hamilton, 2008)

$c_{s_t}$, the y-intercept in Equation (9), takes two different values depending on the unobservable regime governing variable s at time t. The result of the regime change is a vertical shift in the trend line. S is determined probabilistically from the following equation.

$$P_r(s_t = j | s_{t-1} = i, s_{t-2} = k, \dots, y_{t-1}, y_{t-2} \dots) = P_r(s_t = j | s_{t-1} = i) = P_{ij} \quad (10)$$

(Hamilton, 2008)

This model assumes two states. Each state is probabilistically determined by only the prior state. Equation (10) states that the probability of being in state j at time t is based only the state at time $t_{-1}$. p is the probability of being in state 1 and staying in state 1. q is the probability of

being in state q and staying in state q. The state transition table of the two state Markov process is:

$$
\begin{aligned}
&Prob[S_t = 1 \,|S_{t-1} = 1] = p, \\
&Prob[S_t = 0 \,|S_{t-1} = 1] = 1 - p, \\
&Prob[S_t = 0 \,|S_{t-1} = 0] = q, \\
&Prob[S_t = 1 \,|S_{t-1} = 0] = 1 - q
\end{aligned}
\tag{11}
$$

(Hamilton, 1989, p. 360)

The state transitions can be represented as a Markov state transition diagram, shown in the Figure 5.

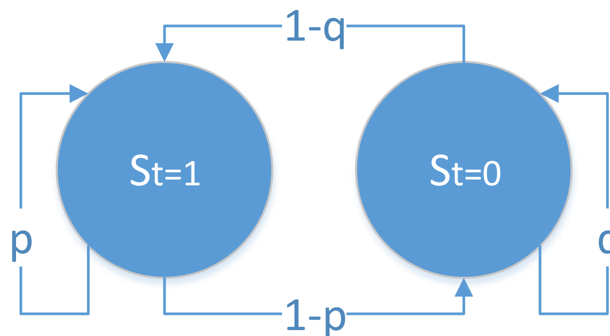



*Figure 5*. Markov state transition diagram.

Even though financial series are certainly nonlinear, nonlinear models do not always improve the out of sample performance over linear models (Dacco & Satchell, 1999). Regarding regime switching models, a small error in calculating the correct regime often results in a larger mean squared error compared to a random walk prediction (ibid.). Other researchers, (Allen & Karjalainen, 1999) for example, have noted that the additional complexity of nonlinear models encourage overfitting of the sample data at the expense of the out of sample forecasts.

**Artificial Intelligence Methods for Time Series Prediction**

The linear and nonlinear econometric methods described above attempt to specify a model describing the underlying data generating process seen in observed data. The first step of this approach is often to analyze the sampled data to determine the best choice of models and initial estimates of parameters. Often, a model is not evident or does not sufficiently explain the observed data. AI methods do not require this step and attempt to discover the correct model directly from sample data. Neural networking and evolutionary algorithms fall into this category.

**Neural networks.**

Whereas evolutionary algorithms model biological evolution, neural networks attempt to model the biological brain. Artificial neural networks (ANN) are composed of a network of neurons and connections between neurons. ANNs attempt to model an unknown nonlinear function. Given a set of inputs and, generally, one output, the system is trained and adjusted incrementally on observed data. Once the system is adequately trained, forecasts can be made by sending additional input into the system and observing the network's output.

Neural network are able to approximate any nonlinear function, given enough internal nodes (Zhang, Patuwo, & Hu, 1998). When lagged values of the series are used as input, the network is the equivalent of the following nonlinear autoregressive function:

$$y_t = f(y_{t-1}, y_{t-2}, \dots, y_{t-p}, \mathrm{w}) + \varepsilon_t \qquad (12)$$

where w is a vector of connection weights.

(Zhang, 2003)

An ANN is typically composed of three layers of nodes with each node connected to all the nodes in the next layer. The first layer, the input layer, accepts network input values. The second layer, the hidden layer, takes the sum of the weighted values of the input layer as input. A

nonlinear activation function is applied to the sum of the weighted inputs at each node, with that value propagated to the final, output layer. Each connection is modeled by a nonlinear activation function, typically a logistic function that maps any real input to an output y in the range [-1<y<1].

An autoregressive, time series prediction problem typically takes a sliding window of lagged values as input to the network. During multiple iterations, a window of series data is fed into the neural network and a back propagation algorithm is used to adjust network connection weights based on the observed error at the output layer. This process is repeated for all training data to arrive at a final network configuration. The ANN is then ready to predict of out of sample data. Boolean output can be modeled by a defined threshold in the network's output with any real value over that threshold taken as true.

A limitation of neural networks is that the structure is relatively static and generally limited to a small number of inputs, such as several lagged values of a target time series. By contrast, the structure of genetic programming is essentially unlimited, bounded only by what can be stated in its target language. GP also allows inclusion of a larger set of possible input values, as the structure is less fixed and more options can be explored due to its inherent parallel search across a large solution population. While ANNs contain a high level of parallelism, only one model is typically built. Evolutionary algorithms build numerous, competing models in parallel. ANNs typically only search for weights in a single, fixed network. The function modeled by the ANN is also a black box with unobservable and uninterpretable inner workings. While useful for forecasting, validating the model is challenging and little knowledge can be gained from observing its workings.

### Genetic algorithms.

Evolutionary algorithms produce software artifacts by applying principles of biology and Darwinian fitness. Evolutionary algorithms are often used for problems where no deterministic method is available and when a large solution space must be searched (Srinivas & Patnaik, 1994). In this methodology, a population of individual solutions is evolved using genetic operators inspired from biology, such as mating and mutation, narrowing the search space while producing increasingly better solutions. Branches of evolutionary algorithms include genetic algorithms (GA), which models solutions as chromosome structures (ibid.) and genetic programming (GP), which models solutions as computer programs (Koza, 1990). This section briefly discusses some applications of genetic algorithms to prediction problems. The following section addresses genetic programming in greater detail.

Most applications of genetic algorithms to stock prediction aim to optimize encoded model. Mahfoud & Mani (1996) encoded variables into a GA chromosome to "represent various factors, such as company earnings, that one might reasonably expect to have predictive value for the future performance of a stock" (ibid.). Their study used 15 attributes such as, price/earnings ratio and growth rate, as well as the stock's return over a given prior period. The GA was able to encode rules such as:

> IF [Earnings Surprise Expectation > 10% and Volatility > 7% and . . .]
>
> THEN Prediction = Up
>
> (Mahfoud & Mani, 1996, p. 567)

This approach is similar to that used in financial models. A limitation with this approach is that the choice of variables needs to be fixed to a single or limited number of points in time. While all

these variables change over time, most models are restricted to values at a fixed point in time, such as the stock price a week ago.

The limitation of genetic algorithms—and a reason why genetic programming may be more applicable to the problem of stock prediction—is seen in the work of Canelas et al. (2012). In this approach to stock prediction, most parameters were predefined and the GA simply found optimal values for these parameters. An example GA chromosome used in the study is shown in Figure 6.

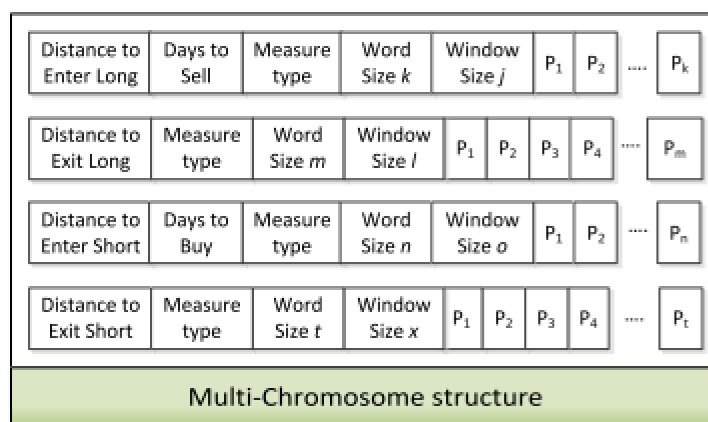



*Figure 6.* Sample genetic algorithm chromosome used for stock prediction by Canelas et al. (2012).

This example is the most complex chromosome in the referenced study and attempts to include a reasonably large number of discoverable parameters in the search. However, there are still many fixed assumptions made using the GA approach. Some of these assumptions are:

- A separate “exit long” pattern is assumed instead of “exiting long” when “enter long” is not true.
- The strategy always sells after a period of time.

- The only indicator used is the discovered pattern in the time series under analysis. This approach cannot take into account other commonly indicators such as new lows/high, moving average correlation, and filter rules.

The limitations just described arise from the linear nature of a GA chromosome. Many tasks will require a more complex problem representation—such as a computer program.

Genetic algorithms have also been used to address regime change. Davis, Lee, & Rodriguez-Yam (2006) used a GA to model nonlinear time series subject to structural breaks. The series is modeled by piecewise decomposition with each piece modeled as an autoregressive process. The model is fully defined by specifying the number and location of the breakpoints and the AR process order of each segment. These unknown parameters were determined using a GA. The GA optimizes a maximum likelihood function with penalties for parameter size (such as number of breaks) on the domain of all possible breaks over the series. Penalties are incorporated to limit overfitting of the training data. Li and Lund (2012) applied a similar approach to modeling climatic time series. Determining change points is necessary in climate modeling in order to normalize data to account for occurrences such as moving of weather stations or methodological changes. They observed that existing subsegmentation approaches, where a series is recursively broken down by adding an additional change point, were not optimal, especially where breakpoints occurred close in time. The study looked only at climate annual data. Expanding the domain to daily or even hour data might prove too granular for this approach.

**Genetic programming.**

Genetic programming is arguably the most flexible branch of evolutionary algorithms and can model any problem whose solution can be described in computer language. Genetic

programming is also preferred over other forms of evolutionary algorithms in problems where the parameters of the problem, called the size and shape, is not known beforehand (Koza et al., 2006). While other evolutionary algorithms use a more fixed representation, such as bit strings, the forms and solutions of genetic programming are essentially limitless.[2]

Programs evolved by genetic programming are often represented as tree structures. As trees are the native syntax of the LISP family of languages, many early implementations of GP used LISP programs as the target representation. To be applicable to GP, a method of automatically ranking potential solutions, by a fitness function, must be definable. Fitness functions can more easily be defined for prediction problems than for more abstract or subjective problems, such as web site development or system architecture design.

Koza (1992, pp. 507-513) examined prediction of the logistics equation, an example of a nonlinear chaotic time series. In such a chaotic series, described by Equation (13), the shape is highly dependent on the initial parameters, $x_0$ and $r$.

$$x_{t+1} = rx_t[1 - x_t] \tag{13}$$

Where $x_0$ $0 <= x_t <= 1$and $0 < r <= 4$ are fixed parameters

(Koza, 1992, p. 508)

Dynamical systems are modeled for GP prediction by the inclusion of a lagged series prior value (PV $= x_{t-1}$, the equivalent of an AR(1) process) as an independent variable in a symbolic regression model. When the initial condition is not known, Koza defined a two-part primitive,

---

[2] In practice, these programs are often functional or expression oriented and do not necessarily resemble human created software, though there is no theoretical reason why this needs to be the case.

syntactically constrained to appear only in the root of the GP expression tree. The first parameter of this new primitive contains the initial real value $x_0$ and the second component contains the series prediction expression. Accurate results were found for x values in the range 1-10.

Mulloy, Riolo, & Savit (1996) investigated prediction of more complex nonlinear time series than those examined by Koza, specifically the Mackey-Glass equation, which prior studies had not successfully modeled. The authors incorporate additional lagged values but limit the lagged input to ten prior values, noting that the size of the search space increase along with the size of the terminal set and that this is an "interesting and perhaps fundamental issue for both time series prediction research and GP dynamics research" (ibid., p. 169). The study used a modified crossover approach to avoid premature convergence to random walk predictions where the next predicted value equals the current observed value. Such behavior can occur in early generations of solution evolution, before more detailed models can be evolved, as the prior value is often a good prediction of the current value. A modified crossover operation, FONTX (forbid one-node tree crossover), was used which simply rejects and reselects nodes when they are both one node trees. This approach improved upon prior results in nonlinear GP prediction. While having a higher MSE in training data, the study reported a lower MSE in testing data. They also claim that an elitist approach, where the best individual of each generation is retained, might further improve performance.

Kaboudan (1998) looked at determining whether a particular series can be predicted at all or is purely random. This simple test was done by running a symbolic regression on the target series and also on a shuffled (randomized) version of that same series. If any regularities exist in the original data, the prediction error of the actual series should be less than that of the shuffled series. The predictability metric used is shown in Equation (14).

$$E = \frac{SSE_s - SSEy}{SSE_s} \tag{14}$$

Where $SSE_s$ is the sum of the squared errors of the shuffled series

$SSE_y$ is the is the sum of the squared errors of the actual series

(Kaboudan, 1998)

A predictability value of E≈0 indicates random data while a value ≈1 indicates fully deterministic data. Several commonly analyzed nonlinear series, with and without added noise, were analyzed and it was shown that the predictability was proportional to the signal-noise ratio.

The approach just described was applied to prediction of individual stock returns and stock prices (Kaboudan, 2000). The following three step process was outlined.

1- Use the predictability metric to determine if a series is predictable
2- Decide which prediction factors to incorporate.
3- Run GP to determine the underlying data generating process

To determine predictability, Kaboudan looked at only lagged values of the series to determine whether a pattern could be uncovered in the data. For actually prediction, Kaboudan used 35 indicators that included lagged pricing metrics and changes measured in the Dow Jones Industrial Average. The study compared pricing series with returned series and found only prices were deemed predictable as per step one above. This observation is likely due to the known autocorrelation of absolute pricing levels. One day ahead prediction of pricing was done using the evolved GP method and compared to a random walk. The GP method showed greater accuracy than the random walk method. Accurate prediction of values further in the future was not successful in this study.

The third step in the process described above has become a common approach to GP stock prediction, where standard operators, technical indicators, and lagged series values are established as the primitive set and an investment decision is made base on an evolved expression approximating the underlying data generating process and its impact on investment decisions. Individual prediction studies will vary due to researcher's judgment in determining GP parameters and experiment specifics.

Another pioneering study of GP and stock market prediction was made by Allen & Karjalainen (1999). This approach can be taken as the canonical approach to GP based stock prediction. This study used GP to investigate the value of technical trading rules, asserting that prior econometric studies have been biased since the researchers choose the specific technical rules to incorporate. This study attempted to use GP to uncover the rules and therefore avoid bias since these would be discovered by machine learning. The study looked at the S&P 500 index daily close from 1925-1995. An expression was evolved to determine whether to be fully invested or fully out of the market. Fitness was evaluated by investment return, with out of the market days accumulating a risk free return rate. Available functions included moving average and extremum values (ma(n), max(n), min(n), over the prior n values), arithmetic functions (+, -, /, *), logical (if-then-else, and, not, or, >, <) as well as Boolean and real valued (randomized between 0 and 2) constants. Also included were current price (price()) and lagged values (lag(n)) functions from the target series. Fitness was calculated as the excess return over buy and hold. Transaction costs were factored into the fitness calculation, greatly affecting the performance and behavior of the reported results. Higher costs led to lower returns and more importantly, fewer trades per year. This is an important consideration as it shows transaction costs cannot be ignored during evolution and then factored in at a later time to calculate actual returns.

Allen & Karjalainen determined that the GP could not beat buy and hold when transaction costs are considered. The study also, however, showed that the average return during periods in the market periods is higher than the return during periods out of the market, indicating a strong predictive value to the algorithm (ibid.). It should also be noted that transaction costs have been continually falling. Figure 7 shows the drop in transaction costs from the mid-1970s through the study period and can likely be extrapolated further. In addition, the 2000s saw the rise of high frequency trading which further served to remove additional arbitrage possibilities in the spread between bid and ask prices. Also, decimalization of US stock markets in 2001 (Securities and Exchange Commission, 2001) lowered the minimum spread from 1/16 dollar to a penny, further reducing trading costs. Reaching its ultimate conclusion, many services such as Robinhood (Robinhood Financial, 2016) currently offer commission free trading with no apparent restriction on trade frequency. Such trends show that transaction costs may factor less into realistic performance models. Backtesting studies, however, may wish to replicate financial conditions at the historical period, including transaction cost and risk free returns.

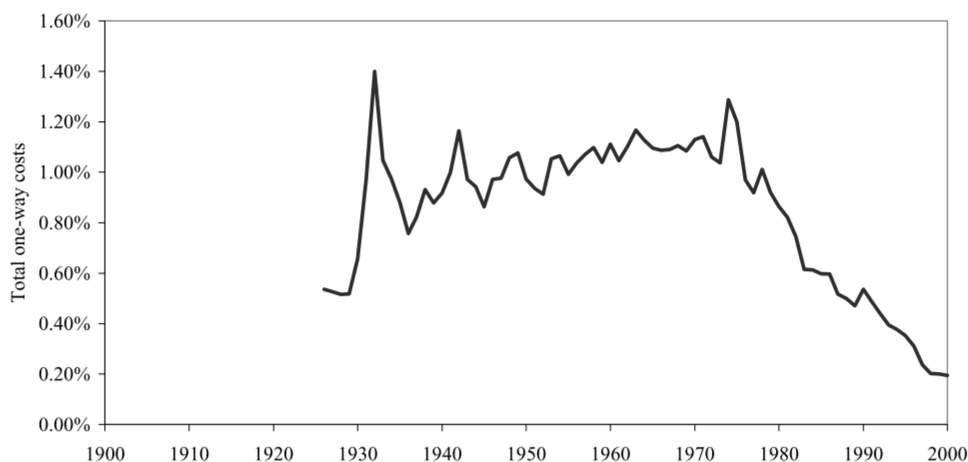



*Figure 7.* Average historical NYSE transaction costs, calculated as half-spread plus NYSE commission. Taken from (Jones, 2002, p. 43).

A study claiming more success in forecasting future returns was presented in (Tsang & Butler, 1998). The authors discuss the EDDIE 7 (Evolutionary Dynamic Data Investment Evaluator) tool that assists in financial decision making. Users of EDDIE select a set of predictor variables and choose a fitness measure of the form “will the target series gain X% in M days”, where X and M are user supplied values. Standard GP methods are used to evolve Boolean answers, which the user can interactively accept or further refine. EDDIE enforced a strict grammar on the GP tree, constrained by the BNF syntax in Figure 8. In testing, a result of true is taken as to indicate a fully invested position while a negative result indicates an out of the market position.

```
<Tree> ::= If-then-else <Condition> <Tree> <Tree> | Decision
<Condition> ::= <Condition> “And” <Condition> |
                <Condition> “Or” <Condition> |
                “Not” <Condition> |
                Variable <RelationOperation> Threshold
<Variable> ::= MA_12 | MA_50 | TBR_12 | TBR_50 | FLR_12 |
               FLR_50 | Vol_12 | Vol_50 | Mom_12 | Mom_50 |
               MomMA_12 | MomMA_50
<RelationOperation> ::= “>” | “<” | “=”
Decision is an integer, Positive or Negative implemented
Threshold is a real number
```

*Figure 8.* The Backus-Naur grammar used in EDDIE 7 (Kampouridis & Tsang, 2012, p. 531). The root if-then-else condition results in a buy/don’t buy answer. The variables are an arbitrary selection of technical analysis metrics.

Prediction of the S&P 500 index was done using the fitness goal “the index will rise by 4% within 63 trading days (3 months)”. Results were compared to a randomized approach and achieved an accuracy of 53.59% compared to 49.47% for the randomized approach. These results, however, may not always beat a buy and hold strategy, which is highly dependent on market trends over the analysis period.

The approach taken in EDDIE differs slightly from studies where the fitness criteria and predictor variables are fixed, as it is designed as an end user research tool. This design choice also avoids issues such as the combinatorial explosion of possible predictors, pinning that

decision on the user. A more recent version of the tool (EDDIE 8) (Kampouridis & Tsang, 2010) incorporated a broader search by allowing parameters (input to the technical analysis indicators variables in Figure 8, which were fixed at 12 and 50) to also be randomly generated. Analysis using actual and synthetic data determined that the wider search space of EDDIE 8 can produce better individual solutions though not improve the overall average (Kampouridis & Tsang, 2012). Specifically, the more constrained EDDIE 7 can perform better where the optimal answers where contained in the smaller, original search space, indicating the tradeoff between search convergence and search space. However, as the optimal solutions are invariably unknown beforehand, the larger search space of EDDIE 8 appeared to be the preferable approach.

Chen & Yeh (1997) investigated the operations of efficient market theory (EMT) using genetic programming. Instead of the probabilistic theories used in econometrics, could search intensity—the difficulty to find any underlying patterns—be a reason why EMT holds? The study first looked at several synthetic nonlinear equations, shown in Figure 9, and concluded that search intensity is a valid measure to gauge the difficulty, or impossibility, of solving certain equations.

$$X_{t+1} = 4x_t(1 - x_t), \;\; x_t \in [0,1] \; \forall t$$

$$X_{t+1} = 4x_t^3 - 3x_t, \;\; x_t \in [-1,1] \; \forall t$$

$$X_{t+1} = 8x_t^4 - 8x_t^2 + 1, \;\; x_t \in [-1,1] \; \forall t$$

*Figure 9*. Nonlinear equations analyzed by Chen & Ye (1997, p. 1047)

As part of this investigation, a prediction study of the rate of return of the S&P 500 index was done. That study concludes that GP can beat random walk prediction, but the computational costs expended in finding appropriate models is too great and therefore cannot disprove EMT. The methodology used was based on prior research which showed that though regularities in

price do not exist, or are averaged out, over the long term, they may exist for short periods. The study chose a smaller analysis window that might exhibit such nonlinear dependencies. They (arbitrarily) chose to look at windows of 50 values and uses a minimum description length criteria to choose the most complex 50 value window across the entire time series, using that as training data. Testing was done by predicting the rate of return for the five following values and measuring mean absolute percentage error (MAPE). The study also compares AR(p=1..10) processes and show that none of these beat random walk while 100% of the GP tests beat a random walk in the in-sample data. Out of sample prediction was also compared to random walk and shows approximately 50% beating a random walk. However, there was a negative correlation between performance of in sample and out of sample predictions, implying overfitting of the model, though many models were able to find regularities in both training and testing data.

The existence of short term regularities and the potential for prediction over shorter time periods is consistent with regime based behavior and the methodology presented in this dissertation. Also, computational processing and availability of data is continually improving so evaluation of any past must continually be revisited.

In examining earlier results in market prediction and genetic programming, Chen, Kuo, & Hoi (2008), noted "the diversity of the results: They are profitable in some markets some of the time, while they fail in other markets at other times, and so they are very inconclusive. (ibid., p. 99)". This may be due to the wide choice of parameters, data setup, and other implementation decisions in the various studies. They further write:

> The real issue is that research in this area is so limited that we are far from concluding anything firm. In particular, GP is notorious for its large number of user- supplied

parameters, and the current research is not rich enough to allow us to inquire whether these parameters may impact the performance of GP. Obviously, in order to better understand the present and the future of GP in evolving trading rules, more research needs to be done. (ibid., p. 100)

**DyFor GP.**

While recent GP prediction research has seen expansion into additional application domains and investigation of optimal parameters, modularity, and alternate GP representations, little has been done in addressing the issue of regime change—an area addressed by several econometric methods. One GP approach that does address this concern is the DyFor GP methodology (Wagner et al., 2007). As this primary goal of the proposed research is enhancing this methodology, DyFor GP is now examined in greater detail.

The DyFor GP process is illustrated in Figure 10 through Figure *14*. Figure 10 shows a time series with two underlying data generation processes applicable at segment1 and segment2. Two initial sliding windows, labeled win1 and win2, are used and a prediction made for a period after the end of the two windows, labeled pred. As both windows are within segment1 and therefore within a stable regime, it is likely that win2 will give a better result as it includes more data points from the current regime.

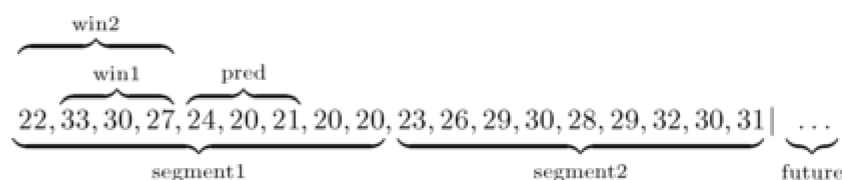



*Figure 10.* DyFor GP sliding window example, 1 of 5.

Based on the prior comparison, the size of both windows are increased as shown in Figure 11. As win2 still encompassed more of the current regime, it will likely still have better

predictive results than win1, so both windows are again expanded to include more of the current regime.

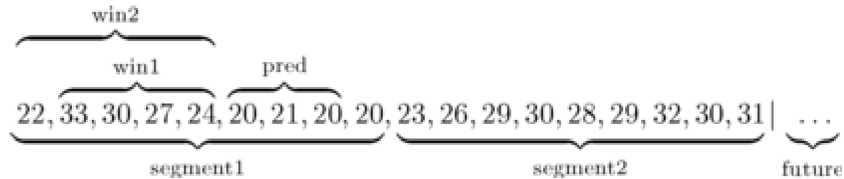



*Figure 11*. DyFor GP sliding window example, 2 of 5.

In Figure 12, the regime begins to change. Win1 will likely have a better prediction, as it includes less of the prior regime and an equal portion of the current regime.

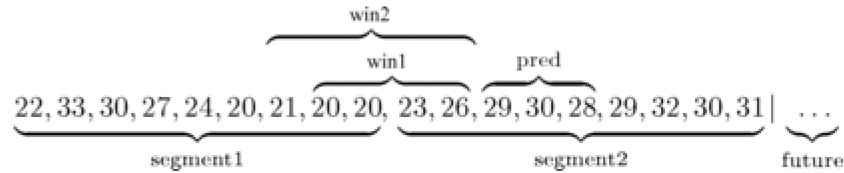



*Figure 12.* DyFor GP sliding window example, 3 of 5.

Therefore, both windows are decreased in Figure 13.

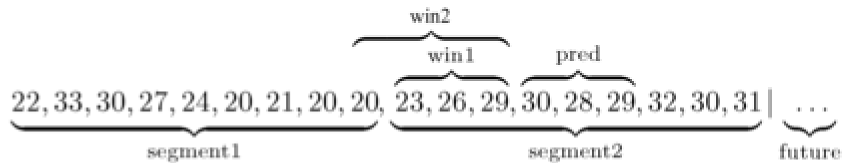



*Figure 13.* DyFor GP sliding window example, 4 of 5.

In Figure 14, when the regime is once again stable, win2 should yield better predictions than win1, so both windows are expanded to encompass more of the current regime.

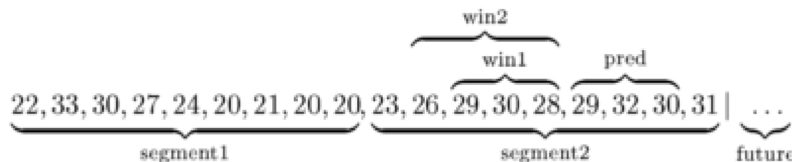



*Figure 14.* DyFor GP sliding window example, 5 of 5.

By analyzing the pattern of window resizing, regime boundaries can also be estimated.

DyFor GP can also learn from experience by saving program snapshots for future use. This approach may provide a performance benefit if future regimes are similar to those already encountered during earlier iterations. When a stable regime is indicated (by a certain number of consecutive window expansions) several of the fittest programs are selected and stored. As long as the regime remains stable, these programs are overwritten with newer, presumably fitter solutions. When a regime change is detected, the selected solutions are saved for future use. When the next regime change is indicated, these stored solutions are incorporated into the current program population in hope that these may be applicable to the new regime if it resembles the period where these solutions were first generated. Stored solutions will be injected into the current population until the regime is stable, at which point new solutions are once again saved. As there is no fixed limit on potential regimes, solution injection may include code both applicable and non-applicable to the current regime. However, program evolution should ideally favor solutions that are more applicable to the current regime.

***Bloat control in DyFor GP.***

Bloat is a long known problem in genetic programing where successive crossovers can result in exponential program growth without increasing program fitness (Jong et al., 2001).This problem is generally remedied by incorporating a fixed program depth or program size limit (ibid.). Other approaches exist such as penalizing larger programs with decreased fitness. Wagner and Michalewicz (2001) proposed alternative measure to control bloat and achieve quicker solution convergence. These approaches were also incorporated into the DyFor GP methodology.

Instead of a fixed depth limit or maximum program size, DyFor GP incorporates a global population size limit. Each evolving program can grow as needed with the restriction that total

population node count does not exceed a set limit. Simply replacing population node limits for program size limits will not control bloat and still may evolve inordinately large programs which will impact system performance.[3] Instead, a new method of selection is used (Wagner et al., 2007). After each dynamic generation[4], the next generation population is created from two groups:

1- Randomly generated programs

2- Subtrees of the fitter programs from the last generation

When dormant solutions are used and a regime change is in process (as described in the section above), programs are also selected from

3- Subtrees of saved solutions from a prior environment

This approach differs from canonical GP and the approach presented in this dissertation, where the next generation is constructed exclusively from the full fitter programs from the prior generation.

Wagner and Michalewicz do not explicitly state how the fitter programs are determined or how the subtrees are constructed. However, it can be inferred that after a series of training generations within a single dynamic generation, a subset of the evolved programs is selected via a tournament and a random subtree from each winning program is promoted to the next generation. A percentage of next generation programs will also be generated from scratch. The successive introduction of new programs necessitates additional training during each dynamic

---

[3] This was confirmed experimentally in this dissertation while testing various size limit approaches.

[4] A dynamic generation in DyFor GP includes one or more training generations and a prediction. After dynamic generation, the window is slid forward in time.

generation, similar to the initial training phase that occurs during a typical new genetic programming run. Wagner and Michalewicz (2001) show that this approach, independent from the regime handling features of DyFor GP, achieves comparable results to other bloat control approaches but uses less computer resources.[5]

**Modularity in genetic programming.**

An ongoing trend in genetic programming has been towards greater modularity. Besides mimicking human programmer practices, modularity can optimize performance via code reuse and increase program understandability (O'Neill et al., 2010). Modularity does not add any new capabilities to GP, as most languages can express the same algorithm with or without functions or similar constructs. On the surface, it simply allows a program to express the same functionally in a smaller size (Woodward, 2003), though it can be shown to improve the performance of certain classes of problems through decomposition and evolutionary development of individual components (Koza, 1992).

GP research shows that enhanced modularity can produce programs more similar to what a human programmer might produce, rather than indecipherable though syntactically and functionally correct code. Banzhaf, Nordin, Keller, & Francone (1998, p. 84) provide an informal comparison of steps in manual programming, such as cut-paste-modify of existing code, to genetic methods, such as crossover and mutation. This comparison further shows the parallel between manual and GP approaches to program creation and the potential to further model genetic programming techniques on established software engineering principles and practices.

---

[5] Wagner and Michalewicz (2001) did not examine the regime change features of DyFor GP.

Past research has seen genetic programming move in this direction by incorporating additional stylistic elements over purely syntactic concerns.

The first attempt at incorporating modularity into genetic programming was through automatically defined functions (ADF) (Koza, 1992). ADF declares a fixed number of functions. These functions are developed independently of, and can be used by, the evolving main program branch.[6] ADFs are specific to a program instance and are not shared by the larger population of evolving programs. Developed concurrently with ADF, module acquisition (Angeline & Pollack, 1992) enables the evolution of code shared by all programs in the GP population. Instead of a predefined number of functions, the module acquisition approach randomly extracts subtrees from the main program tree in a single program and replaces this code with a call to the new function. The function may be propagated to other programs in the population through genetic crossover. High fitness functions that spread throughout the population become the equivalent of shared code modules.

In examining modularity in GP-based approaches to stock market prediction, Yu et al. (2005) showed that black box functions tend to evolve into true/false constants, eliminating the benefit of modularity as equivalent code could be written with simple Boolean terminals. Automatically defined macros (Spector, 1995) attempt to evolve control structures, not just black box functions, and address the halting problem of recursive programs by lazy argument evaluation. Yu & Clark (1998) achieved modularity through the use of lambda abstractions,

---

[6] (Koza, 1994) discusses dynamically determining ADF parameters in a more detailed treatment of the subject.

anonymous functions used as parameters to other functions. By expanding the primitive set to include higher order functions that accept other functions as parameters, incorporating operations such as map and fold which apply functions to a list of values, they were able to improve performance in the Even-N-Parity problem (Koza, 1992) compared to the canonical GP approach with ADF. Whenever a higher order function is chosen as a program element, a lambda abstraction (anonymous function) is generated. These elements are pinned to the higher order function as a strongly typed parameter and can mate with other lambda abstractions of the same type. This approach also allows a limited form of recursion without the risk of an infinite loop. While not specifically addressing the type of regime behavior discussed in this proposal, incorporating higher order function could prove beneficial as an expansion of the GP primitive set if complexity does not increase disproportionally.

Though these and other approaches for improved modularity have been proposed, ADF still remain the most prevalent in the academic literature, perhaps because of its simplicity. ADF will be used in comparison to the modularity approaches presented in this dissertation. Module acquisition is also addressed as it was an early approach to modularity and holds many features and goals in common with this dissertation research.

**Patterns in Software Engineering**

Design patterns are recurring solutions to common problems. Software design patterns are often used as the basis for individual components within a larger application (Schmidt, Fayad, & Johnson, 1996). Design patterns differ from software frameworks, which are the basis for entire applications (Johnson, 1997). Design patterns can simplify code, improve understandability, and accelerate development. Such a form of modularity may also improve the

performance of programs generated by genetic programming as measured by structural complexity and computational effort.

Design patterns are perhaps the next stage in genetic programming development. Earlier GP research focused on elements of modularity such as automatically defined functions and recursion, more associated with the target language itself. Design patterns are software engineering techniques that can be applied to a broad range of problems. Both approaches serve to improve the quality of applications in terms of modularity, readability, and performance, therefore increasing the potential knowledge gained from the transparent nature of genetic programming solutions.

Gamma, Helm, Johnson, & Vlissides (1995) list 23 software design patterns to "record experience in designing object-oriented software as design patterns. Each design pattern systematically names, explains, and evaluates an important and recurring design in object-oriented systems" (ibid., p. 2). They add "Design patterns make it easier to reuse successful designs and architectures" (ibid.). For each pattern, Gamma et al. describe the pattern's intent, motivation, applicability, structure (using class diagrams), implementation, and other details. These patterns are often referred to as Gang of Four (GOF) patterns.

The template method pattern is one of the original Gang of Four patterns. The template method pattern defines "the skeleton of an algorithm in an operation, deferring some steps to subclasses. Template Method lets subclasses redefine certain steps of an algorithm without changing the algorithm's structure" (ibid., p. 325). This pattern allows an overall algorithm to be defined while allowing multiple implementations of individual steps. This pattern is typically implemented in object-oriented programming using an abstract class that cannot be directly instantiated. Only concrete subclasses of the abstract class may be instantiated. The concrete

subclasses must implement any methods in the overall algorithm that are not implemented in the abstract class. This approach allows for a great deal of code reuse and targeted behavior by instantiation of appropriate subclasses.

Figure 15 shows a class diagram illustrating the template method pattern. In this figure, an abstract class defines a concrete template method that references two local methods, primitiveOperation1 and primitiveOperation2. These two methods are declared abstract, meaning that the abstract class does not contain implementations for these methods but instead will defer to a concrete (non-abstract) subclass. These concrete classes must implement the abstract methods declared by the parent. Additionally, an interface specifies the contract that must be followed by the abstract and concrete classes. The template method defines the overall logic of the process, utilizing abstract and concrete methods contained or declared within the abstract class.

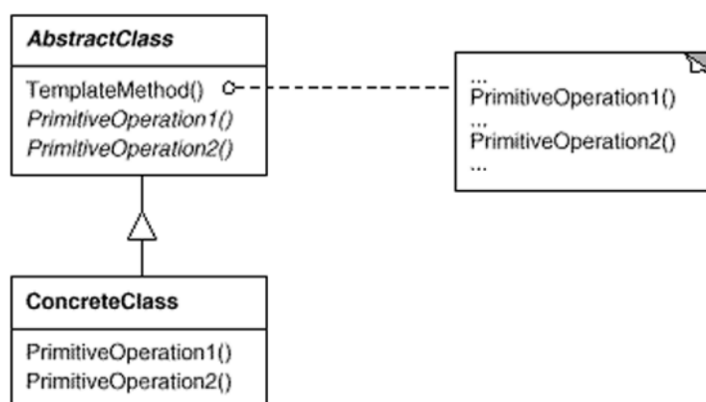



*Figure 15.* Template method pattern class diagram (Gamma et al., p. 327).

Another GOF pattern that allows for multiple implementations of an overall algorithm is the Strategy pattern. This pattern defines “a family of algorithms, encapsulate each one, and make[s] them interchangeable (ibid., p. 315)”. The pattern “lets the algorithm vary independent from the clients that use it (ibid.)”.

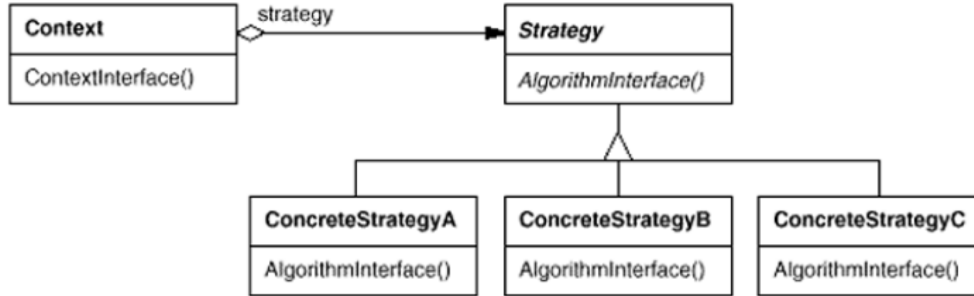



*Figure 16.* Strategy pattern class diagram (Gamma et al., p. 316).

In Figure 16, Context contains logic necessary to implement a task. Three possible strategies exist for a portion of this task. Instead of incorporating all logic into Context and deciding which approach to execute using conditionals, the pattern defines a Strategy interface along with concrete implementations of this interface. The interface is injected into the context at run time (at Context creation or later). The strategy pattern is often applied at program run time while the abstract template pattern is typically defined at compile time. However, template method pattern classes can be instantiated dynamically if needed.

The template and strategy patterns are described in greater detail in Appendix A. The template method pattern is used exclusively as a basis for the methods presented in this dissertation. Though closely related, the strategy pattern was not incorporated and is left for future consideration.

# Chapter 3
# Methodology

## Overview

This dissertation expands upon prior research in applying genetic programming to time series prediction. Two new modularity techniques were developed: automatically defined templates (ADT) and automatically acquired templates (AAT). These methods are specifically suited to handle dynamic changes in the underlying data generation process.

In the context of this research, templates (also called template methods or abstract template methods) refer to the abstract operations specified in the template method pattern as described in (Gamma et al., 1995, p. 328). In this software pattern, these abstract operations must be implemented by concrete implementations. In the context of this research, the concrete implementations are regime specific. This new approach is compared to DyFor GP, an existing GP methodology that also addresses regime change in time series prediction, and is also compared to other standard GP approaches.

A genetic programming system was developed that incorporates the two new modularity approaches. The DyFor GP methodology was also implemented to enable direct comparison of performance gains and uncover limitations of either approach.[7] The design and implementation of the system developed are presented in the following sections. The two new modularity methods are briefly described. Regime handling is also discussed as are other common GP related concerns.

---

[7] The bloat control features of DyFor GP were not considered, as these features concern program efficiency and not necessarily regime change

Most genetic programming implementations follow the canonical flow developed by Koza (1992) and reproduced in Figure 17.

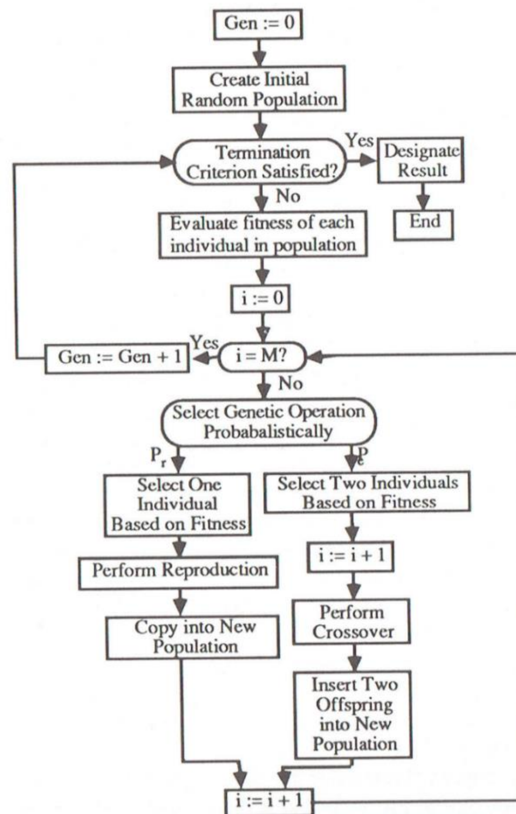



*Figure 17.* Canonical genetic programming flow. The lower two branches represent probabilistic selection of genetic operators involving two parents, such as crossover, or one parent, such as mutation. Taken from (Koza, 1992, p. 76).

The flow when automatically defined functions are included into the process is essentially the same. Several function definition branches are defined and incorporated into the single evolving program which is treated as a single unit, as shown in Figure 18. Koza (1994) specified a single results producing branch and one or more function definition branches. The only change when incorporating ADFs is that two-parent genetic operations can only occur between nodes in the same branch.

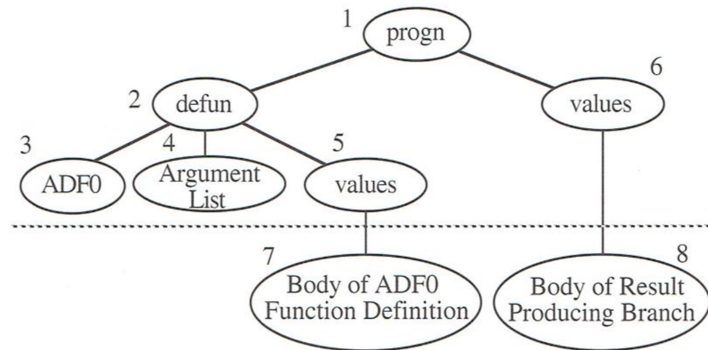



*Figure 18.* Program tree including an ADF branch.
Taken from (John R. Koza, 1994, p. 74).

The methodology developed for this dissertation follows the same paradigm as ADF, with the addition of several new program branch types. Besides the results producing branch, this approach also contains the following elements:

- One regime determination branch, modeled as a separate program.
- One or more template definition branches. These branches implement regime specific modularity as defined by the appropriate method: ADT or AAT. In general, there will be one implementation of each template definition branch per regime.

Instead of a single program, each individual in the population references two evolving programs—a result producing program (RPP) and a regime indicator group (RIG)—as shown in Figure 19. As in the canonical approach, each program is treated as a single evolving unit. Two parent crossover is only allowed between branches of the same. Selection and fitness operations are modified and are described below.

A population of regime indicator groups evolves in parallel with the main program branch. A regime indicator group will contain $2^n$ Boolean indicators for n possible regimes. The number of potential regimes, n, is predetermined by the analyst. Domain knowledge can

reasonably determine this number. For example, stock markets regimes are typically classified as bull/bear/sideways markets. Regimes can also be applied to other situations, such as low or high inflation periods.

Evolutionary regime discovery is optional. Predefined, rather than evolved, regime determination logic may be used, if domain knowledge, or other algorithms, such as pattern recognition or classification, makes this possible.

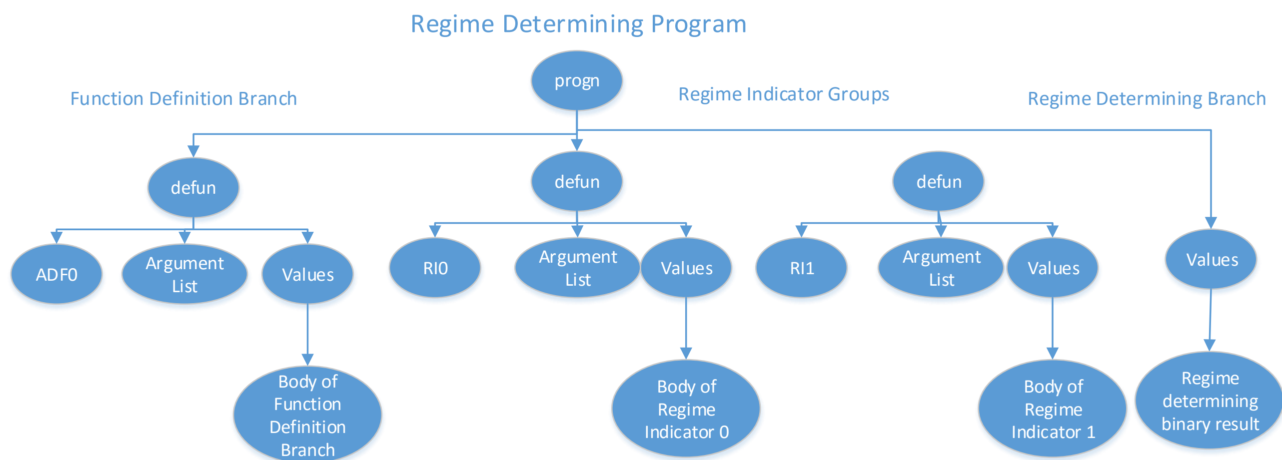



*Figure 19*. Regime determining program branch tree structure.

The result producing program in ADT is similar to the ADF approach shown in Figure 18, but replaces the function definition branch with a template definition branch. A sample program of this type is shown in Figure 20.

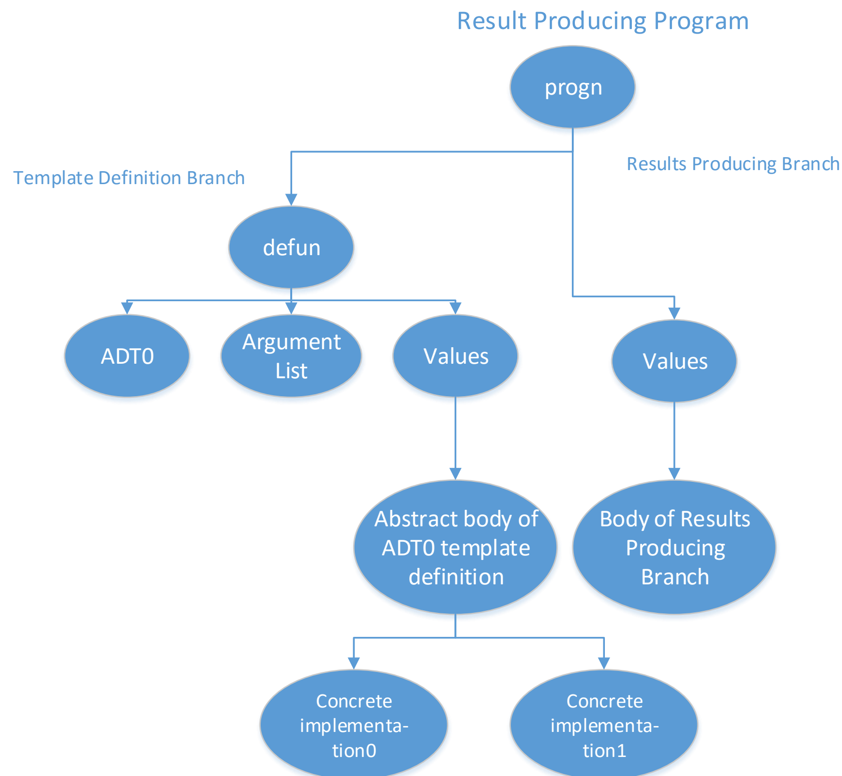



*Figure 20*. ADT tree structure. Both branches are represented as a single program containing two possible regimes. Labels artificially conform to common LISP syntax used by Koza (1992, 1994).

Fitness of RPP individuals can only be calculated independently if no regime dependency logic is needed, as in canonical GP or ADF. In ADT, a regime program is required to fully calculate the fitness of a RPP program. Fitness calculation is not possible for RIG programs, as there is generally no fitness function to determine its performance at regime detection other than in synthetically created cases. These two programs must be joined prior to fitness calculation.

The modified ADT algorithm is shown in Figure 21. The major steps are the following:

1. The initial population of regime indicator groups (RIG) and results producing program (RPP) is randomly initialized using available primitives and modularity approaches.
2. Calculate initial population fitness using an appropriate selection approach. Details of this fitness calculation method is discussed below.

3. The termination criteria are checked after each iteration.
4. Calculate population fitness at the start of each generation using an appropriate selection method. Details of this fitness calculation method is discussed below.
5. Probabilistically determine which genetic operation will be applied to the RIG and RPP. The selected operation may be different for each program group. Candidate individuals from RIG and RPP are chosen by traditional N-way tournament selection.
6. For crossover, two parents are needed to produce two offspring. For all other selected genetic operations, one parent is needed to produce one offspring. Determine if another offspring is needed for either RIG or RPP. If yes, execute tournament selection again.
7. Execute the required genetic operation and insert the new offspring into the current program population.

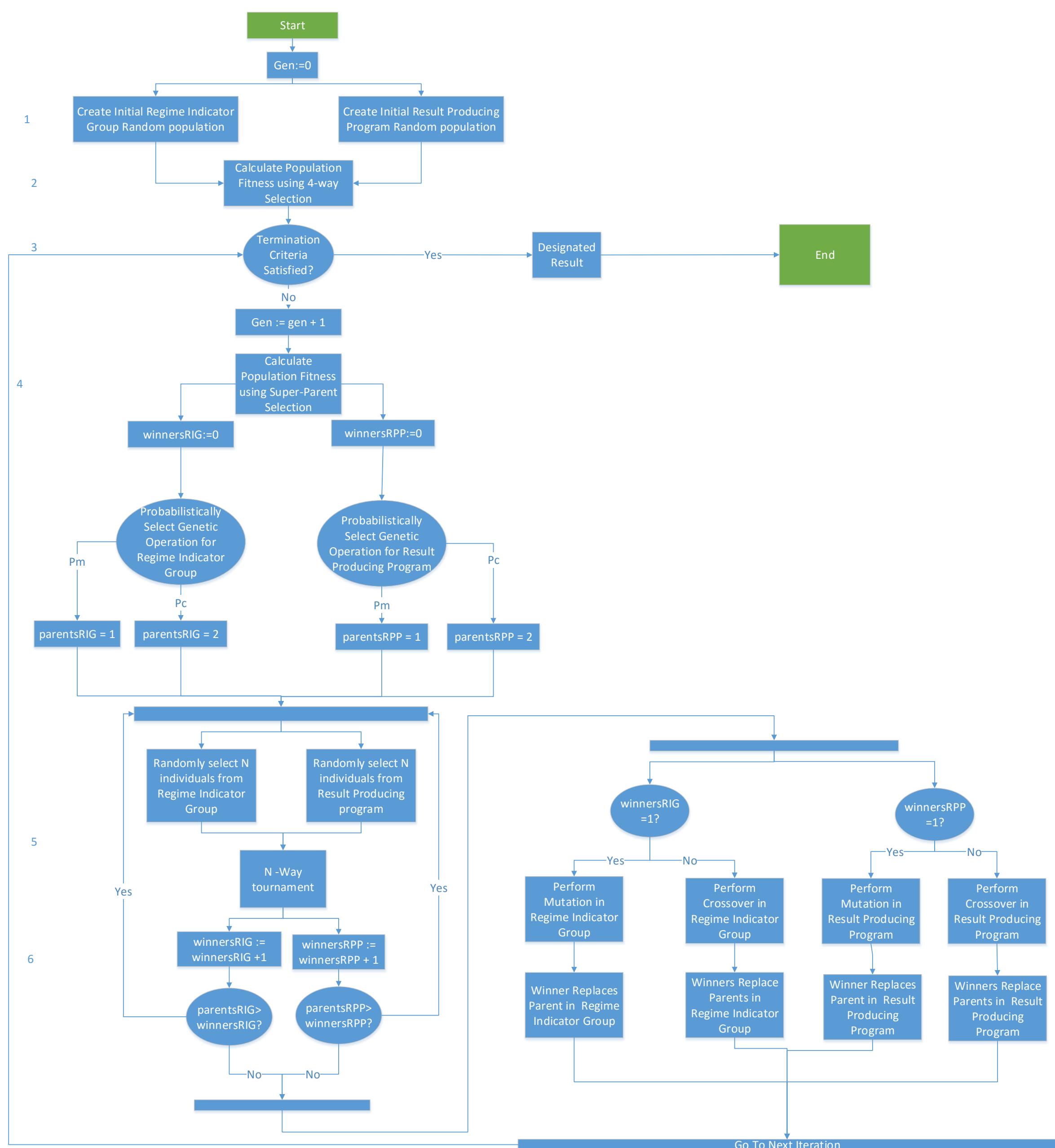



*Figure 21*. ADT algorithm flow chart.

Two different approaches to evolving the regime indicator population were developed and tested. In the first approach, the RIG population was completely decoupled from the RPP population; each population evolved independently. For a fitness calculation to occur, an individual from the RIG population must be selected and paired with an individual from the RPP

population at calculation time. An alternative approach, also implemented, permanently couples an RPP program with an RIG program, similar to the coupling of a results producing branch and a function defining branch in ADF. Note that the decoupled approach allows the total population size of each program group to be definitely separately, while the couple approach requires the same number of RPP individuals as RIG individuals. This flexibility may be valuable for domains where more resource must be directed to one type of program, though this capability did not prove an advantage in the domain addressed in this dissertation. These two approaches are discussed in greater detail below. Both approaches were compared in an initial set of experiments, with the best approach used exclusively in the final set of experiment.

## Fitness and Selection

The primary difference between the proposed approach and canonical GP is the use of and interaction between two program branches: the regime detection branch and the results producing branch. Tournament selection is used exclusively as the proposed approach does not easily lend itself to rank based selection. As typically done in tournament selection, several individuals are selected from the RIG and RPP populations. The individual with the best fitness in each population is chosen for further propagation.

### Decoupled approach to selection.

Decoupled evolution maintains independence between the RPP and RIG program populations; these two populations evolve separately. In order to calculate the fitness of an individual program, a program from the other branch is required. Two variations on selection are used: super parent tournament and four-way tournament. Probabilistic choice can be used to determine which approach is used in each instance.

***Super parent.***

In the super parent approach, the best (elite) results producing program and best regime detection program from the prior generation are used as the “other” program in every fitness calculation for the subsequent generation. In this approach, the fitness of any one program can be interpreted as the incremental effect that program will have on the best results so far. This approach also allows the calculation of the full population fitness prior to selection, a feature necessary for other selection strategies such as rank or fitness proportional selection.

The super parent approach is not possible for the initial fitness evaluation in generation 1, as no best programs yet exist. In this case, another approach described below, four-way tournament, may be used, or a simple random selection of individuals may be performed.

***Four-way tournament.***

Regime indication often has no applicable fitness function, as regimes are not necessarily observable. Therefore, the fitness of a regime indicator program is defined as the fitness of the result producing program using its functionality. As no independent fitness measures exist, the regime indicator fitness is evaluated on a relative basis, looking at its impact on two (or more) result producing programs.

To implement four-way tournament selection, two regime groups are selected at random to participate in a 4-way tournament. Two RPP programs are also selected at random for participation. Both regime indicator functions are applied to each main program to determine which of the two indicator groups better predicts the actual regime. A total of four fitness calculations will therefore be made. Each regime indicator will have two fitness score, one related to each main program. The regime indicator group with the higher total fitness score will

be considered the winner of the tournament. Likewise, the main program with the higher total fitness score will be considered the winner of the tournament. The winners in each group will be selected for mating with the winners from a subsequent tournament.

Comparison between the effects of two different regime indicators serves the same purpose as dual sliding windows in DyFor GP. The more fit a regime indicator, the more likely it will propagate to the next generation. As each indicator group contains logic to determine any potential regime, these do not need to be saved off for later use, as is done in DyFor GP, but can continually evolve. Allowing continued evolution of regime indicator groups should provide improved fitness in subsequent generations by effectively increasing the number of program generations.

**Coupled approach to selection.**

The coupled approach permanently associates a single RPP with a single RIG individual. Each program still only participates in evolutionary operations with other programs of the same type, but no selection step for joining a regime determining program with a result producing program is necessary. The programs shown in Figure 19 and Figure 20 are combined in the coupled approach into a single program, as shown in Figure 22. The architecture of the joined program is essentially identical to the architecture of an ADF program, Figure 18, with the inclusion of regime specific, template oriented, behavior.

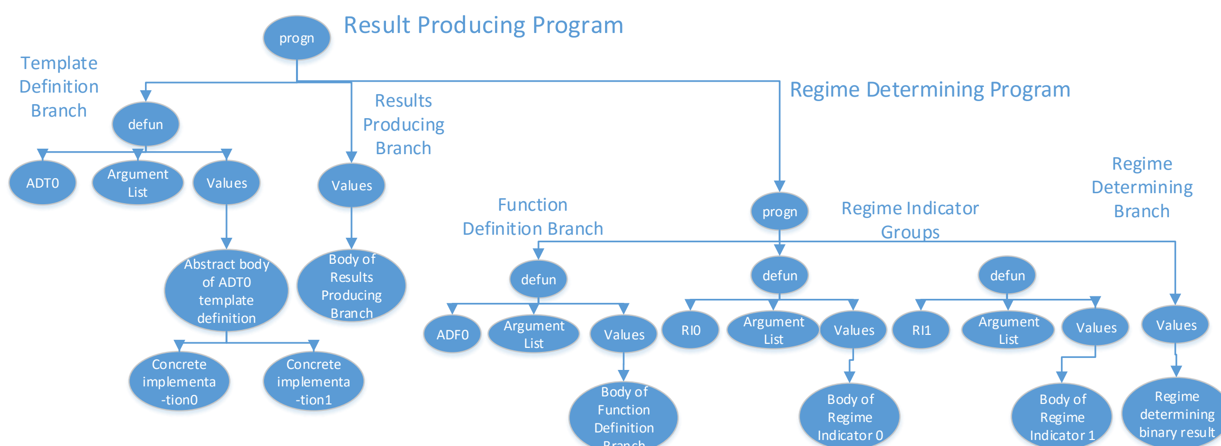



*Figure 22.* Tree structure for coupled ADT.

***N-way tournament.***

For the coupled approach, no selection step is needed to pair regime determining programs with result producing programs. Therefore, a standard N-Way tournament is appropriate whenever program selection is required. In this type of tournament section, N programs are randomly chosen with the program with the best fitness determined the winner.[8]

**Fitness calculation.**

Fitness evaluation requires both a result producing program and a regime detection program. For each evaluation, the regime is first determined by executing the applicable regime determining program. The RPB program is then evaluated in the context of the selected regime. Any included templates in the program will be evaluated using the concrete implementation for the predicted regime. Actual fitness is determined by the domain specific calculation used, applied to all points in the applicable data window. For symbolic regression and synthetic time series prediction, this final fitness is the mean squared error or mean error compared to the

[8] N is typically between 2 and 4.

known correct value. The final fitness value is generally the average error over all individual calculations.

***Exponential moving average.***

Fitness calculations of predictions taken over a data window typically average the results over the entire window. This approach, essentially identical to a simple moving average of period equal to the data window size, gives equal weight to all predictions; those predictions at the start of the period impact fitness the same as the most recent data points. When regime change is encountered, this equal weighting may hurt subsequent predictions, especially when a large data window is used. DyFor GP uses an adjustable sliding window to better handle this situation. A method proposed here is to use an exponential moving average fitness calculation instead of a simple moving average.

In an exponential moving average (EMA), more recent values are weighed higher than older values. The weighing of older data points decreases exponentially and approaches, but does not reach, zero. EMA is frequently used in time series prediction to model the decreasing impact of earlier shocks to the series. In the current context, shocks are prediction errors, as determined by the applicable fitness function. As was already shown in Equation (5), the EMA is calculated recursively, its value being a simple proportioning between the current series value and the prior EMA. The proportion allocated to each of the two values may vary as needed in order to give more or less weight to recent values.

**Automatically defined templates.**

The first and simplest modularity approach presented in this dissertation, automatically defined templates (ADT), is patterned after automatically defined functions (Koza, 1994), the pioneering GP modularity method. In ADT, one or more abstract template methods are specified

as part of the system parameters. These template methods become a part of the available program primitives and are evaluated at execution time using a regime specific implementation. As in ADF, these template methods evolve separately from the result producing branch, which may include the template method(s) in its available function set. Each program contains multiple implementations for each defined template, one for each potential regime. In ADF, by contrast, each program contains exactly one implementation of any function.

#### Crossover.

As in ADF, crossover in ADT is constrained to program branches of the same type. If an ADT is chosen for crossover, another ADT from the other program must also be chosen. If a program contains multiple ADTs, only one is chosen for crossover. Crossover may therefore occur between ADTs of different arities. Each ADT contains an implementation for each regime. Crossover occurs only between program trees implementing the same regime.

### Automatically acquired templates.

The second modularity approach presented, automatically acquired template (AAT), is patterned after module acquisition (Angeline & Pollack, 1993). Templates methods are discovered during program evolution and extracted from the evolving program into a shared library. AAT allows templates to be shared across programs within the population. This form of shared code is not available in ADF or ADT.

As in MA, AAT randomly selects a node in a program from the population and extracts that sub tree as a new abstract template method. The extracted code is replaced by a call to this new method in the original program. The extracted template method is inserted into a shared, global library. Separate results producing and regime determining libraries are maintained, associated with the two respective program branches.

Module acquisition enabled sharing code modules through the crossover operations. Similarly, when a subtree containing an AAT call is selected for crossover, the AAT call may be exchanged with the other program. Differently from MA, AATs also continue evolving. In the case where two AAT nodes are selected for crossover, the two library methods will be used in the crossover instead of the RPP program. As multiple programs may be referencing these shared templates, these evolutionary changes may be propagated across a large number of programs.

Similar to module acquisition, AAT compression does not add the newly extract module to the list of available program primitives. Adding these extracted methods primitives would likely decrease program efficiency, as the number of library functions can expand quickly. MA also defines an "expand" method that replaces any module calls in a single program with the module code. This is done to avoid a proliferation of modules and to add back some diversity in the program population that was removed by the extraction operation. If a module is expanded and no references to that module remain, it is removed from the program population. AAT also implements this feature.

**Crossover.**

A major difference between AAT and MA is that in MA, extracted modules are immune from any further modification. Kinnear (1994) listed this as a possible deficiency in MA compared to ADF. In AAT, by contrast, further module evolution is allowed. During AAT crossover, if an AAT and a non AAT node are selected, those nodes and their contained subtrees are exchanged. The AAT implementation simply moves between programs. This approach would not allow for further modification of AATs. To allow for further evolution, if two AATs are selected for crossover, the AAT implementations are exchanged and not the calling

programs. As in ADT, regime implementations will only crossover with the corresponding regime implementation from the other program.

**Sufficiency**

A core principle of genetic programming is sufficiency: the primitive operators available should be capable of solving the problem at hand (Koza, 1992). As financial time series is a principle domain of this research, a reasonable selection of financial was developed as part of the implementation based on reference experiments. Common mathematical operators used to model time series are also included. These primitives are described in Appendix C.

**Specific Methodology**

The methodology described above can be illustrated in the following simplistic example. The basic genetic programming framework is not described here. A standard GP framework, such as discussed in (Koza, 1992) with modifications described above should be assumed. Only features specific to this dissertation are discussed in greater detail below. An additional example of stock market prediction along with a template oriented implementation is discussed in Appendix B.

The following example aims to evolve a population of Boolean expressions to determine whether to be invested in the stock market at any point in time. The fitness measure is not relevant to this example, but can be assumed to be simple profit over a discrete time horizon. Investment decisions are based on a long-flat Boolean indicator dictating either full investment or no exposure. Four regimes are initially defined by the analyst. Each regime indicator group must therefore contain two indicators, each returning a Boolean value of true or false. Regimes can be determined by the indicator group values in a simple manner as illustrated in Table 1.

Table 1. Regime Determination Using Two Indicators

| Indicator 1 | Indicator 2 | Binary | Regime |
|---|---|---|---|
| false | false | 00 | 1 |
| false | true | 01 | 2 |
| true | false | 10 | 3 |
| true | true | 11 | 4 |

Each indicator group can be seen as a single expression tree involving two indicator functions concatenated to form a binary number. The internals of each indicator expression is constrained only by the available primitive functions and by the strongly-typed Boolean return type requirement.

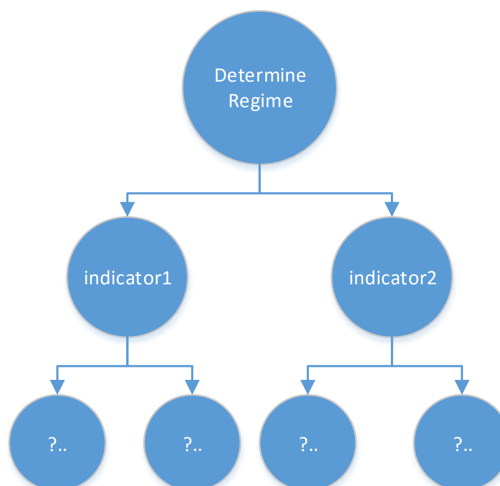



*Figure 23*. Sample regime determining program tree. The bottom nodes represent the parameters to the Boolean indicator functions and may be any evolvable expression tree.

**Automatically defined template example.**

Automatically defined templates (ADT) is in most ways identical to automatically defined functions (ADF). ADT requires prior specification of the number of template-defining branches. The return types of the templates and number of parameters are also determined by the

analyst. Each template method is then made available to the result producing branch as a primitive operation. The principal difference between ADT and ADF is that ADT contains a distinct implementation of the function for each regime.

At some point in program evolution, the following Boolean expression may exist:

```
MA(20,0) > MA(60,0) and  MA(20,-20) > MA(00,0) and ADT0()
```

MA() is a simple moving average calculation defined as:

```
Integer MA (int Days, int Offset)
Days: calculate moving average over this number of periods (i.e.
60 day moving average)
Offset: take the moving average value a fixed number of days in
the past (i.e. Value of moving average 15 days ago)
```

ADTO is an automatically defined template with four regime specific implementations. The evolved program can be seen as a combination of regime specific logic (ADTO) and fixed logic (everything else). The example program is shown in Figure 24 in tree structure. ADT0 is implemented for each discovered regime, as shown in Figure 25.

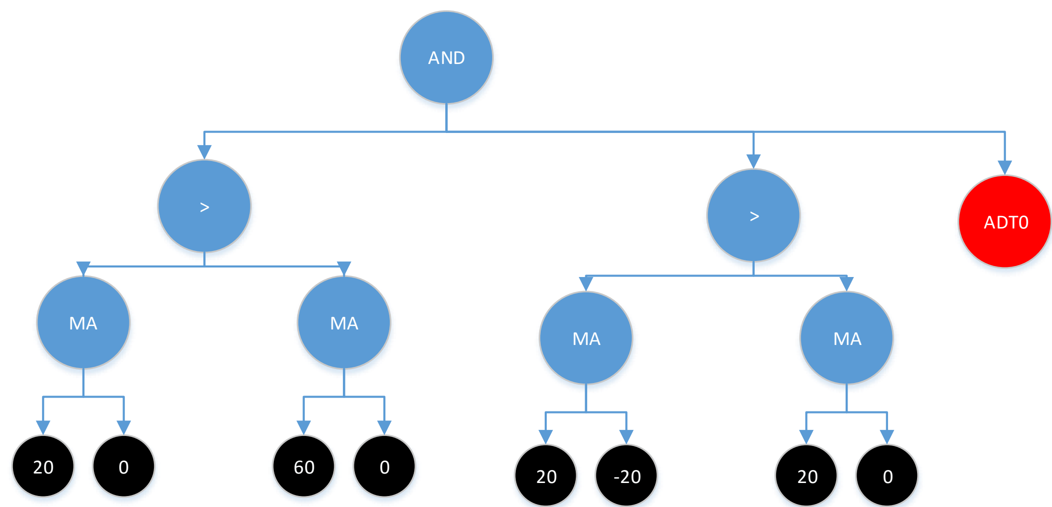



*Figure 24.* Program tree incorporating an abstract method.

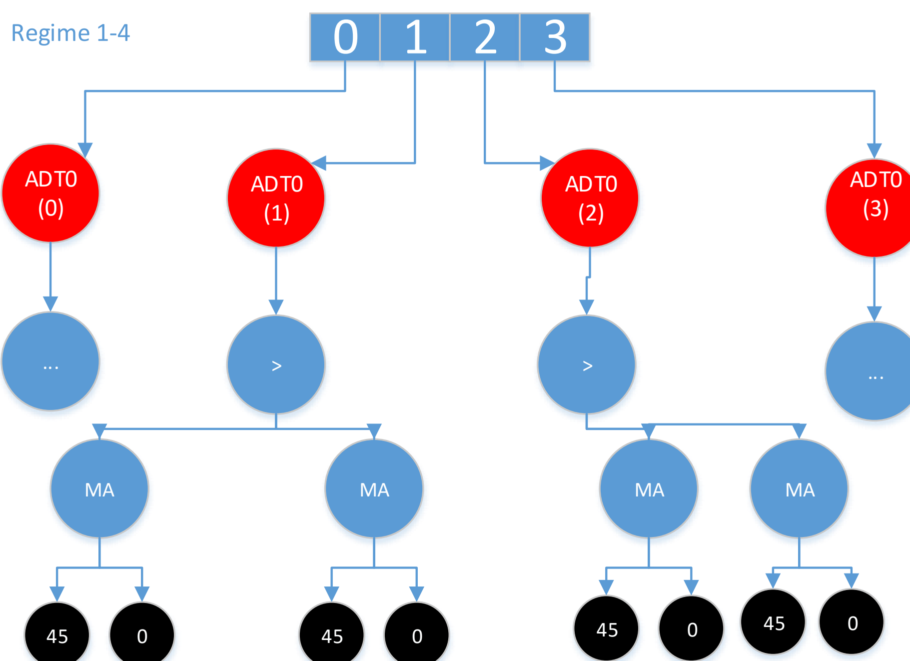



*Figure 25*. Regime specific template implementations. A numeric regime indicator determines which implementation is executed.

The program in Figure 24 can be interpreted as a single template method containing one abstract operation (ADT0) and additional concrete operations (everything else). If the root AND node was a child of another parent node (i.e. used by another operation in a larger program), either the AND node or the parent node could be considered the root of a template method. Neither interpretation will affect the functionality of this approach.

**Automatically acquired templates example.**

The second form of modularity developed in this dissertation is automatically acquired templates (AAT). This approach is modeled after module acquisition (MA) (Angeline & Pollack, 1993). In MA, a node is randomly chosen in a program tree for extraction. That node's subtree is removed from the results producing branch and is used to create a new function in the module library. The extracted code is replaced by a call to the new function. In MA, any nodes more than given depth below the root of the function are not included in the new function, but are instead

defined as parameters to the new function, and therefore kept part of the results producing branch (Angeline, 1994).

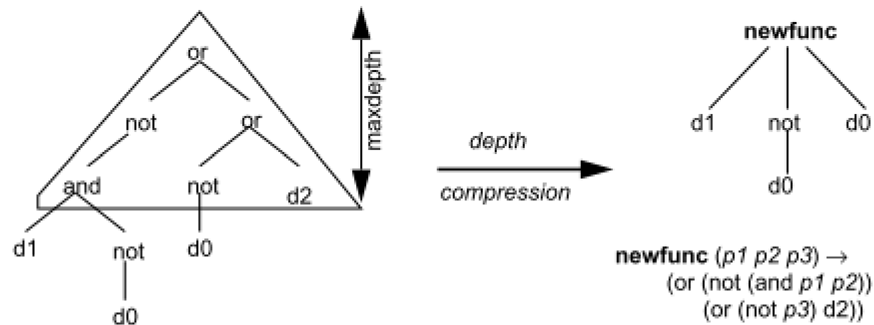



*Figure 26*. Module acquisition compression operation. A depth limited subtree is extracted and a new function, newfunc, is defined. newfunc includes parameters for each branch not included in the function because of the depth limit. Taken from (Angeline, 1994, p. 13).

AAT includes a similar approach, but, as in ADT, regime specific implementations is created for each extracted function. Further evolution of the extracted modules is allowed within the crossover constraints discussed earlier in this paper. Extracted node depth limit was not incorporated into the current implementation of AAT, but could be added in the future.

AAT is illustrated in the Figure 27 through Figure 30. Figure 27 shows a program tree similar to the example in Figure 26.

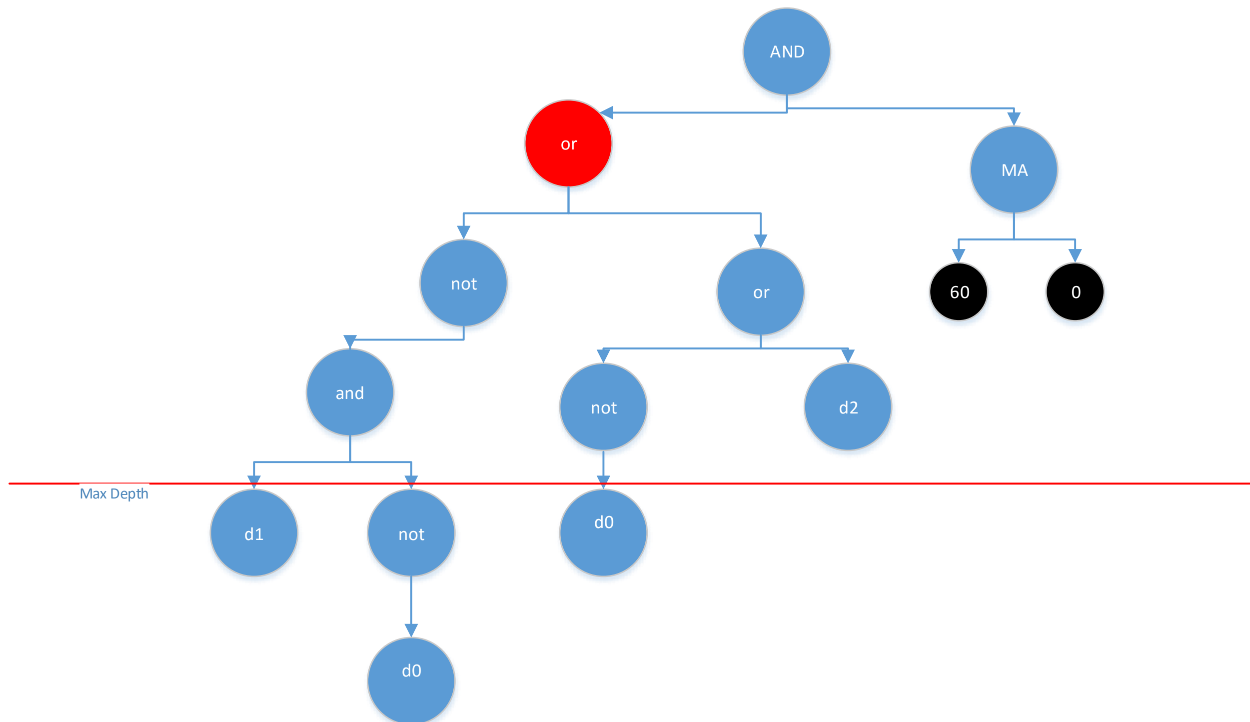



*Figure 27.* Program tree prior to AAT extraction. Target node for extraction and maximum depth are show in red. Depth limit is not implemented in the current version of AAT.

The left sub tree, rooted at the topmost OR, is chosen for extraction into a separate template method. The original tree is modified as shown in Figure 28.

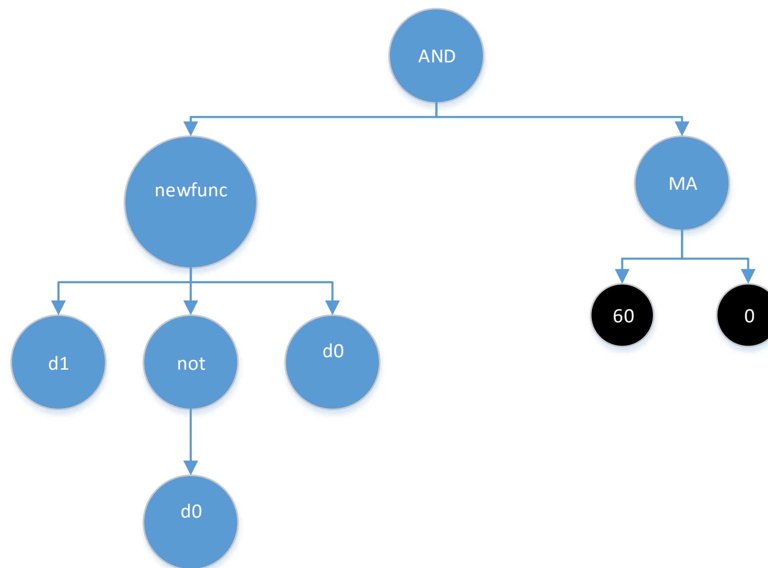



*Figure 28.* Program tree after AAT extraction.

Three parameters were created for the new function, using the depth limiting extraction operation of module acquisition. The new function will therefore accept three parameters and is shown in Figure 29.

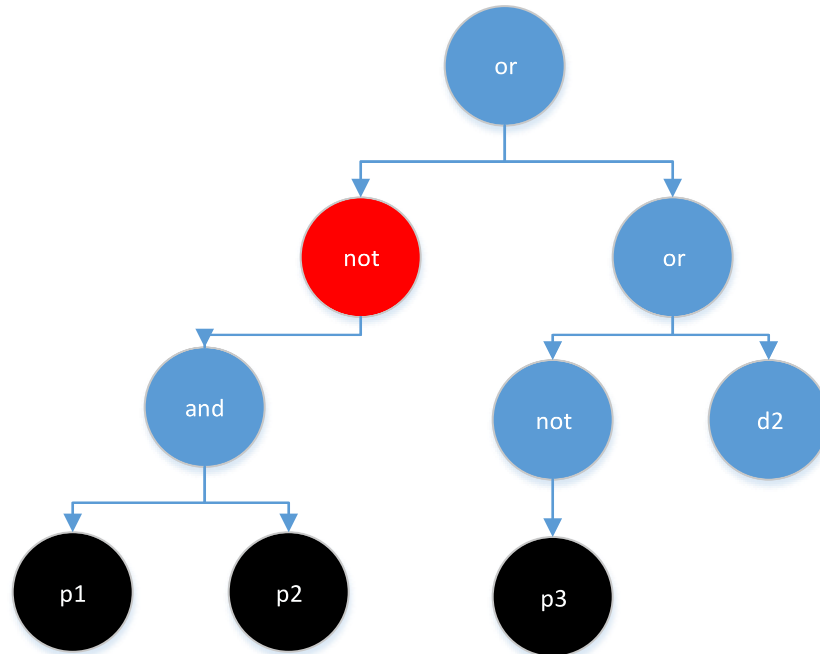



*Figure 29.* Function tree extracted by AAT. The "not" node, shown in red, is chosen for abstraction.

A template method is now abstracted as a separate, evolvable, function. Further pattern-based modularity is now applied to the new template method. Each node within the method is assigned a random possibility for abstraction. Those nodes within the new template method chosen for abstraction will reference regime specific implementations. In this example, the leftmost NOT operation is selected for abstraction. Therefore, a concrete function of Boolean type must be implemented for each regime as shown in Figure 30. The regime specific functions are seeded with the original abstract method code but each will evolve independently. P1 and P2 are the same nodes from the original abstract method.

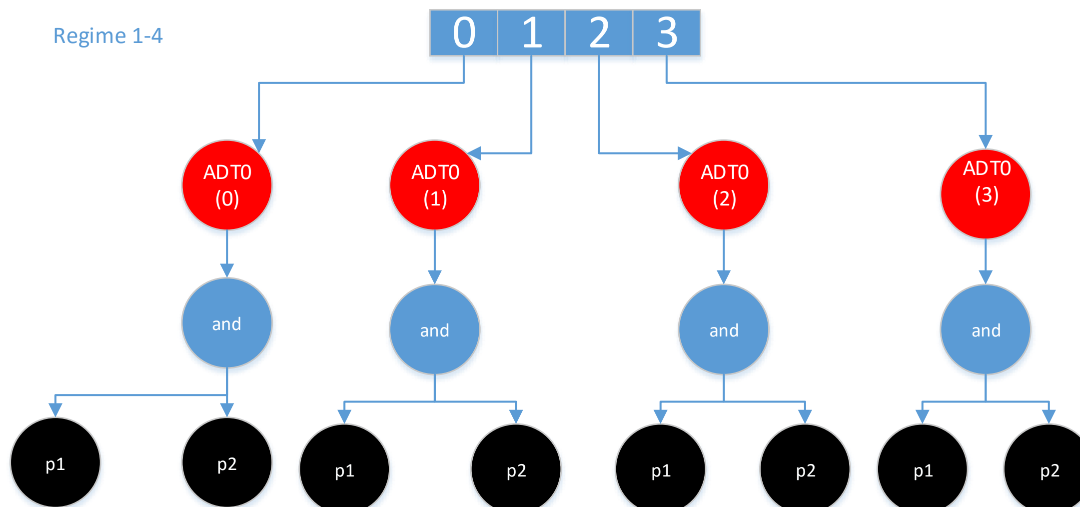



*Figure 30.* Regime specific template implementations created by AAT extraction.

**Implementation.**

The methodology described above was implemented and validated through the creation of a genetic programming system. GP systems have been successfully built using a variety of languages and platforms. Due to the expertise of the researcher, a Java system was developed. Using Java also allows a large number of third party libraries, such as for technical analysis or other domain specific purposes, to be used.[9]

Many early tree-based GP implementations used LISP as the program representation, as LISP syntax corresponds directly to the underlying abstract syntax tree (Koza, 1992, p. 81). Clojure is a dialect of LISP that runs on the Java virtual machine and integrates with the Java language (Hickey, 2014). Initially, the implemented system was built to evolve executable Clojure programs. The evolved code was able to incorporate Java libraries to reference domain specific functionality such as econometric functions.  However, initial performance profiling

---

[9] Python and C also provide a large number of third party libraries suitable this purpose.

determined that the overhead of evaluating Clojure programs from a Java framework and having those Clojure programs call other Java methods, while functional, performed poorly. Therefore, an alternative approach evaluating the Java tree representation directly was implemented. This method performed extremely well.

Each evolved program branch is represented internally as Java tree. Evolutionary operations will take place on Java object trees. Each node in the tree will contain objects that map directly to the available primitive set. To determine program fitness, the expression tree is evaluated, beginning at the root node and proceeding in a depth first manner incorporating eager node evaluation.

The following symbolic regression example attempts to fit a function to approximate sample data. In this example, the sample data is generated by the function $x^2 + x + 1$ in the domain [-5 ... 5]. In additional to typical GP parameters such as population size, crossover probabilities, selection methods, etc., the available primitives must be specified.

Two types of primitives are defined: terminals and functions. Any Java class that implements the interface Terminal and Function may be included as an available primitive. For this example, a set of terminals and functions are defined and are shown in Figure 31. These two sets combine to make up the primitives available to the evolutionary process. Other domains, such as financial prediction, will include a different, and generally much larger, primitive set.

```
Terminal[] terminals = new Terminal[]{
   new RandomInteger(lowRandom, highRandom), new Variable("x")};

Function[] functions = new Function[]{
            new ClojureAdd(), new ClojureDivide(), new ClojureSubtract(),
            new ClojureMultiply()};
```

*Figure 31.* Primitives set used in GP example.

Each of the concrete terminal classes must implement the Terminal interface. Each of the concrete function classes must implement the Function interface. This relationship is illustrated in the class diagrams in Figure C1 and Figure C2 in Appendix C.

As an example, at some point during evolution, the following program is evolved, represented by the Clojure expression (+ (+ 5 x) 5) and shown in Figure 32 in tree structure.

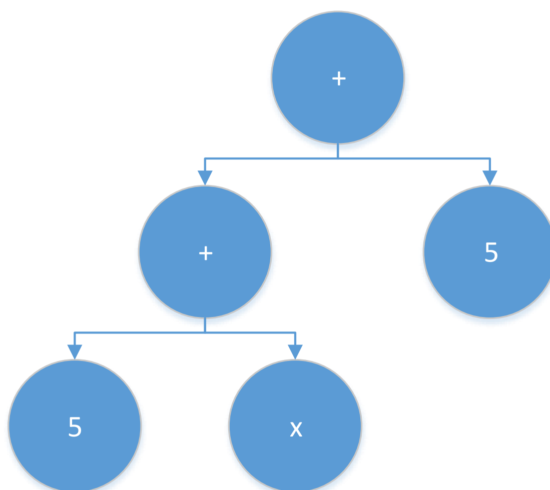



*Figure 32.* Tree structure representing the sample Clojure expression (+ (+ 5 x) 5).

The java implementation builds a similar tree, but uses the classes shown in Figure 33.

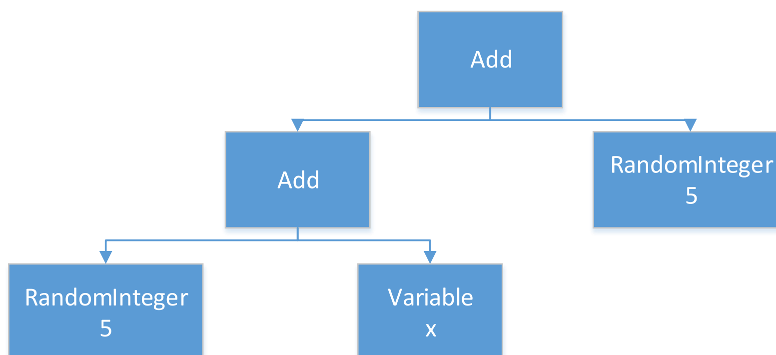



*Figure 33.* Java implementation of a sample S-expression tree.

Directional linkages are implemented via a parameter array in each function of the same size as the arity of that function. For example, the Add class contains an array of two primitives, representing the two parameters to the addition function.

It is this Java tree structure that actually undergoes the evolutionary operation. For evaluation, the root node is passed to an evaluation processor, along with the relevant test data, for fitness evaluation. The evaluation processor evaluates the root Add node and eagerly evaluates all child nodes, before returning the evaluation result. In this case, the test data is simply the integers in the range -5 to 5, along with the actual value of the function $x^2 + x + 1$. Using a fitness function that minimizes average error, this individual program would have an error of 7.36, as shown in Table 2. A simple regression example such as this will generally converge to the correct answer quickly.

Table 2. Sample Fitness Evaluation Using Mean Error

| X | Calculated | Expected | Error |
|---|---|---|---|
| -5 | 5 | 21 | 16 |
| -4 | 6 | 13 | 7 |
| -3 | 7 | 7 | 0 |
| -2 | 8 | 3 | 5 |
| -1 | 9 | 1 | 8 |
| 0 | 10 | 1 | 9 |
| 1 | 11 | 3 | 8 |
| 2 | 12 | 7 | 5 |
| 3 | 13 | 13 | 0 |
| 4 | 14 | 21 | 7 |
| 5 | 15 | 31 | 16 |
| Average Error: | | 7.3 | |

As part of this dissertation, an implementation of DyFor GP was created based on the specifications in (Wagner, 2005) and (Wagner et al., 2007). A new implementation was needed as no extensible version of DyFor GP was available and to provide a common implementation

platform for realistic comparisons. ADF was also implemented and used as a baseline comparison for project experiments.

The template method and most other gang of four patterns is typically implemented in object oriented languages using subclasses (Gamma et al., 1995, p. 325). Many functional languages, such as LISP and Clojure, do not use literal subclasses. To simulate subclasses and regime specific behavior, the regime is first determined and passed as a parameter to the result producing program. When a template node is encountered for evaluation, the appropriate regime implementation, as indicated by the regime parameter, is selected and evaluated.

**Optimizations.**

Several techniques were used to improve performance of the prototype. These optimizations are not specific to the approach discussed in this paper and may have been taken from existing methodologies where noted. The techniques here are just a small portion of the possible optimization that can be done for improved performance of a software system such as this.

***Strongly typed GP.***

Canonical GP often incorporated the closure principle, which holds that any function result or terminal value may be used as the input of any program function. While this is convenient and avoids many implementation challenges, this approach was determined inappropriate for the various financial functions needed for realistic market data scenarios.

The prototype implements strongly typed genetic programming (Montana, 1995), where each terminal and function returns a specific type and each function parameter is defined as a specific type. Boolean and Numeric were the only data types used in the implementation.

***Garbage collection in AAT.***

When an expanded AAT library module is no longer used by any program in the current population, it can be permanently removed from the library, there being no way for it to be reincorporated into a future program. A garbage collection process was incorporated to efficiently implement this feature. Every N (a configurable parameter) generations, the library is checked for non-referenced programs and these are removed from the library. This technique was necessary to efficiently manage memory, as libraries were observed to grow quickly and to contain mostly unused functions.

***Maximum predicted values.***

In time series prediction, and especially in symbolic regression, the MSE of a bad prediction can easily approach infinity. This prediction can negatively skew the overall results and invalidate overall averages. Instead of throwing these predictions out as outliers, a maximum predicted value was included as a configurable program parameter. This approach, admittedly, requires some domain knowledge of the series to be predicted and the underlying data generation process. For this study, a maximum prediction of +/- 10 was used in symbolic regression and time series prediction. A maximum allowable prediction is not needed for market data investment experiments as a worst case fitness will approach zero, not infinity.

***Trivial prediction protection.***

Often, a time series prediction algorithm will converge to a random walk prediction, where the predicted value simply equals the current value. The resultant program will be equivalent to $y(x) = y(x-1)$. This often occurs in early generations of GP where no regularities are yet found. This situation can only occur if lagged values of the target series are

included in the primitive set. Therefore, a feature was implemented to filter out individual programs where a certain percentage of predictions exactly match the prior value.[10]

It can also be argued that a random walk prediction is actually a good prediction, especially if no better alternative exists. However, implementing trivial prediction protection discourages solutions converging around a random walk prediction. In the case where a random walk is truly the best prediction, the results will often achieve a near random walk (i.e. $y(x) = y(x-2)$ or approximate).

***Forbid two root crossover.***

Mulloy et al. (1996) developed a genetic operator called FONTX (forbid one tree crossover). This operator is based on the observation that any crossover of two single node programs will result in a duplicate of both parents and therefore not improve overall population fitness. Such crossovers can also promote the type of trivial predictions discussed in the preceding section. The FONTX approach was modified slightly to prohibit crossover of any two root nodes even if they are not single node programs. As crossover of root nodes will always produce two identical offspring, this larger set of forbidden crossovers prohibit a wider variety of identical offspring.

**Omniscient regime detection.**

Regime determination is not always obvious, even to an analyst looking at the raw data. Other times, regime boundaries may be evident, but may still be difficult to programmatically

[10] A parameter of 95% was used in the experiments in this dissertation. An "exact" match includes values within a predefined variance; 0.0001 in these experiments.

determine due to noise or alternative possible regimes. A goal of this dissertation was to develop methodology incorporating regime determination and use that information to improve time series prediction predictions. To better evaluate the value of that goal, an additional experiment was included where the regime determining branch is an infallible, fixed implementation. No evolution of the regime determining branch needs to occur and the results achieved can be seen as the best case scenario, where perfect regime detection is achieved. Omniscient regime detection is not included in the market data experiments as actual regime boundaries are open to interpretation.

**Experiments.**

Several experiments using synthetic time series were described in (Wagner & Michalewicz, 2008). These experiments were reproduced and their results compared. An implementation of DyFor GP was created for this dissertation with the goal of sharing as much common code as possible so not to skew comparisons due to implementation details. DyFor specific code amounted to only of 413 out of 11,510 lines of Java source code.

***Synthetic series.***

Wagner & Michalewicz's (2008) prediction experiments included two synthetic time series. Synthetic time series were used to force distinct and observable regime changes. The series used are show in Figure 34 and Figure 35 and are described in greater detail in Chapter 4 of this document. This synthetic approach may not be consistent with any real world time series, but is a valuable controlled test of the described methodology and may be applicable to other, non-financial domains, such as control theory. In addition to the two synthetic series used for prediction experiments, an additional series was developed for use in a symbolic regression experiment.

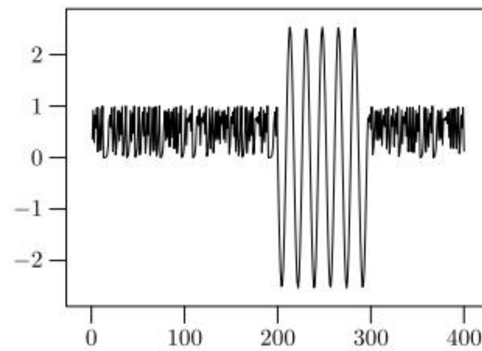



*Figure 34.* LG-OZ-LG synthetic time series used by Wagner & Michalewicz (2008).

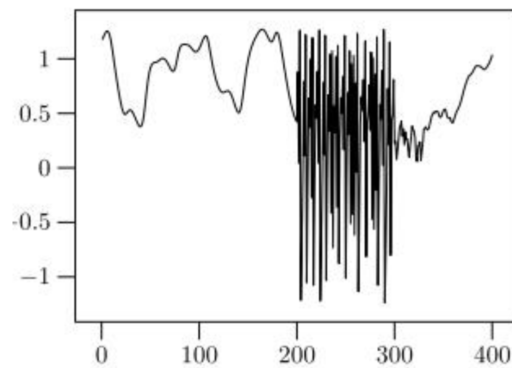



*Figure 35*. MG-HEN-MG synthetic time series used by Wagner & Michalewicz (2008).

Synthetic series experiments compared the following approaches:

1. ADT

2. AAT

3. DyFor GP

Symbolic regression experiments compared:

1. ADT

2. AAT

3. Canonical GP

4. ADF

***Market data series.***

For a real world experiment, Wagner & Michalewicz (2008) addressed US Gross Domestic Product prediction. This dissertation instead chose to consider stock prediction as a real world example. An experiment presented by Chen et al. (2008) was instead used as a comparison benchmark.

Looking at prior inconclusive results in market prediction using genetic programming, Chen et al. (ibid.) attempted a more compete study by looking at eight international stock markets and eight foreign exchange markets.[11] Parameters included those used in earlier benchmark studies and time frames were updated to include more current data. The approach used genetic programming to produce an invest/don't invest decision at any point in time. Fitness was measured by the investment gain relative to buy and hold approach.

The market data experiment done for this dissertation replicated the S&P 500 index test reported by Chen et al. (ibid.). Additional factors included in that experiment, such as short selling and comparison to approaches other than buy and hold, were not considered.

**Measuring performance.**

Several categories of performance can be measured:

1- GP performance – best measured by how quickly the solution converges to a final answer before plateauing or even worsening.

[11] The Chen et al. (2008) study was based strongly on (Allen & Karjalainen, 1999) and attempted to expand that study to multiple international equity and currency exchange (FOREX) markets as well as and to compare GP with trading strategies other than buy and hold. Both studies share many of the same parameters incorporate similar fitness functions. As the data is slightly more up to date, this study uses Chen et al. as a benchmark.

2- GP accuracy – domain specific measurements determining how close the solution comes to a known answer, or a measure of relative accuracy between two competing solutions.

3- System performance – best measured by CPU load, memory usage, and processing time.

Koza (1994) evaluated GP performance of alternative approaches in terms of solution size and computational effort. The latter measure was defined as the number of fitness evaluations required during program execution. Chen, Kuo, & Shieh (2002) define “search intensity” as a function of both population size and total generations, the product of which approximately equals the number of fitness evaluations. However, while increasing either of these parameters will generally increase GP performance, each achieves a different incremental improvement. In addition, a diminishing return may be reached when increasing either parameter. While Chen et al. give advice on determining optimal population size and number of generations, a simpler count of fitness comparisons is an appropriate measure of computational effort and was used to measure computational performance in the dissertation experiments.

Specific to regime dependent study presented here, measurements will be made to determine the proportion of regime dependent code versus shared code. A metric describing this proportion is defined in Equation (15).

$$RV = \frac{RDN}{TN} \tag{15}$$

Where: RDN = regime dependent nodes

TN = total nodes

A solution with no regime dependent code is obviously no better, and likely less efficient, than a non-regime aware solution. Ideally a balance of regime and non-regime dependent code

will be realized. This measure is only applicable to the proposed approach but helps explain the results achieved and is noted where measurable.

Many measures of GP accuracy exist in the literature. For symbolic regression prediction problems, where a single deterministic solution is known, mean squared error (MSE) is often used. However, for a more stochastic system, MSE could yield unduly high errors even if the general trend of the prediction is correct. This is also a problem with approaches that attempt to predict point in time values of a stock series. A stock prediction off by one day would likely still be profitable, but could have a high MSE when measured against the actual value. By incorporating a maximum predicted value, MSE can be used even when unboundedly large prediction may occur.

Prediction problems typically use the prediction itself as the sole measure of accuracy, such be investment percentage gain (Canelas et al., 2012) or prediction accuracy (Wagner et al., 2007). Li & Tsang (1999) used GP to determine whether a stock series would gain 2.2% or more within 21 trading days. The GP output positive or negative. The fitness measure used was to use Rate of Correctness, defined as:

$$RC = \frac{P + N}{T} \tag{16}$$

where: P = number of correct positive predictions

N = number of correct negative predictions

T = total number of predictions made

(Jin Li & Tsang, 1999).

Such as measure is only applicable to binary outcomes. A system that looks for general profitable investments would not likely use such as narrow fitness measure. Any gain over a

fixed safe return, or other risk-free alternatives, would be a desirable result. Li & Tsang (ibid.) also proposed other measures such as Rate of Missing changes and Rate of Failure and attempted to combine these three measures into a weighted fitness function tailored to the investors risk tolerance. In this dissertation, such a measure would not be appropriate and it is geared towards a determination of effectiveness between several approaches and not necessarily to maximize profit in a day to day trading environment. For stock market prediction, simple profit is used as the fitness measure and will be used to compare prediction accuracy. Transaction costs will only be considered when comparing against other studies that included such costs. Volatility and investment risk, often considered in market prediction studies, were not incorporated as these were not included in the benchmark experiments replicated in this dissertation.

GP performance was measured by total fitness and node evaluations. CPU and memory execution were not considered as these are dependent on the actual hardware used and on available primitives and their implementation.

**Resource Requirements**

The system implemented in this dissertation was developed on a Windows 10, I5 dual core CPU laptop with 8 GB of Memory. JetBrains IntelliJ 15 was used as the Java development environment. The Java system was built on top of the spring boot framework (Pivotal Software, 2016). The parameter files, included in Appendix D, are based on the Spring Boot command line format. Market data was retrieved from Quandl (Quandl, 2016) and loaded into a SQL Server Express database running on an Amazon Relational Database Services (RDS) db.t2.micro instance. Results were compiled using Microsoft Access 2016 and Microsoft Excel 2016. This document was produced using Microsoft Word 2016.

Experiments were run on as Amazon EC2 (Amazon Web Services, 2014) on Ubuntu Linux 14.04 m4.large and m4.xlarge instances or on the development laptop. Runs were generally limited to one concurrent run per CPU.

## Summary

This section described the methodology and implementation developed to incorporate software design patterns into genetic programming for better prediction of financial and other nonlinear time series. It was shown that regime change is a critical and overlooked area in time series prediction using evolutionary methods. Genetic programming, along with the template method pattern, should allow custom regime specific implementation to evolve in the context of an overall prediction methodology. Incorporating design patterns will result in generated code more similar to what would be produced by human programmers. The next chapter describes the results of experiments using the methodology described in this chapter.

# Chapter 4
# Results

This section presents the results of experiments performed for this dissertation. The tests are divided into two categories:

1. Synthetic Test
2. Market Data Tests

The synthetic data series were used to provide targets with known data generation processes so that accurate fitness can be measured. These series are also used to simulate regime change, as such a scenario is often not evident actual real world data sets.

The market data sets enable testing the proposed methodologies on real world data, in situations similar to what would be used in practice. Such series often contain a random component and therefore exact prediction of these series cannot be made. Alternative prediction methods can, however, be compared on a relative basis.

The exact methodology used is described with each experiment. Variations on the methodology described in Chapter 3 were necessary to replicate certain benchmark studies that may have used different approaches to handling out of sample data.

## Synthetic Series

Three synthetic series were analyzed. Two of these are chaotic series, reproduced from prior studies on time series prediction. The third series is developed for this study as a simpler, non-chaotic series, also containing abrupt regime changes. Several approaches were evaluated for each synthetic series. Not all approaches were considered for each experiment. These GP approaches include:

1. Canonical Genetic Programming (GP)
2. Automatic Defined Functions (ADF)
3. Automatically Defined Template (ADT)
4. Automatically Acquired Templates (AAT)
5. DyFor GP

Both market series and synthetic series have been used in past studies on prediction using evolutionary algorithms. Where applicable, the results achieved in this study are compared with those presented in earlier works.

**Data analysis.**

Three synthetic series were evaluated. Two of these series were used in (Wagner & Michalewicz, 2008). These series, called LGOZLG and MGHEMMG, are built by splicing together two different series in order to simulate a change to the underlying data generation process. These two series can be considered chaotic, as their shape is highly dependent on initial conditions. An additional series developed for this research, called SINCOS, is included as a simpler example of regime change, not dependent on initial conditions or lagged values.

***SINCOS.***

The first synthetic series, SINCOS, is a non-chaotic series that is not dependent on initial conditions nor prior series values for its data generation. A hurdle often encountered in automated prediction is the convergence on trivial, though approximate solutions, such as predicting $y(t+1) = y(t)$. (Mulloy et al., 1996). In many cases, such as stock market series, this is a reasonable prediction. For the synthetic series described above, this may be a reasonable approximation, but it is not indicative of the underlying data generation process. A synthetic series was therefore created that avoids this problem by not relying on lagged values.

A SINCOS series is created by sequencing the following individual series:

$$
\begin{aligned}
&0<=x<70: && Y_t = sin(x) + \sqrt{x} \\
&70<=x<130: && Y_t = cos(x) - \sqrt{x} \\
&130<=x<200: && Y_t = sin(x-130) + \sqrt{x-130}
\end{aligned}
\quad (17)
$$

The resultant series is show in Figure 36.

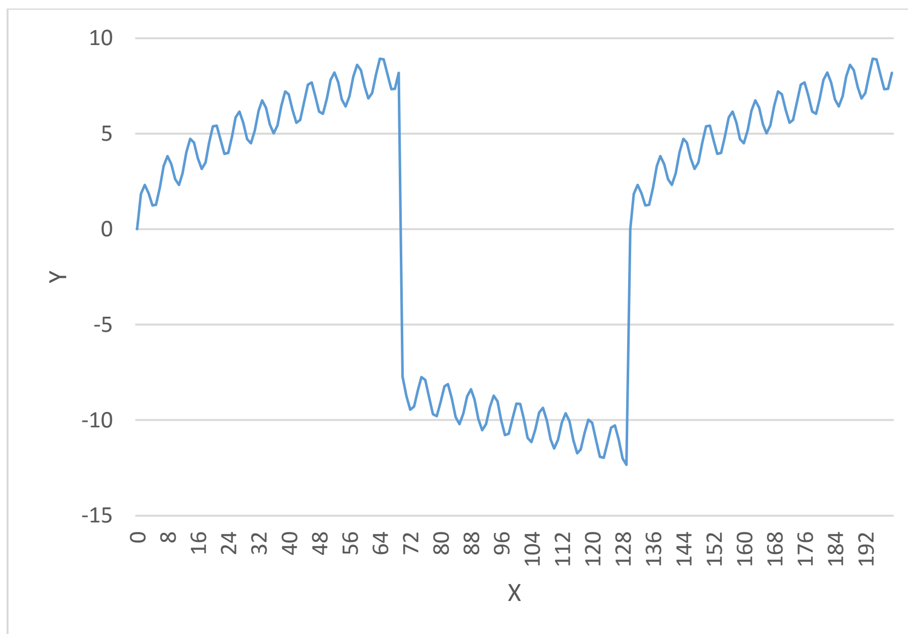



*Figure 36.* SINCOS synthetic time series.

This series is used to test symbolic regression, where an equation for a time series is discovered based on empirical data points. The entire series was used for training and no out of process prediction stage was incorporated. DyFor GP is not included in this experiment as it is only applicable to prediction problems. Canonical GP and ADF are applicable and were both included in the experiment.

*Experimental approach.*

Each evolutionary algorithm is run on the entire time series for a fixed number of training generations. Each generation will produce a best fit regression expression. Elitism is used to preserve high performing individuals across generations. No out of sample testing is incorporated and no prediction is done. The best result achieved after the final training generations are used as the overall regression.

***LGOZLG.***

This series, taken from (Wagner & Michalewicz, 2008), is created by combining a logistic map (LG) with an Ozaki simple linear function (OZ).[12] The formulas for each component of the LOGOZLG series are:

$$\text{LG: } Y_{(t+1)} = 4Y_t(1 - Y_t)$$
$$\text{OZ: } Y_{(t+1)} = 1.8708Y_t - Y_{t-1} \tag{18}$$

As per Wagner & Michalewicz (ibid.), the combined 400 length series is taken by using LG for t values 1-200, OZ for t values 201-296, and LG for t values 297-400. As the initial LG component depends on a prior series value, 0.9 is used for Y(t=0) as in the benchmark experiment. The final series is illustrated in Figure 37.

[12] This series is referred to as LG-OZ-LG in (Wagner & Michalewicz, 2008). This document uses the slight abbreviation LGOZLG.

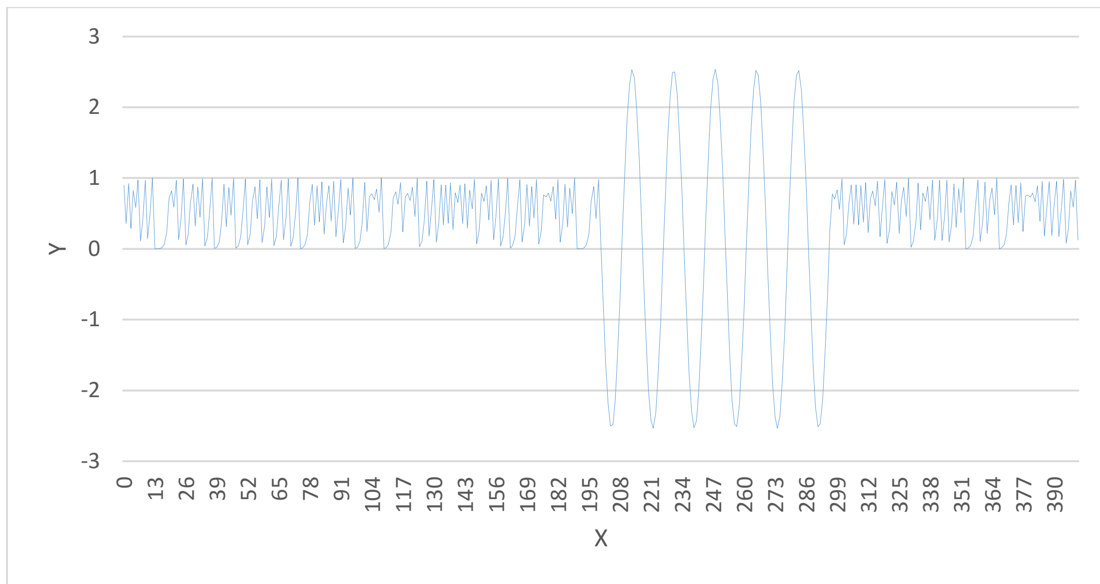



*Figure 37.* LGOZLG synthetic time series.

The experiments described by Wagner & Michalewicz (2008) perform training on points 1 through 100 and prediction on points 101 through 400. Training occurs within the confines of a single regime. Therefore, regime dependent methods such as ADT should hold no initial advantage. In addition, as regime change occurs around point 200, ADT will continue to hold no advantage as it has not yet been trained on this new regime. DyFor GP should perform well, as it simply reacts to decreasing prediction accuracy by adjusting its two dynamic windows. In order to better gauge the performance of ADT, training was done on points 150-250 with prediction done on points 251-400. This training window includes two regimes. The original 1-100 training window used in the benchmark experiment is also evaluated.

*Experimental approach.*

In both this series and the following synthetic series, initial training is done for a fixed number of generations on a static training window. Following training, a prediction is made, using the best individual from the training phase. After each prediction, another training generations is run and a new best individual selected. Training is done using a fixed size window

that slides forward with the prediction target. Following training, the prediction target is moved forward and the predication/training cycle continues until the end of the target time series reached.

***MGHENMG***

This series, also taken from (Wagner & Michalewicz, 2008), is created by combining a by combining a Mackey-Glass series (MG) with a Henon Map (HEN).[13] The formulas for each component are:

$$\text{MG: } Y_{(t+1)} = Y_t + \frac{0.2Y_{t-30}}{1+{Y_{t-30}}^{10}} - 0.1Y_t$$

$$\text{HEN: } Y_{(t+1)} = 0.3Y_{t-1} + 1 - 1.4Y_t^2 \quad (19)$$

As per Wagner & Michalewicz (ibid.), the combined series is created by using MG for t values 0-200, HEN for t values 201-300, and MG for t values 301-400. Training begins at data point t=31 to allow for the necessary number of lagged data values. As any point in the series depends on the prior 30 values, a data generation process is followed as outlined in (Wagner & Michalewicz, 2008) that first calculates 30 random values to serve as offsets. An MG series of length 1200 is then generated. The last 200 data points of this 1200 length series are used as the first MG section. This data generation algorithm will add a degree of randomness to each run. However, for consistency between test runs, a set of 30 random values were generated and reused in all subsequent runs. These random values are listed in Appendix F. An example of a

[13] This series is referred to as MG-HEN-MG in (Wagner & Michalewicz, 2008)

generated MGHENMG series is shown in Figure 38. As with LGOZLG, training will occur at period 150-250 to include two regimes.

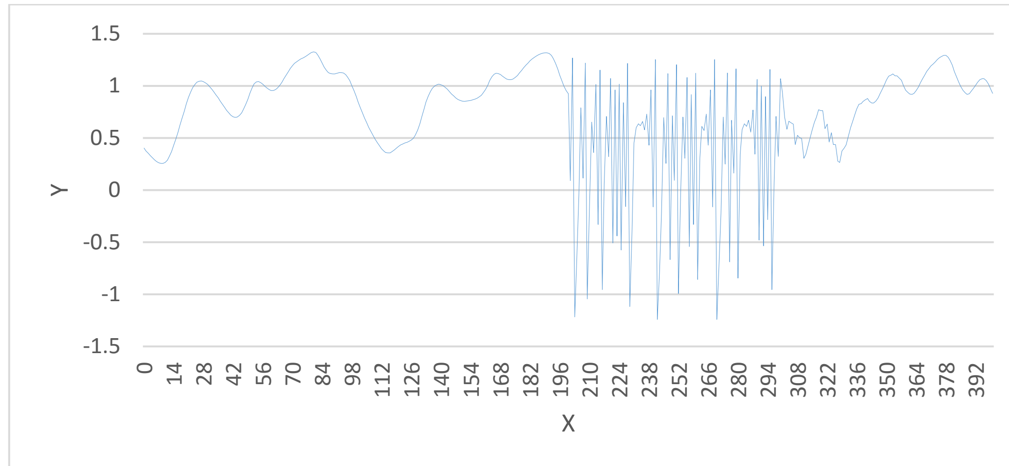



*Figure 38*. MGHENMG synthetic time series. Initial conditions are provided in Appendix F.

***Experiment parameters.***

Where possible, the same parameters used in  (Wagner & Michalewicz, 2008) were replicated. A primary difference is that in this experiment, a fixed population size is used while the benchmark study maintains a total population node size limit. A total node limit, allowing individual to grow as needed, is not appropriate without implementing additional population control features. A more common approach of incorporating maximum tree depth was used in this experiment to control program growth. This value was lowered from more traditional value of 17 to a value of 10. A lower value should force more creative and interpretable solutions and limit overfitting. In addition, as these series are synthetically generated, the optimal solutions to these series are known to be achievable within those limits.

Table 3 through Table 6 show the GP parameters used for these experiments. Differences among experiments are described immediately below the common parameters. Full experiment parameters are provided in Appendix D.

Table 3. Common Parameters Used in Experiments

| Parameter | Value |
|---|---|
| Population Size | 3000 (prediction) |
| | 5000 (regression) |
| Initialization method | Ramped Half-and-Half |
| Tournament Size | 4 |
| Crossover rate | 0.9 |
| Reproduction rate | 0.0 |
| Mutation rate | 0.1 |
| Training Generations | 41 (prediction) |
| | 100 (regression) |
| Termination | Max. generations reached |
| Training Window | Full series for symbolic regression |
| | See below for prediction experiments |
| Initial Depth | 5 |
| Max Depth | 10 |

Table 4. ADT Parameters

| Parameter | Value |
|---|---|
| Number of regimes | 2 |
| ADF arities | 1,2 |
| Training Window | 110 |

Table 5. AAT Parameters

| Parameter | Value |
|---|---|
| Number of regimes | 2 |
| Crossover rate | 0.8 |

| Parameter | Value |
|---|---|
| Reproduction rate | 0.0 |
| Mutation rate | 0.1 |
| Compression rate | 0.05 |
| Expansion rate | 0.05 |
| Minimum compression Size | 5 |
| Training Window | 100 |

Table 6. DyFor GP Parameters

| Parameter | Value |
|---|---|
| Max Window Size | 200 |
| Min window size | 20 |
| Start window size | 80 |
| Window difference | 20 |
| N | 3 |
| Save Off | 10 |

Synthetic series prediction experiments were run using two different training windows. Training window size remained 100 but the starting point was moved. Version 1 begins at point 0 while version 2 beings at point 130. The first version keeps all training in the same regime. This would appear to be advantageous for DyFor GP. The second version performs initial training across regime boundaries, seemingly advantageous for ADT/AAT, as that approach can consider multiple regimes simultaneously.

The primitives chosen were the same as those used in (Wagner & Michalewicz, 2008). However, additional ephemeral real numbers were included, as they were deemed necessary for sufficiency and were likely used in the benchmark experiment though not discussed. Protected

operations are implemented where appropriate, as is typically described in the literature. The following primitives were used:

Functions:

1. Addition
2. Subtraction
3. Multiplication
4. Division
5. Sin
6. Cos
7. Square Root
8. Exponentiation of Euler's number (e).
9. Natural Logarithm

Terminals:

1. Random Integer [-1...110]
2. LGOZLG: OffsetValue (1), OffsetValue (2)
   MGHENMG: OffsetValue (1) ... OffsetValue (31)

To enable sufficiency in regime determination, additional functions were made available to the regime determining branches in ADT and AAT:

Regime Functions:

1. Addition
2. Subtraction
3. Multiplication
4. Division
5. Logical And
6. Logical Not

7. Greater than
8. Arbitrary series offset value.
9. Period Minimum
10. Period Maximum
11. Period Standard Deviation
12. Period Average

As these additional functions should add no value to DyFor GP and perhaps detract from its performance, they were not included in the DyFor GP set of available primitives. These primitives are described in Appendix C.

**Findings.**

The results recorded in the synthetic series experiments are provided in the following sections.

***SINCOS.***

Table 7 summarizes the results obtained from the SINCOS experiment. 20 runs of population size 5000 were performed. The average and best fitness, standard devitalization, and average number of evaluations required are listed. Both coupled and decoupled ADT and AAT are tested. Samples of actual parameter input is provided in Appendix D. The full set of experimental parameters is available on the data distribution website mentioned in Appendix I.

ADT and AAT were the best and worst performers, respectively. Regime analysis showed that ADT essentially discovered the true regime while AAT did not. Both these methods required approximately double the fitness evaluations of the canonical methods when using the decoupled approach.

The improvement when using the coupled approach is significant. Coupled ADT consistently provided the best overall results in this experiment. As shown in the charts in

Appendix G, ADT correctly determined the actual regime and approximately correct regression on the actual series. Of equal import, the coupled approach requires approximately the same number of fitness evaluations as the canonical approaches, while, in the case of ADT, providing much better results.

Table 7. SINCOS Symbolic Regression Results

| Series | Runs | Avg. Final Fitness | Std. Dev. | 95% CI | Best Final Fitness | Avg. evaluations |
|---|---|---|---|---|---|---|
| | | | Decoupled | | | |
| ADT | 20 | 1.016 | 0.795 | [0.667 - 1.365] | 0.323 | 11,427 |
| ADF | 20 | 1.708 | 0.228 | [1.608 - 1.808] | 1.269 | 5,902 |
| GP | 20 | 1.729 | 0.159 | [1.659 - 1.798] | 1.361 | 6,504 |
| AAT | 20 | 1.905 | 0.433 | [1.715 - 2.094] | 1.289 | 11,362 |
| | | | Coupled | | | |
| ADT | 20 | 0.742 | 0.134 | [0.683 - 0.817] | 0.398 | 5808 |
| ADF | 20 | 1.715 | 0.250 | [1.606 - 1.856] | 1.093 | 5907 |
| GP | 20 | 1.806 | 0.209 | [1.714 - 1.923] | 1.448 | 6497 |
| AAT | 20 | 2.030 | 0.553 | [1.787 - 2.340] | 1.497 | 5767 |

*Note*. CI=Confidence Interval, calculated using the CONFIDENCE.NORM function in Microsoft Excel. ADF and GP are not impacted by coupling approach. Any observed differences are due solely to expected random variations.

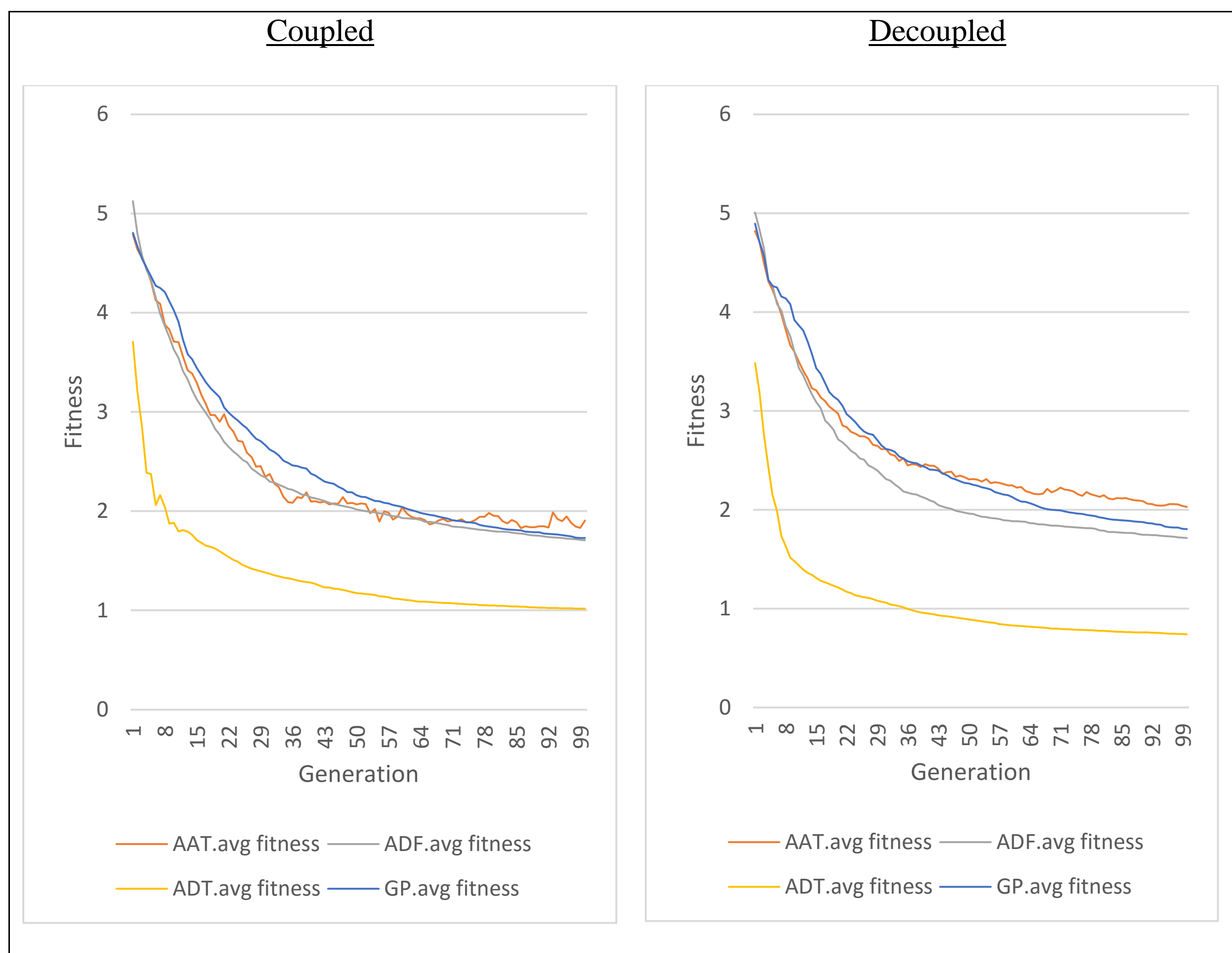



*Figure 39.* Average fitness for SINCOS symbolic regression experiment.

***LGOZLG***

In the LGOZLG experiments, ADT also outperformed AAT and a coupled approach also outperformed a decoupled approach. ADT also outperformed the benchmark DyFor GP in this test at a 95% confidence level. Contrary to expectations, Omni runs did not generally produce better results. An explanation of this fact is that evolution tended to use the additional available modularity for purposed other than the expected regime demarcation. AAT continued to perform poorly, being the only method that did not beat a random walk prediction on average.

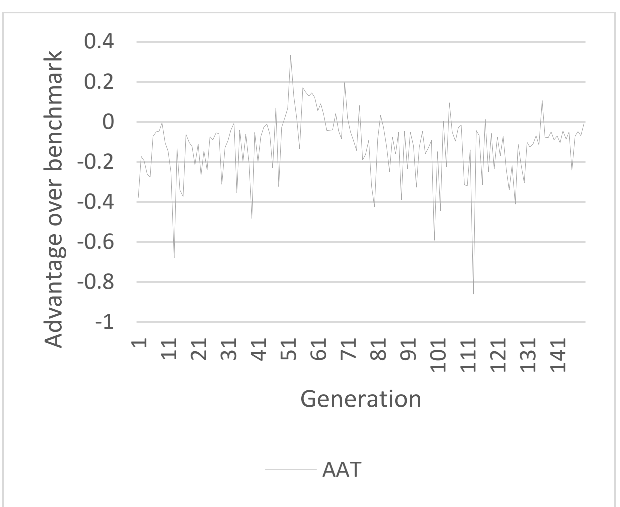
Table 8. LGOZLG Prediction Results

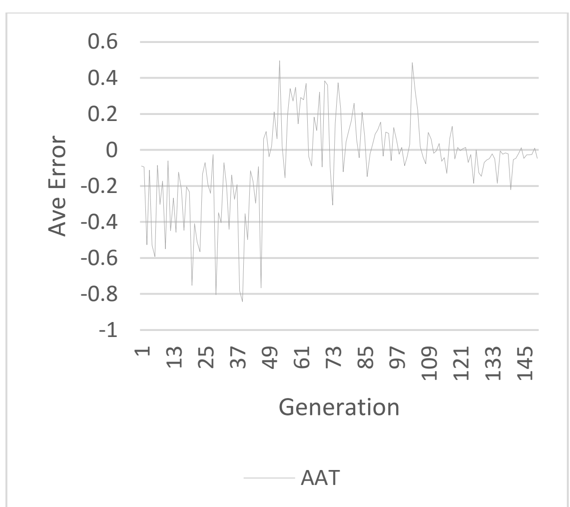
| Series | Mean MSE | Std. Dev. | 95% CI | Min. MSE | Random walk prediction[a] |
|---|---|---|---|---|---|
| DyFor GP | 0.204 | 0.104 | [0.158 - 0.250] | 0.052 | 0.313 |
| GP | 0.305 | 0.187 | [0.223 - 0.387] | 0.100 | 0.313 |
| | | | Decoupled | | |
| ADT | 0.118 | 0.066 | [0.089 - 0.147] | 0.047 | 0.313 |
| ADT OMNI | 0.143 | 0.139 | [0.082 - 0.204] | 0.002 | 0.313 |
| AAT | 0.430 | 0.286 | [0.305 - 0.555] | 0.108 | 0.313 |
| AAT Omni | 0.559 | 0.327 | [0.416 - 0.702] | 0.103 | 0.313 |
| | | | Coupled | | |
| ADT | **0.100** | 0.066 | [0.071 - 0.129] | 0.056 | 0.313 |
| ADT OMNI | 0.128 | 0.066 | [0.099 - 0.157] | 0.014 | 0.313 |
| AAT Omni | 0.377 | 0.213 | [0.284 - 0.470] | 0.106 | 0.313 |
| AAT | 0.491 | 0.444 | [0.297 - 0.686] | 0.101 | 0.313 |

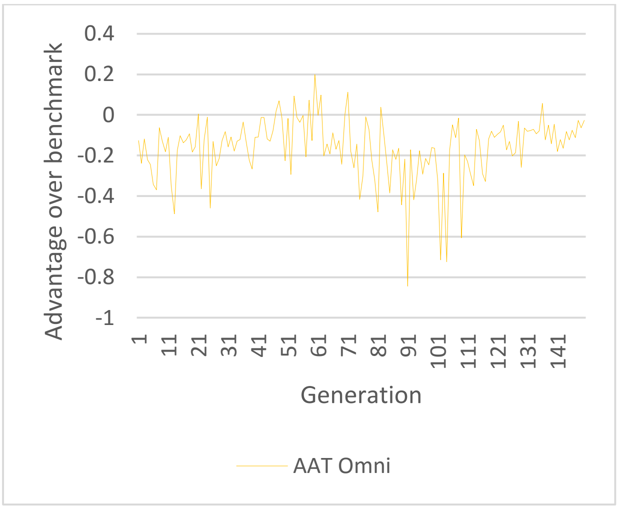
*Note*. CI=Confidence Interval; Training done on points 150-250. Predictions done on points 251-400. Coupling approach is not relevant to DyFor GP and GP.
[a] Assumes predicted next value equals current value.

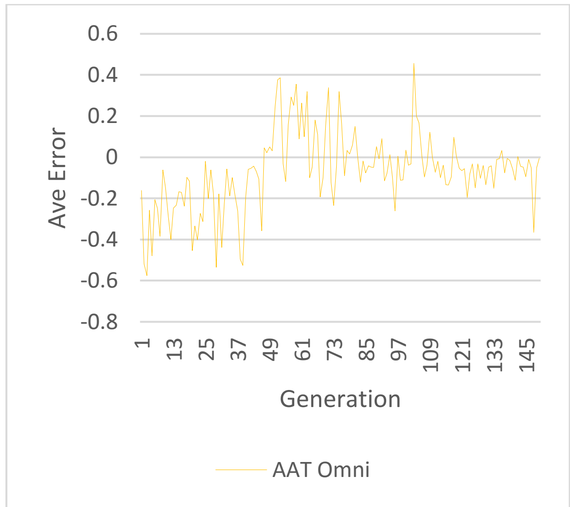
Figure 40 illustrates the relative prediction accuracy of all approaches versus the DyFor GP benchmark. While ADT achieves a much better overall score, DyFor does better in the initial prediction rounds. ADT and ADT Omni perform better during the regime switch that occurs around generation 200. These results are in line with expectations, as ADT has already seen that regime in earlier training, while DyFor GP is simply reacting to worsening predictions. Further confirming expectations, DyFor performs somewhat better than canonical GP, indicating that its regime handling capabilities do have a positive effect

**Decoupled** | **Coupled**



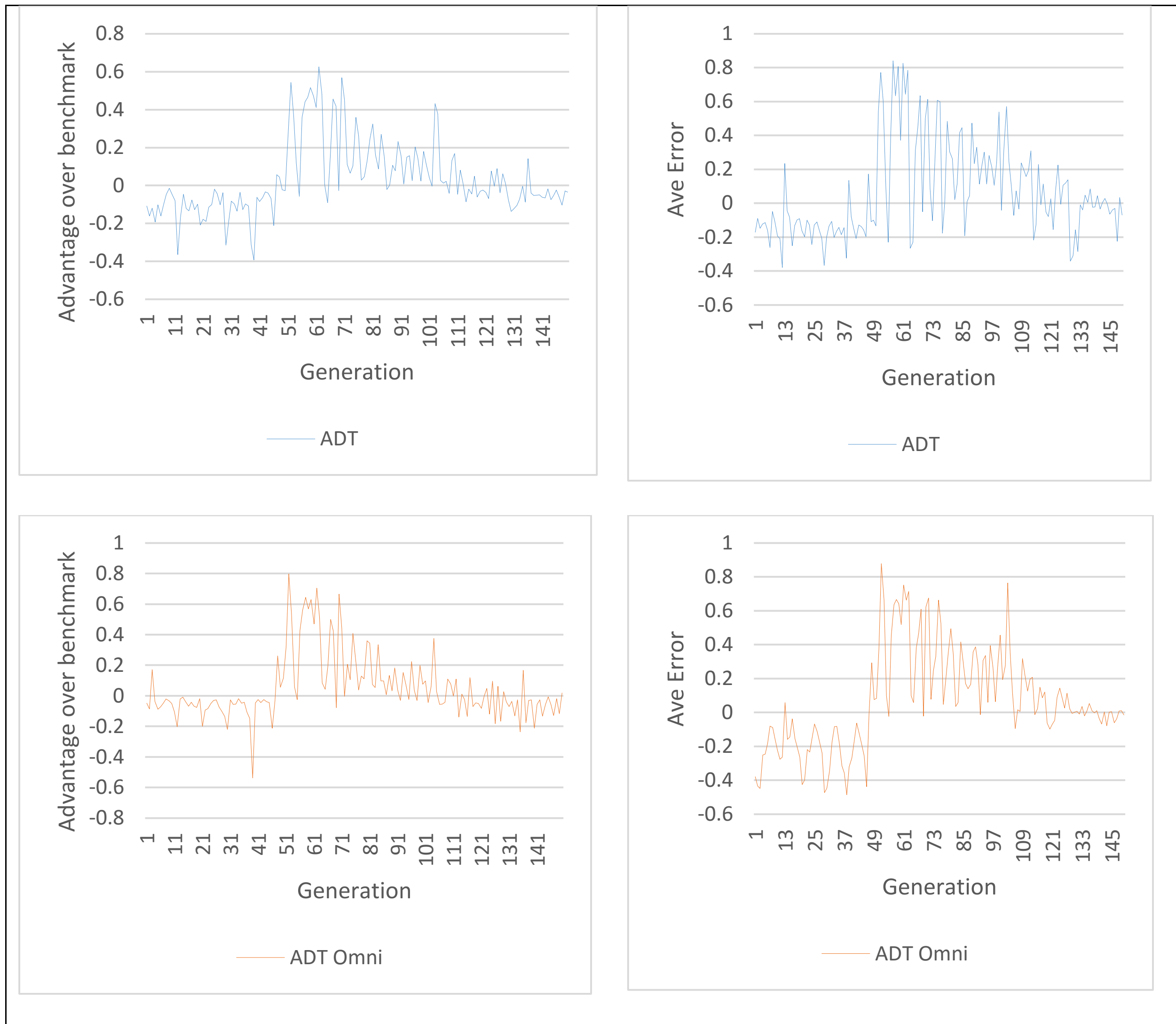



*Figure 40.* Relative performance of ADT vs. DyFor GP in LGOZLG experiment. Values below 0 indicate the approach performed worse that the DyFor GP benchmark.

Total fitness evaluations, shown in Table 9, are similar to what was seen in the prior experiment. As was already shown, ADT and AAT using the coupled approach perform approximately the same number of fitness calculations as the canonical approach, while using the decoupled approach almost doubles this number. While not as high as the decoupled approach, node evaluations in AAT and ADT are much higher than the canonical, due to the additional genetic material, and therefore larger average program size, available in multiple regime specific implementation branches.

Table 9. LGOZLG Total Evaluations

| Series | Fitness Evaluations | Fitness Calculations | Node Evaluations |
|---|---|---|---|
| GP | 3819 | 414,209 | 24,111,099 |
| DyFor GP | 3720 | 336,454 | 17,472,354 |
| | | Decoupled | |
| AAT Omni | 3609 | 390,830 | 26,032,503 |
| AAT | 6670 | 723,470 | 43,689,199 |
| ADT OMNI | 3647 | 395,994 | 26,320,581 |
| ADT | 6921 | 751,886 | 49,671,275 |
| | | Coupled | |
| AAT Omni | 3332 | 361,315 | 13,981,495 |
| AAT | 3363 | 364,968 | 17,857,202 |
| ADT OMNI | 3317 | 360,156 | 21,613,898 |
| ADT | 3558 | 386,384 | 35,425,943 |

*Note*. Each run includes 191 training generations. In addition to fitness evaluations, this table also includes the actual number of nodes evaluated, a function of population size and program size. Omni approaches show a lower number of node evaluations compared to their non-Omni counterparts since regime evaluation can be accomplished in a single, indivisible function.

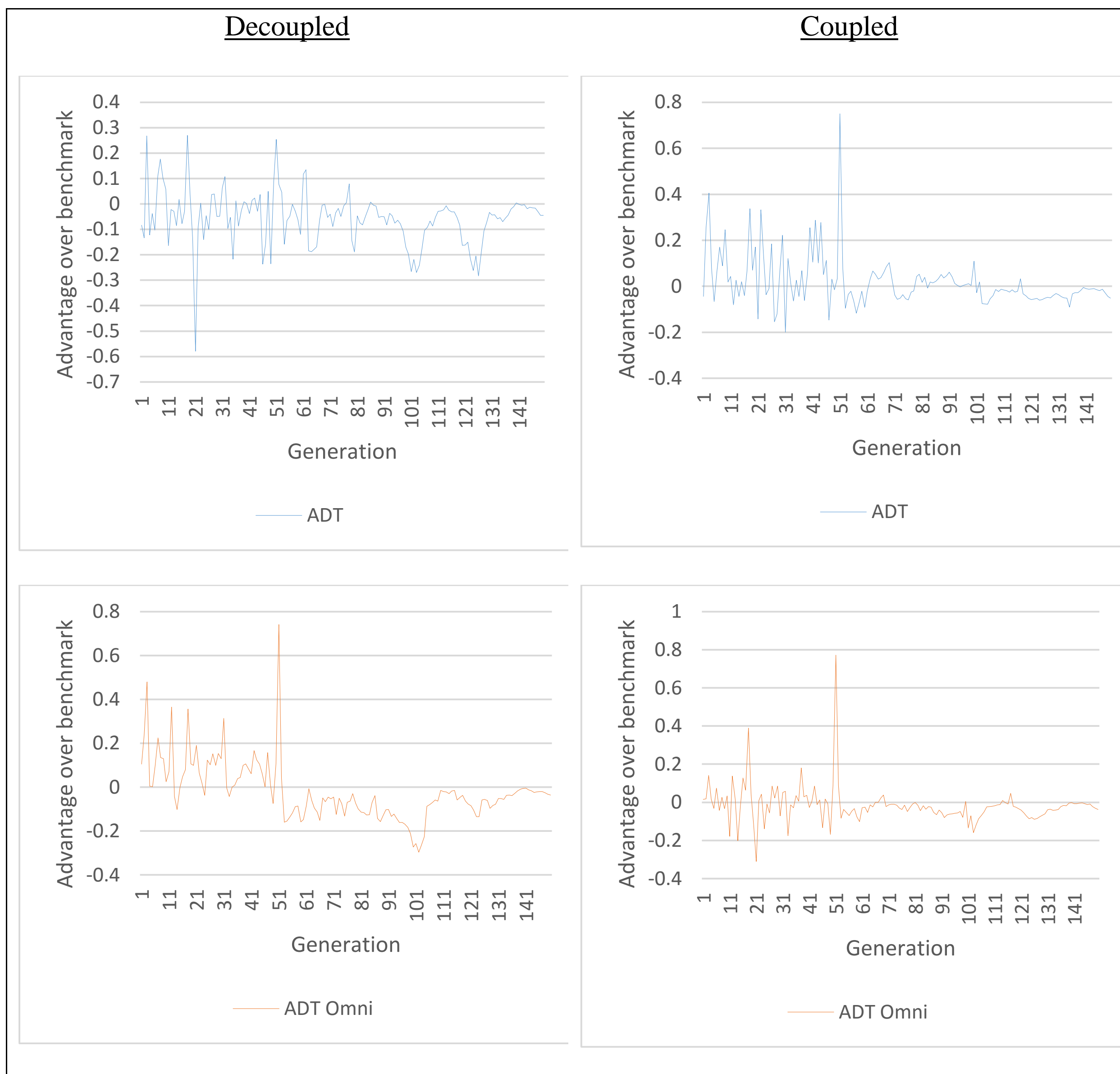
***MGHENMG.***

In the MGHENMG experiment, as was seen in the LGOZLG experiment, ADT outperformed AAT and the coupled approach outperformed the decoupled approach. The coupled version of ADT also outperformed DyFor GP to a 95% level of confidence, while the decoupled version did not. Omniscient regime detection runs did not generally produce the best results. Fitness calculations and node evaluations, shown in Table 11, are consistent with the results reported for LGOZLG.

Table 10. MGHENMG Prediction Results

| Series | Ave. MSE | Std. Dev. | 95% CI | Min. MSE | Random Walk Prediction[a] |
|---|---|---|---|---|---|
| DyFor GP | 0.096 | 0.042 | [0.078 - 0.114] | 0.049 | 0.442 |
| GP | 0.145 | 0.046 | [0.125 - 0.165] | 0.072 | 0.442 |
| | | | Decoupled | | |
| ADT Omni | 0.103 | 0.068 | [0.073 - 0.133] | 0.008 | 0.442 |
| ADT | 0.140 | 0.073 | [0.108 - 0.172] | 0.027 | 0.442 |
| AAT Omni | 0.184 | 0.053 | [0.161 - 0.207] | 0.079 | 0.442 |
| AAT | 0.196 | 0.128 | [0.140 - 0.252] | 0.076 | 0.442 |
| | | | Coupled | | |
| ADT | **0.056** | 0.020 | [0.048 - 0.065] | 0.031 | 0.442 |
| ADT Omni | 0.096 | 0.049 | [0.074 - 0.117] | 0.027 | 0.442 |
| AAT Omni | 0.182 | 0.056 | [0.157 - 0.206] | 0.050 | 0.442 |
| AAT | 0.186 | 0.070 | [0.155 - 0.216] | 0.087 | 0.442 |

*Note*. CI=Confidence Interval; Training done on points 150-250. Predictions done on points 251-400. [a]Assumes predicted next value equals current value. [b]Coupling approach does not impact DyFor GP or GP.


Decoupled
Coupled
Advantage over benchmark
Generation
ADT
ADT Omni

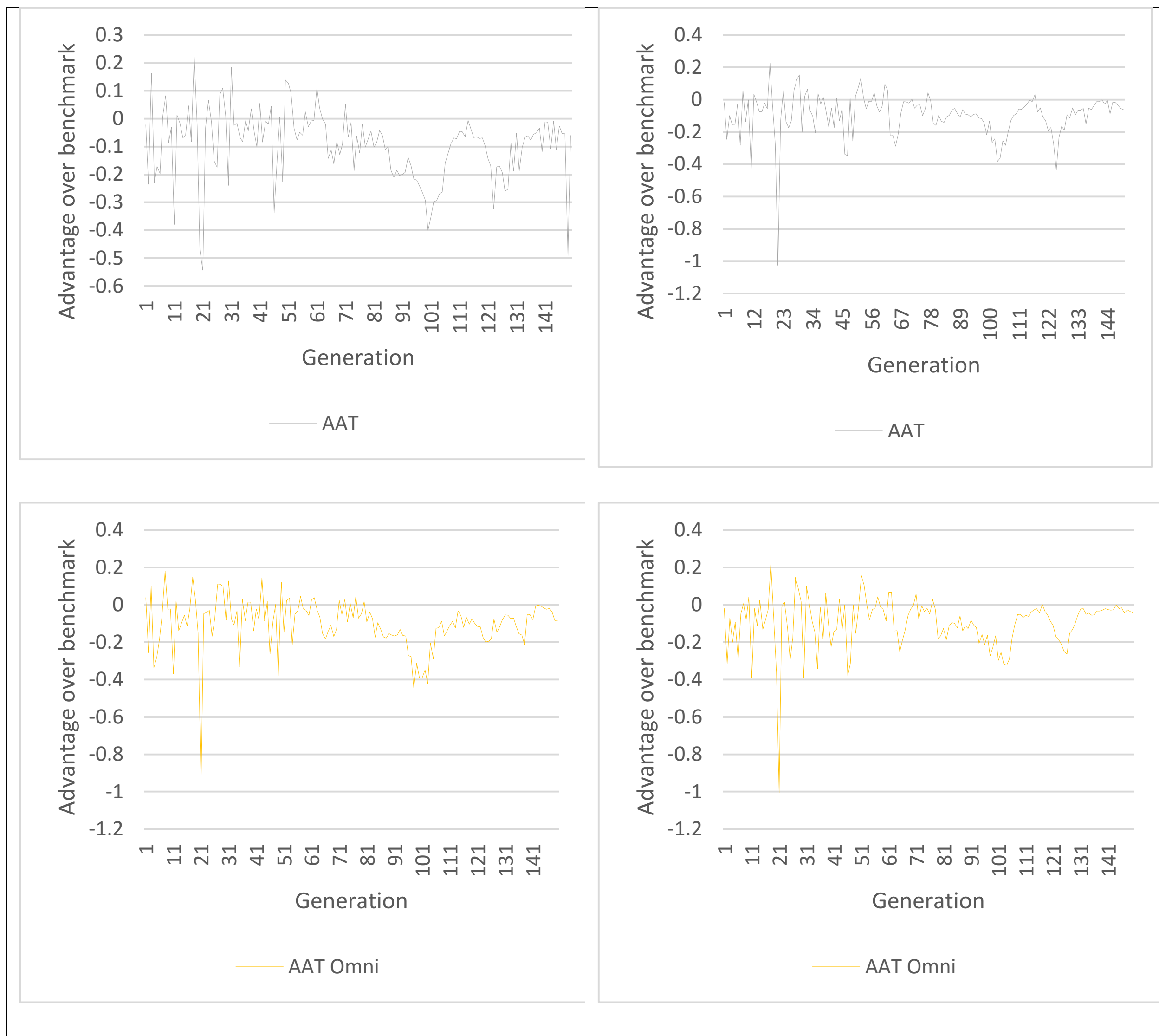



*Figure 41*. Relative performance of ADT vs. DyFor GP in MGHENMG experiment. Values below 0 indicate the approach performed worse that the DyFor GP benchmark.

Table 11. MGHENMG Total Evaluations

| Series | Fitness Evaluations | Fitness Calculations | Node Evaluations |
|---|---|---|---|
| GP | 3974 | 432,272 | 15,723,564 |
| DyFor GP | 3868 | 436,644 | 13,702,717 |
| | Decoupled | | |
| AAT | 6747 | 733,229 | 36,446,727 |
| AAT Omni | 3739 | 406,475 | 14,691,832 |

| Series | Fitness Evaluations | Fitness Calculations | Node Evaluations |
|---|---|---|---|
| ADT | 6921 | 751,637 | 34,826,588 |
| ADT Omni | 3607 | 391,585 | 24,071,967 |
| | | Coupled | |
| AAT | 3463 | 376,371 | 17,843,166 |
| AAT Omni | 3421 | 371,795 | 12,747,006 |
| ADT | 3543 | 384,301 | 39,598,888 |
| ADT Omni | 3311 | 359,648 | 17,977,841 |
| AAT | 3463 | 376,371 | 17,843,166 |

*Note*. Each run includes 191 training generations.

***Experiment validation.***

The LGOZLG and MGHENMG tests just described were trained on periods spanning two regimes, a change from the benchmark tests described in (Wagner & Michalewicz, 2008). The benchmark tests incorporated initial training using data points 1-100, a window including only one regime. To better validate these results, additional tests were run using data points 1-100 as training periods.[14] The coupled approach is used exclusively. Variations are run using training windows sizes of 80 and 110. These window sizes are chosen as they are the starting DyFor GP window size and the average of minimum and maximum DyFor GP window size, respectively.

As AAT proved to be inferior to ADT, that approach was not included in this round of validation testing. Instead, a new proposed technique using exponential moving average as part

[14] LGOZLG tests were run using values 1-100 for training. MGHENMG tests were run using values 31-130 as the training period as the values depend on up to 30 lagged values. It is not clear if the benchmark also adjusted the training range. It is unlikely the difference significantly affects the results either way.

of the fitness calculation is included. The Omni approaches are also not included, as they did not provide consistently better performance than evolutionary regime discovery. The results of this additional round of testing are shown in Table 12.

As expected, ADT performed worse, relative to DyFor GP, than in the prior set of experiments. However, incorporating the exponential moving average technique posted the best overall accuracy, though not for all EMA runs and not to a 95% confidence level. These results confirm the observation by Chen et al. (2008) that GP performance is highly dependent on the chosen parameters and data profile. Nevertheless, the overall performance of ADT compares favorably to DyFor GP.

Table 12. Results for LGOZLG and MGHENMG Validation Tests

| Series | Mean MSE | Std. Dev. | 95% CI | Min. MSE | Random walk prediction |
|---|---|---|---|---|---|
| | | | LGOZLG | | |
| ADT EMA 110 | 0.492 | 0.138 | [0.431 - 0.552] | 0.229 | 0.308 |
| DyFor GP | 0.500 | 0.127 | [0.444 - 0.555] | 0.309 | 0.308 |
| ADT EMA 80 | 0.520 | 0.218 | [0.424 - 0.615] | 0.203 | 0.308 |
| GP | 0.553 | 0.138 | ]0.493 - 0.613] | 0.297 | 0.308 |
| ADT 80 | 0.570 | 0.156 | [0.501 - 0.638] | 0.257 | 0.308 |
| ADT 110 | 0.668 | 0.272 | [0.549 - 0.787] | 0.380 | 0.308 |
| | | | MGHENMG | | |
| ADT EMA 80 | 0.232 | 0.086 | [0.194 - 0.269] | 0.099 | 0.503 |
| DyFor GP | 0.276 | 0.110 | [0.227 - 0.324] | 0.155 | 0.503 |
| ADT EMA 110 | 0.290 | 0.072 | [0.258 - 0.321] | 0.183 | 0.503 |
| ADT 80 | 0.294 | 0.089 | [0.255 - 0.333] | 0.143 | 0.503 |

| Series | Mean MSE | Std. Dev. | 95% CI | Min. MSE | Random walk prediction |
|---|---|---|---|---|---|
| GP | 0.337 | 0.209 | [0.245 - 0.428] | 0.130 | 0.503 |
| ADT 110 | 0.348 | 0.113 | [0.298 - 0.397] | 0.220 | 0.503 |

*Note*. EMA=Exponential Moving average fitness method; 80/100=training window size; 20 runs were performed for each series. An EMA multiplier of (2 / (Time periods+1)) is used which reduces the current weighing by 50% each time the period doubles.

*Figure 42.* Relative performance of ADT vs DyFor GP in LGOZLG validation tests. Values above 0 indicate ADT advantage.

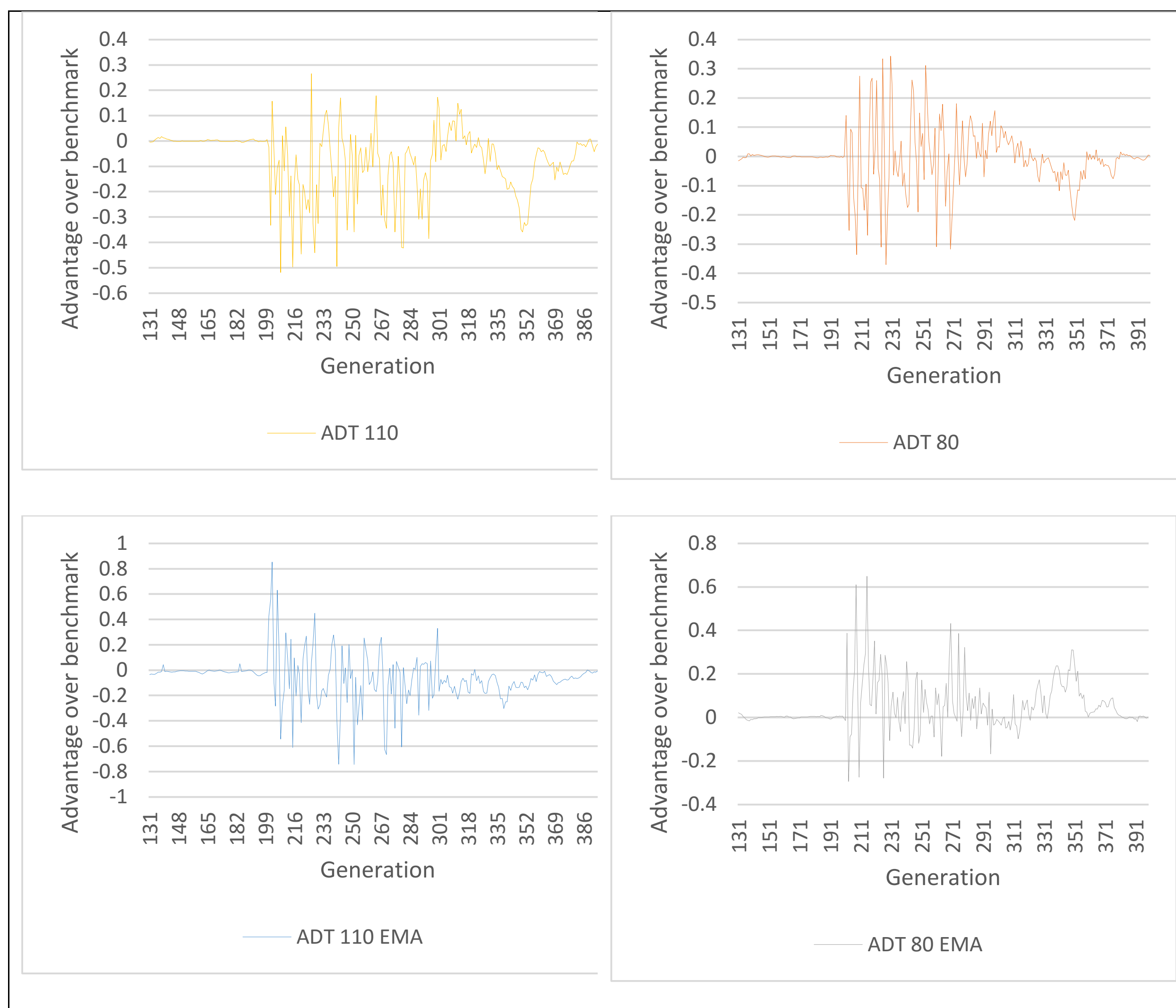



*Figure 43.* Relative performance of ADT vs. DyFor GP in MGHENMG validation experiment. Training was performed on data points 31-130. Values above 0 indicate ADT advantage.

The results achieved using DyFor GP where generally inferior to those reported in the benchmark study (Wagner & Michalewicz, 2008) and reproduced in Table 13. However, that study used a slightly different approach of including multiple dynamic generations for each window slide. The experiments done for this dissertation only included one training generation after each prediction. Additional training likely contributed to the superior results reported the benchmark study.

Multiple dynamic generations were not used in this study for several reasons. Firstly, the time needed to run such an experiment is great. The bloat control and diversity enhancement approaches in DyFor GP incorporate 50% new individuals every dynamic generation. The training phase serves a purpose similar to an initial training period. This period has been shown to consistently yield excellent results during initial training over the initial training window. These approaches are valuable and will be considered for future enhancements, as these combine the best features of both DyFor GP and ADT.

Table 13. Results Reported in Wagner & Michalewicz Benchmark Experiment

| Series | Runs | Mean MSE | Std. Dev. | 95% CI | Random walk prediction |
|---|---|---|---|---|---|
| | | LG-OZ-LG | | | |
| Standard GP (without update) | 20 | 0.9979 | 6.5540 | N/A | 0.308 |
| Standard GP (with update) | 20 | 0.3047 | 0.0334 | [0.290- 0.319] | 0.308 |
| DyFor GP | 20 | 0.2344 | 0.0567 | [0.210 - 0.259] | 0.308 |
| | | MG-HEN-MG | | | |
| Standard GP (without update) | 20 | 0.6039 | 0.4216 | [0.419 - 0.789] | 0.503 |
| Standard GP (with update) | 20 | 0.1960 | 0.0830 | [0.160 - 0.232] | 0.503 |
| DyFor GP | 20 | 0.1880 | 0.0278 | [0.176 - 0.200] | 0.503 |

*Note*. Taken from (Wagner & Michalewicz, 2008). CI and random walk values added to original data. A 70000 node global limit was used in this test. Update refers to additional training occurring in prediction phase. It is not clear if any bloat control methods were applied to standard GP approaches.

Table 14 shows the average program sizes recorded. The average result producing program size of DyFor and canonical GP are larger due to the lack ADFs. ADT generally divides nodes equally among the result producing and regime specific ADF branches. There being two

regime specific branches per ADT program, the overall node counts are proportionally higher in that approach.

Table 14. Average Population Node Counts

| Method | Avg. Nodes per Program | Avg. ADF Nodes per Program | Avg. Total Nodes |
|---|---|---|---|
| | | LGOZLG | |
| ADT 80 | 19.26 | 46.51 | 197,301 |
| ADT 110 | 19.15 | 48.32 | 202,407 |
| ADT 80 EMA | 21.98 | 46.43 | 205,222 |
| ADT EMA 110 | 22.03 | 50.46 | 217,481 |
| GP | 43.22 | N/A | 129,647 |
| DyFor GP | 44.53 | N/A | 133,600 |
| | | MGHENMG | |
| ADT 80 | 18.88 | 40.34 | 177,666 |
| ADT 110 | 21.46 | 42.83 | 192,879 |
| ADT 80 EMA | 18.38 | 44.33 | 188,150 |
| ADT EMA 110 | 18.99 | 43.75 | 188,197 |
| DyFor GP | 32.55 | N/A | 97,663 |
| GP | 41.55 | N/A | 124,662 |

*Note*. 80/100 in method titles indicate training window sizes; EMA indicates an exponential moving average approach was used in fitness calculations.

**Summary.**

Three sets of experiments using synthetic time series were performed. Two of these experiments tested chaotic series prediction and the third experiment tested symbolic regression on a nonlinear series. Two approaches to coordinating regime determining programs and result producing programs—coupled and decoupled—were also tested. ADT was the best performing method in all three sets of experiments when using a coupled approach. Decoupled ADT bested

all other methods in two of three tests, while DyFor GP performed better than decoupled ADT in one of three tests. AAT was the worst performing method using both a coupled and non-coupled approach. In two of three tests, ADT identified the known regime, while AAT did not identify the known regime in any test.[15]

A second set of experiments was performed to better simulate the exact parameters used in the benchmark experiment. A new technique for optimizing fitness calculation using an exponential moving average (EMA) was also introduced in this experiment. ADT with EMA posted the best results in this set of test, but the margin was less than in the prior test. This result was expected due to the differences in training periods in both tests. In the first test, ADT was trained across two regimes, while in the second test, ADT was trained in a period containing only one regime. EMA was shown to increase performance over non-EMA ADT.

## Market Data Series

### Data analysis.

This set of experiments looked exclusively at the S&P 500 index (S&P Dow Jones Indices LLC, 2016). The S&P 500 index series is inarguably the most popular benchmark comparison used in the financial industry. The experiments were modeled after the experiments presented in (Chen et al., 2008). That study examined the period from 1988 through 2004. Figure 44, Figure 45, and Figure 46 show the full index history, period of this study, and period of prediction, respectively.

[15] The determined regime was analyzed only in the best performing runs of ADT and AAT. There is no way to average out regime determination over all sets of runs.

AAT is not included in this set of experiments. Though it showed good results in some test, its performance was not consistent. AAT performed extremely poor in many of the synthetic tests, while ADT consistently performed well. As coupled ADT was the best performing out of all proposed methods, that variation was used exclusively in further experiments. Exponential moving average fitness calculation is not used as that technique is only relevant in prediction tasks when a known correct value can be determined at each point in the training window.

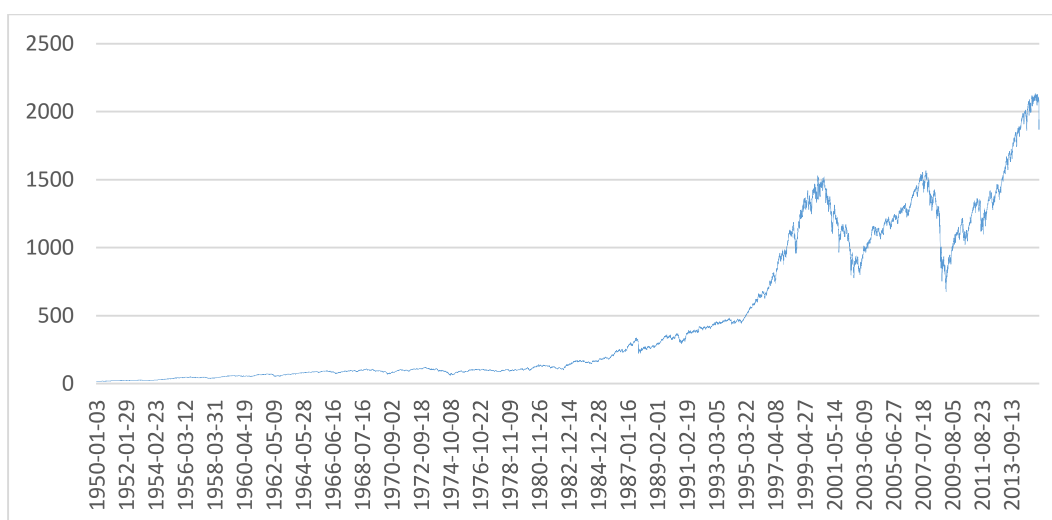



*Figure 44.* S&P 500 index.

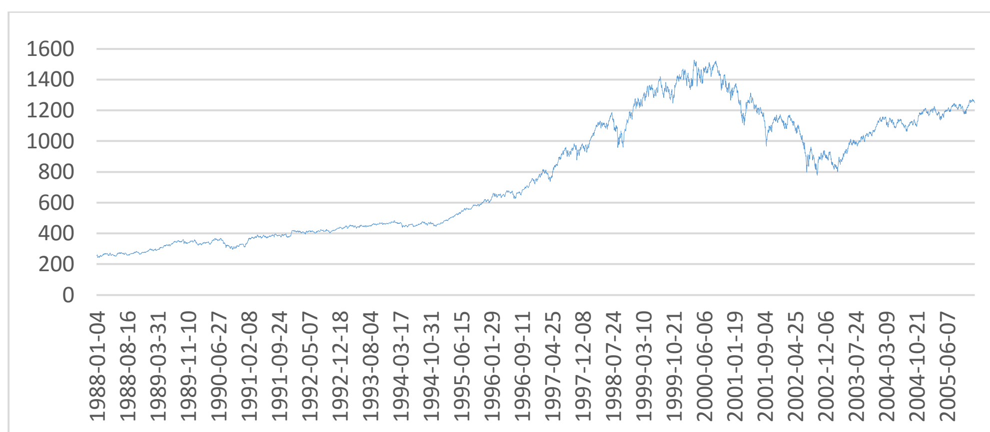



*Figure 45.* S&P 500 index during study period.

*Figure 46.* S&P 500 index during prediction period.

The benchmark experiment considers only the price history of the target S&P 500 series in its prediction. Volume, a data point often used in technical analysis, was not incorporated. Other possible influencing factors, such as unemployment, interest rates, commodity prices, etc., were not considered. Absolute prediction performance was not the ultimate aim of this experiment, but only a relative comparison between various prediction approaches using the same input parameters. Better predictions could likely be achieved by considering more input data, though optimal choice of input is not always obvious and is left as a future research question.

Predicting financial time series often requires the series be normalized to appear stationary. While this is a requirement in linear statistical methods, such as ARIMA, studies show that this can also provide better prediction results in genetic programming (Chen et al., 2008, pp. 140-141). In the benchmark study, Chen et al. normalized the series by dividing each value by its 250 day moving average. While other normalizations are often used, such as daily return or log-return, the market data experiments replicated the 250 day moving average normalization. The resultant normalized series is shown in Figure 47. This transformation will

center the resultant series around a value of 1. This normalization is also helpful in limiting the set of terminals that must be available, as most data points are of a similar magnitude.

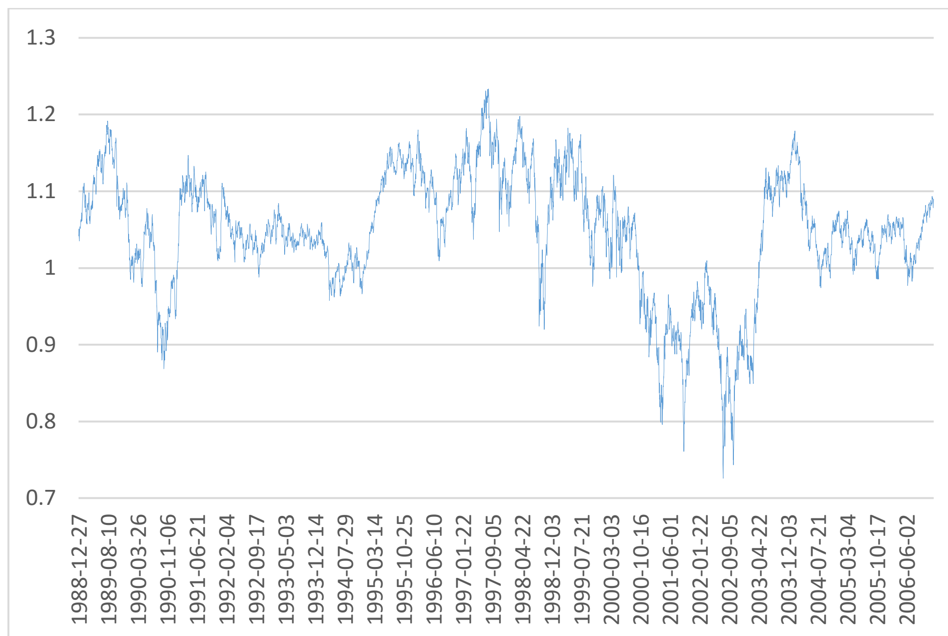



*Figure 47.* S&P 500 normalized series. Each value is divided by its 250 day moving average. The resulting series centers itself around the value 1.

***Experimental approach.***

In the prior experiments, evolution occurred for a fixed number of training generations. After training, predictions were made until the end of the time series was reached, with additional training occurring after each prediction. This set of experiments matches the approach taken by Chen et al. which differs from the prior experiment by the inclusion of a validation period. Immediately after each training generation, the best individual from that generation is selected and used to predict values in an out of sample period. The best performing individual over all validation generations is saved. That individual is then used to predict values in the prediction period. Evolution occurs only during training periods.

To facilitate and validate the new approach described above, the series is broken up into three overlapping segments. Prediction and training is performed on each segment independently. The segment ranges are shown in Figure 48.

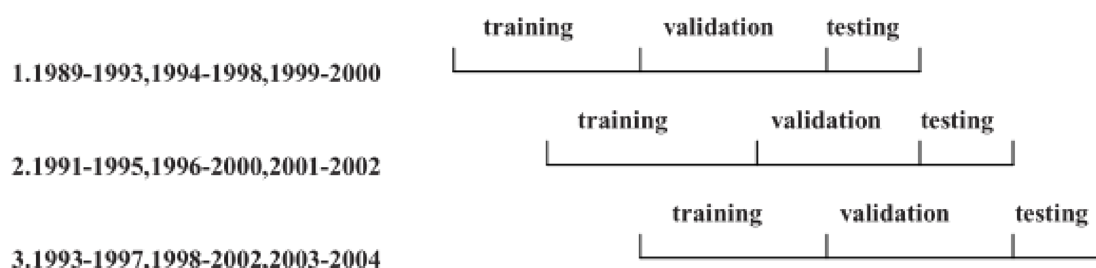



*Figure 48.* Overlapping training, validation, and testing periods used by Chen, Kuo & Hoi. The testing period corresponds to the prediction period discussed elsewhere in this dissertation. Taken from (Chen et al., 2008, p. 110).

As DyFor GP favors a dynamic sliding window approach with periodic retraining, this methodology was not tested with overlapping training periods. Instead, additional testing was done in a single run over the full series from 1989 through 2004. As ADT supports both dynamic and fixed prediction, that approach is included in the full series test. In this experiment, training was performed for years 1989 through 1998. Prediction occurred from 1999 through 2004, as was done by Chen et al. As this encompasses a much larger series, weekly training and predictions were done instead of daily. Also, similar to the synthetic studies, no validation period was incorporate but training was performed after each prediction step. This approach is the only possibility for DyFor GP due to it requirements for dynamic feedback.

***Fitness calculation.***

The fitness measurement in the market data experiment was the total percentage return achieved during the applicable time period. Returns are calculated by maintaining a running account and shares balance. At each investment decision point, the relevant program is run which

returns a Boolean value indicated the investment decision. When out of the market, the algorithm earns the prevailing T-bill rate for that period. The T-bill historical rate of return is shown in Figure 49.

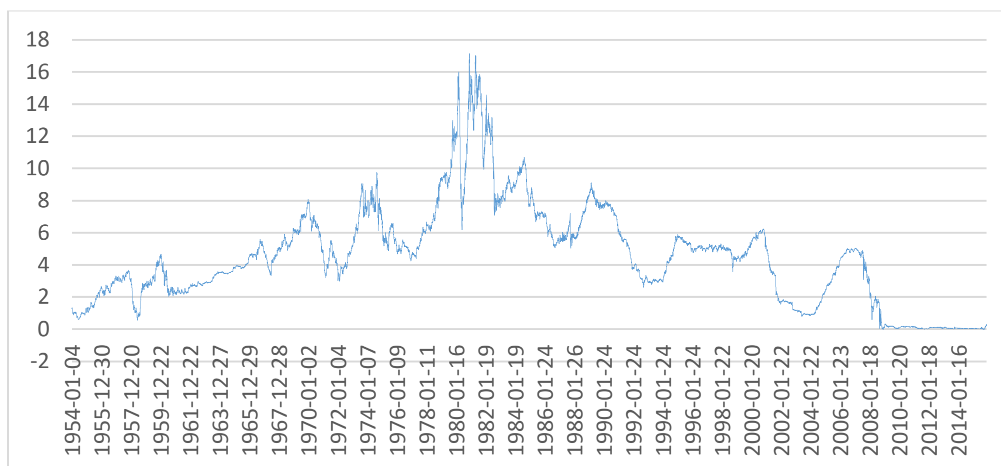



*Figure 49*. 3-month Treasury bill historical yield. Values are not limited to the study period. A larger period is shown for historical context.

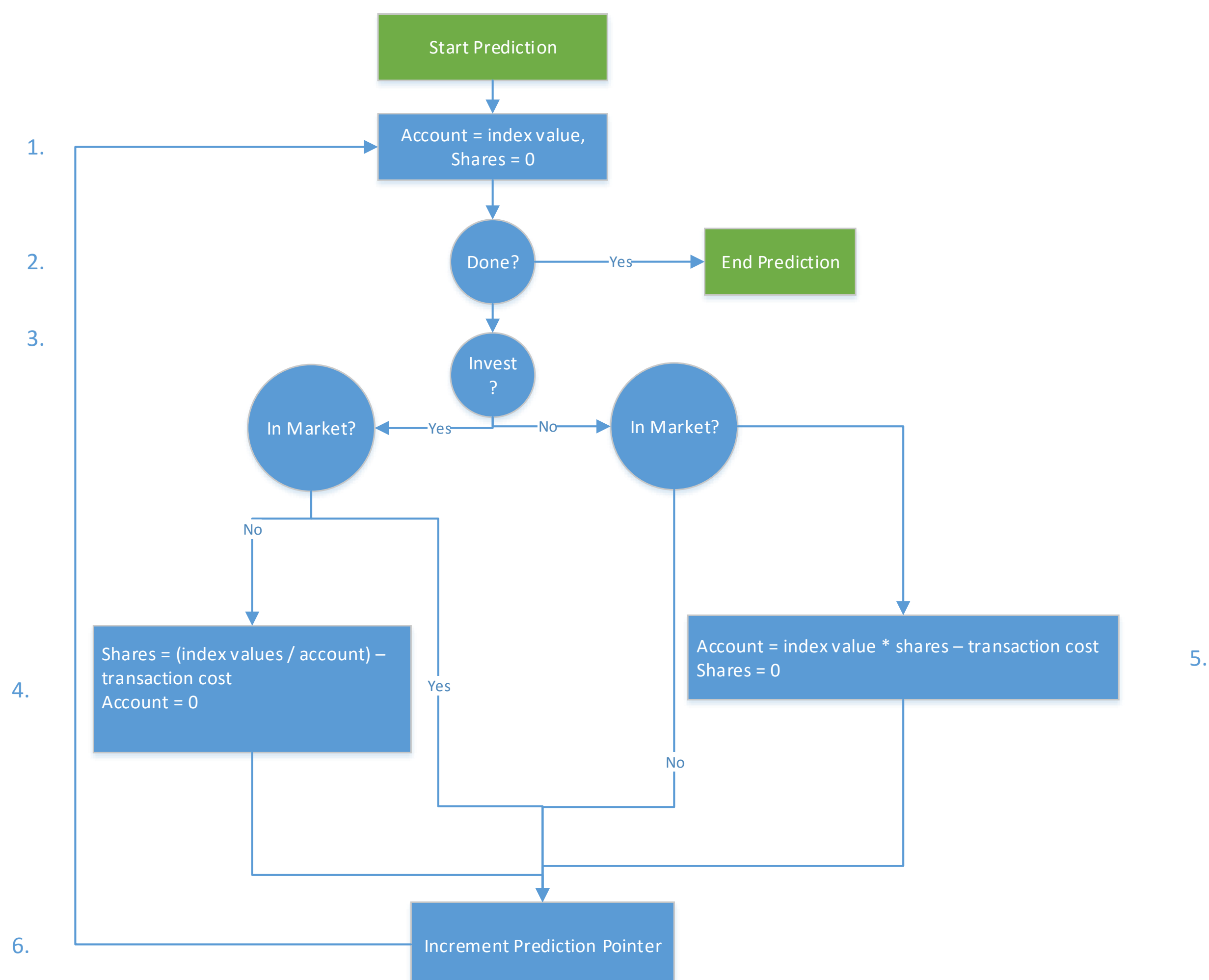



*Figure 50*. Total return calculation algorithm used in market data experiments.

This market prediction algorithm is show in and in Figure 50. The primary steps in the algorithm are as follows:

1. Initialize account balance to the index value at the period starting point. Initialize shares start to 0.
2. If the prediction window reaches the end of the prediction period, stop the process and return the final total return.
3. Run the investment program at the current prediction point. This will yield an invest/don’t invest decision.

4. If invest and already in market, do nothing. If invest and not currently in market, purchase shares at the current market (index) price using the full available account balance. Transactions cost, based on the current account balance, are applied prior to purchase of shares
5. If don't invest and not in the market, do nothing. If don't invest and in the market, sell all shares at the current market (index) price. Deduct a transaction cost and save the remainder as the current account balance.
6. Slide the prediction target forward and repeat the prediction phase.

***Experimental parameters.***

The parameters used in the market data experiment is shown in Table 15. This study attempted to reproduce the results of Chen et al. Several parameters used in that study are questionable. While mutation has become more popular in GP since Koza discounted it, a 40% mutation rate is unusually high. Also, the inclusion of a 0.5% transaction cost is probably unrealistically high, even in the study time period. Chen et al. noted this and they also ran experiments with transactions cost of 0% (ibid., p. 136). They noted that the results were marginally better and the complexity of the evolved program increased from an average node size of 19.9 to an average node size of 24.4 (ibid., p. 138). As this study is not an attempt to beat buy and hold, but to compare new modularity approaches, the study was done with the 0.5% transaction cost.

Table 15. Experimental Parameters Used in Market Prediction Tests

| Parameter | Value |
| --- | --- |
| Arity (ADF/ADT Only) | 2,2 |
| Crossover | 50% |
| Elitist | Single best individual |
| Function Set | add, subtract, multiply, divide, gt, lt, and, or, not, offsetValue, Boolean if-Else, Moving Average, Period Maximum, Period Minimum, norm |
| Initialization | Ramp-half-and-half |
| Max Depth | 10 |
| Max initial Depth | 5 |
| Max Node Size | 100 |
| Mutation | 40% |
| Population Size | 500 |
| Regimes (ADT Only) | 2 |
| Reproduction | 10%[a] |
| Risk Free Return | 3-month T-Bill rate |
| Stagnation tolerance | 50 |
| Terminal set | Random Integer (0 - 250), Random Double (0 -2), true, false, current series value[b] |
| Tournament size | 2 |
| Training Generations | 100 |
| Transaction Cost | 0.5% |

*Note*. Detailed descriptions of functions and terminals are provided in Appendix C.
[a]The referenced study shows a 9.8% reproduction rate. This is due to a slightly different approach to elitism. That study choses a percentage of elite instead of a single elite. As it turns out, .02% elite section equals exactly one individual out of a population size of 500. [b]The current value is achievable through the offset value function and an offset of 0. However, an additional discrete terminal representing the current price is added to favor this important value.

Table 16. DyFor GP Parameters Used in Market Data Experiments

| Parameter | Value |
|---|---|
| Max Window Size | 375 |
| Min Window Size | 125 |
| N | 3 |
| Save Off | 10 |
| Start Window Size | 250 |

**Findings.**

The results recorded in the market data experiment are displayed below. Data is separated by prediction period and transaction cost inclusion approach. Table 17 provides the performance for each of the three prediction periods with transaction costs and out of market interest taken into account. ADT was the best performing approach in three out of three tests. ADT also beat DyFor GP in the full period test. Only in the 2003-2004 prediction does ADT achieve a 95% confidence level in beating GP. All other comparisons have overlapping confidence intervals.

Similar to results reported by other researchers, for example (Allen & Karjalainen, 1999) and (Chen et al., 2008), no tested approach beat a buy and hold strategy on average when taking transaction costs into account.

When transaction costs are ignored, the results change significantly. Most interesting, all evolutionary approaches beat buy and hold on average when not considering transaction costs. All evolutionary approaches beat buy and hold on average in all but one period; only ADF performed better than buy and hold in 2003-2004.

Table 17. Market Data Experiment Results Including Transaction Costs

| Method | Mean | Std. Dev. | Min | Max | 95% CI | # beating benchmark |
|---|---|---|---|---|---|---|
| | | | 1999-2000 | | | |
| Buy & Hold | 0.0751 | | | | | |
| GP | 0.0434 | 0.0664 | -0.1917 | 0.1197 | [0.0250 ... 0.0618] | 5/50 |
| ADF | 0.0309 | 0.0798 | -0.3054 | 0.0845 | [0.0088 ... 0.0530] | 3/50 |
| ADT | 0.0510 | 0.0519 | -0.1974 | 0.1042 | [0.0366 ... 0.0654] | 5/50 |
| | | | 2001-2002 | | | |
| Buy & Hold | -0.3144 | | | | | |
| GP | -0.3693 | 0.1306 | -0.8087 | -0.2885 | [-0.4055 ... -0.3331] | 1/50 |
| ADF | -0.3347 | 0.0887 | -0.7290 | -0.1777 | [-0.3593 ... -0.3102] | 2/50 |
| ADT | -0.3697 | 0.1390 | -0.7450 | -0.0134 | [-0.4082 ... -0.3312] | 1/50 |
| | | | 2003-2004 | | | |
| Buy & Hold | 0.3332 | | | | | |
| GP | 0.2945 | 0.0497 | 0.1432 | 0.3291 | [0.2807 ... 0.3083] | 0/50 |
| ADF | 0.3139 | 0.0390 | 0.1170 | 0.3539 | [0.3031 ... 0.3247] | 1/50 |
| ADT | 0.3247 | 0.0150 | 0.2349 | 0.3522 | [0.3205 ... 0.3289] | 2/50 |

*Note*. Transaction costs and out of market risk free return are included in this set of experiments.

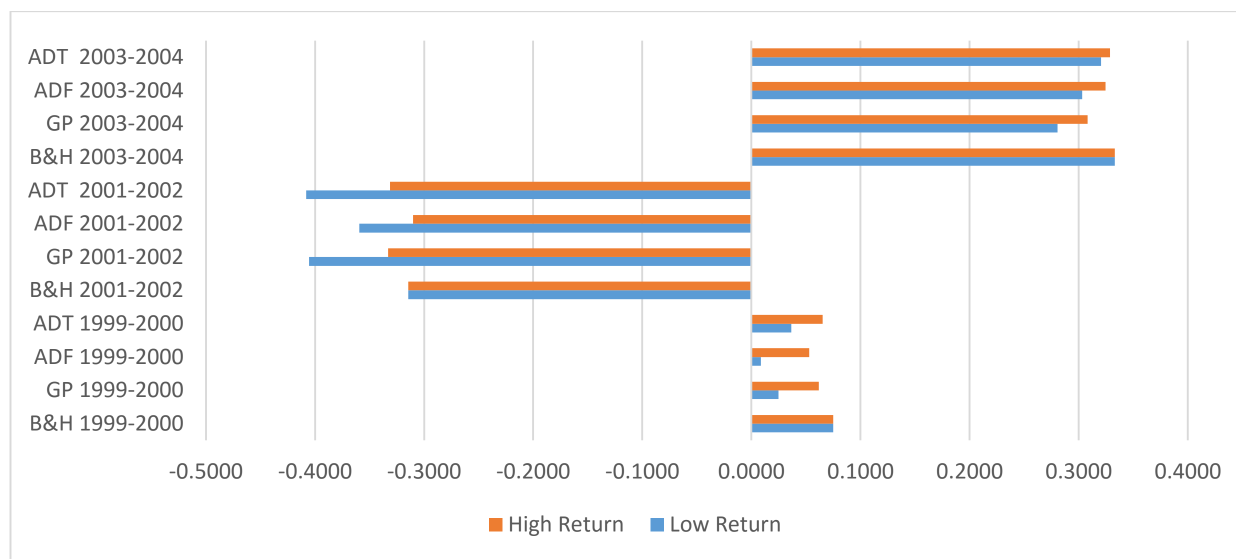



*Figure 51.* 95% confidence intervals for market data experiments.

Table 18. Sliding Window Market Data Experiment Results With Transaction Costs

| Method | Mean | Std. Dev. | Min | Max | 95% CI | # beating benchmark |
|---|---|---|---|---|---|---|
| | | | 1999-2000 | | | |
| Buy & Hold | 0.0634 | | | | | |
| ADT | 0.0018 | 0.1372 | -0.4290 | 0.2388 | [-0.0362 ... 0.1010] | 17/50 |
| DyFor GP | -0.0157 | 0.1101 | -0.2690 | 0.1426 | [-0.0463 ... 0.0639] | 15/50 |
| | | | 2001-2002 | | | |
| Buy & Hold | -0.3339 | | | | | |
| ADT | **-0.1364** | 0.1014 | -0.2964 | 0.1601 | [-0.1645 ... -0.0631] | 50/50 |
| DyFor GP | **-0.1018** | 0.0819 | -0.2810 | 0.0817 | [-0.1245 ... -0.0426] | 50/50 |
| | | | 2003-2004 | | | |
| Buy & Hold | 0.2970 | | | | | |
| ADT | 0.1035 | 0.0653 | -0.0603 | 0.2529 | [0.0854 ... 0.1507] | 0/50 |
| DyFor GP | 0.0489 | 0.0723 | -0.1780 | 0.2156 | [0.0289 ... 0.1012] | 0/50 |

| Method | Mean | Std. Dev. | Min | Max | 95% CI | # beating benchmark |
|---|---|---|---|---|---|---|
| | | | 1999-2004 | | | |
| Buy & Hold | -0.0189 | | | | | |
| ADT | -0.0349 | 0.1933 | -0.5395 | 0.4592 | [-0.0884 ... 0.0187] | 24/50 |
| DyFor GP | -0.0698 | 0.1413 | -0.3597 | 0.2136 | [-0.1089 ... -0.0306] | 15/50 |

*Note.* A single run was done over the period 1989-2004. Training was performed from 1989 through 1998. Prediction occurred from 1999-2004. Retraining occurred after each prediction. Mean returns shown in bold beat the buy and hold benchmark. Transaction costs and out of market risk free returns are incorporated.

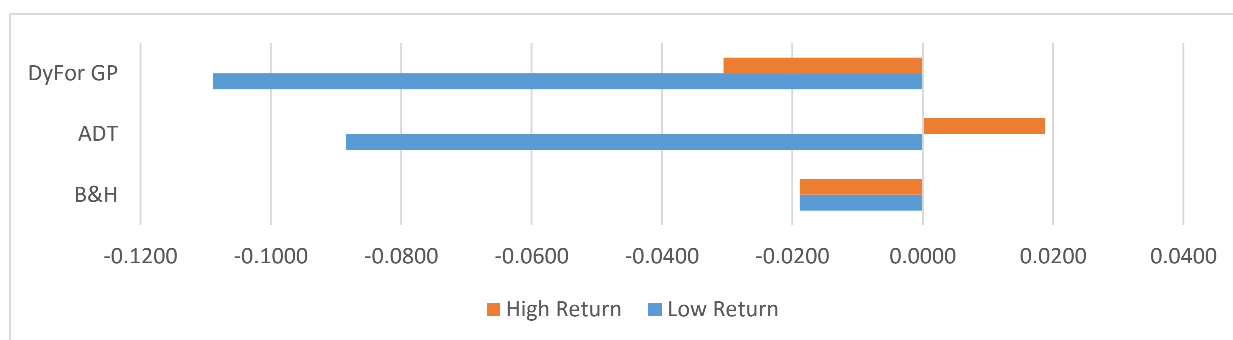



*Figure 52.* 95% confidence intervals for sliding window market data experiments.

The results of the benchmark study from (Chen et al., 2008) is shown in Table 19 for comparison. These results appear slightly better than those reported in Table 17 for the 1999-2000 and 2001-2002, and are comparable in the 2003-2004 period. This discrepancy may be due to minor differences in period boundaries, return calculation, or unpublished parameter differences.

Table 19. Findings Reported by Chen, Kuo, & Hoi

| Period | Mean | Std. Dev. | Max | Min | B&H |
|---|---|---|---|---|---|
| 1999-2000 | 0.0655 | 0.0342 | 0.1171 | -0.1294 | 0.0644 |
| 2001-2002 | -0.3171 | 0.0498 | -0.0461 | -0.3486 | -0.3228 |
| 2003-2004 | 0.3065 | 0.0334 | 0.3199 | 0.1173 | 0.3199 |

*Note.* Taken from (Chen et al., 2008, p. 112) . The referenced study included 7 additional foreign market indexes. The B&H return is slightly different from that calculated in this study which may be due to differences in exact start and end dates used for each overlapping period.

Table 20 and Table 21 show the results when transaction costs are not considered. As may be expected, results are considerable better, with evolutionary algorithms besting buy and hold on average in nine of eleven tests.

Table 20. Market Data Experiment Results Not Including Transaction Costs

| Method | Mean | Std. Dev. | Min | Max | 95% CI | # beating benchmark |
|---|---|---|---|---|---|---|
| | | | 1999-2000 | | | |
| Buy & Hold | 0.0751 | | | | | |
| GP | **0.1494** | 0.1088 | -0.0438 | 0.4525 | [0.1192 ... 0.1795] | 35/50 |
| ADF | **0.1418** | 0.1238 | -0.0399 | 0.5112 | [0.1075 ... 0.1761] | 35/50 |
| ADT | **0.1567** | 0.1099 | -0.0068 | 0.4796 | [0.1262 ... 0.1871] | 37/50 |
| | | | 2001-2002 | | | |
| Buy & Hold | -0.3144 | | | | | |
| GP | **-0.3121** | 0.0573 | -0.4081 | -0.0348 | [-0.3280 ... -0.2962] | 17/50 |
| ADF | **-0.3023** | 0.0848 | -0.5153 | 0.0196 | [-0.3258 ... -0.2788] | 18/50 |
| ADT | **-0.2843** | 0.0635 | -0.3924 | -0.1245 | [-0.3020 ... -0.2667] | 32/50 |
| | | | 2003-2004 | | | |
| Buy & Hold | 0.3332 | | | | | |
| GP | 0.3045 | 0.0929 | 0.0463 | 0.5045 | [0.2788 ... 0.3303] | 15/50 |
| ADF | **0.3395** | 0.1171 | -0.0016 | 0.5597 | [0.3070 ... 0.3719] | 22/50 |
| ADT | 0.3329 | 0.1202 | 0.0775 | 0.6443 | [0.2996 ... 0.3663] | 29/50 |

*Note*. Mean returns shown in bold beat the buy and hold benchmark.

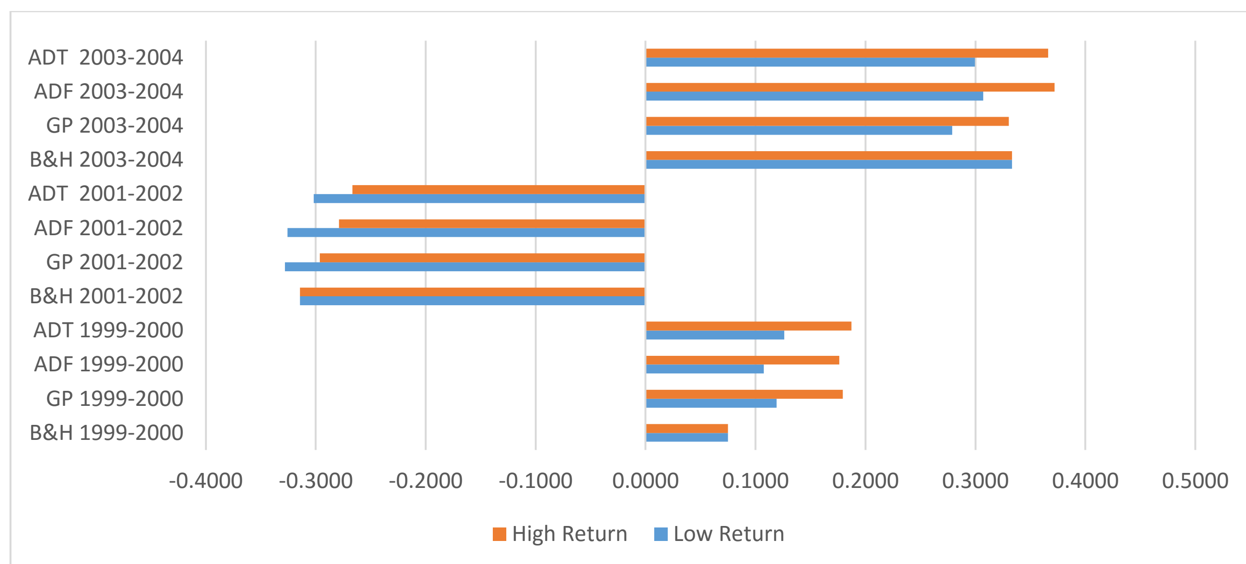



*Figure 53.* 95% confidence intervals for market data experiments not including transaction costs.

Table 21. Sliding Window Market Data Experiment Results Not Including Transaction Costs

| Method | Mean | Std. Dev. | Min | Max | 95% CI | # beating benchmark |
|---|---|---|---|---|---|---|
| | | | 1999-2000 | | | |
| Buy & Hold | 0.0634 | | | | | |
| ADT | **0.0788** | 0.1071 | -0.1106 | 0.3576 | [0.0491 ... 0.1562] | 27/50 |
| DyFor GP | **0.0807** | 0.1323 | -0.1904 | 0.3408 | [0.0440 ... 0.1763] | 26/50 |
| | | | 2001-2002 | | | |
| Buy & Hold | -0.3339 | | | | | |
| ADT | **-0.0524** | 0.1026 | -0.2674 | 0.1521 | [-0.0808 ... 0.0218] | 50/50 |
| DyFor GP | **-0.0594** | 0.0862 | -0.2314 | 0.1020 | [-0.0833 ... 0.0029] | 50/50 |
| | | | 2003-2004 | | | |
| Buy & Hold | 0.2970 | | | | | |
| ADT | 0.1246 | 0.0782 | -0.0132 | 0.3739 | [0.1029 ... 0.1811] | 2/50 |
| DyFor GP | 0.1233 | 0.0702 | -0.0297 | 0.2783 | [0.1038 ... 0.1740] | 0/50 |
| | | | 1999-2004 | | | |
| Buy & Hold | -0.0189 | | | | | |
| ADT | **0.1683** | 0.2005 | -0.1946 | 0.6959 | [0.1128 ... 0.2239] | 39/50 |
| DyFor GP | **0.1568** | 0.1887 | -0.2618 | 0.5762 | [0.1045 ... 0.2091] | 41/50 |

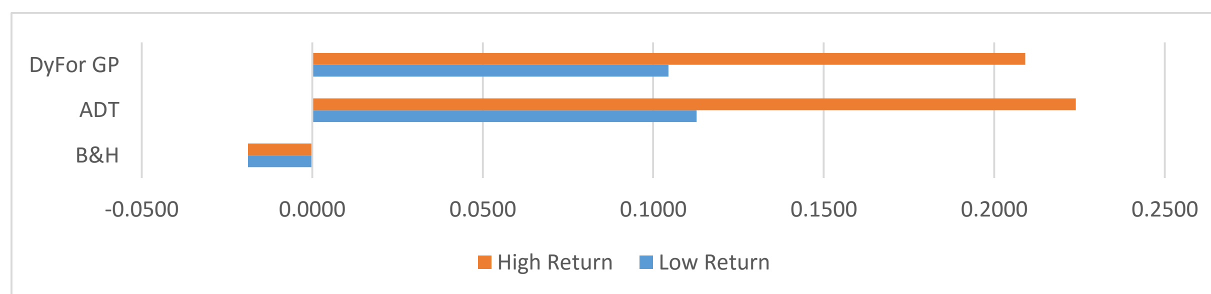



*Figure 54.* 95% confidence intervals for sliding window market data experiments not including transaction costs.

**Summary.**

Three tests were performed on three overlapping training and investment periods using the S&P 500 index as a target. Transaction costs and out of market risk free return were incorporated. ADT was the best performing approach in two of three tests. Neither ADT nor any other approach beat a buy and hold strategy in any of the testing periods.

An additional test was run over the entire analysis period utilizing a sliding window approach with additional training after each prediction. DyFor GP was only included in this test. ADT performed better than DyFor GP in this experiment, but did not beat a buy and hold strategy.

An additional set of tests were run that ignored transaction costs and out of market returns. In these tests, ADT performed best in two of three periods and beat buy and hold in two of three periods. In a full period, sliding window test, ADT performed better than DyFor GP and both approaches beat a buy and hold strategy. ADT and DyFor GP returned 16.8% and 15.7% respectively, for the periods 1999-2004 compared to a return of -1.89% achieved using a buy and hold strategy. The full window tests with retraining yielded better results than approaches without retraining, indicating that periodic retraining is necessary for dynamically changing data such as real world financial time series.

While identification of regimes in the S&P 500 index history over the examined time period is subject to interpretation, ADT did not appear to indicate any distinct regime. This, however, did not affect its performance as it still achieved the best results overall.

**Summary of Results**

The experimental results reported in the prior section show that coupled ADT was the overall best approach tested. AAT and, to a lesser extent, decoupled ADT were poor performers by comparison. Regime discovery showed mixed benefits, only occasionally uncovering the known regime in several synthetic tests, while providing no interpretable regime indication in the market data tests.

Coupled ADT was generally the best performer in the market data tests. Regime indicators did not appear to pick out potential regimes. Instead, extra available genetic material was used to further model the underlying data generation process in ways perhaps superior to simple regime determination and template implementation. This observation is further supported by the comparable performance achieved when omniscient regime detection was incorporated as when regimes were determined through evolutionary means.

# Chapter 5
# Conclusions, Implications, Recommendations, and Summary

This chapter reviews the work presented in this dissertation and further evaluates and summarized the results obtained. Several promising areas for future research also mentioned.

## Conclusions

The goal of this dissertation was to improve the performance of time series prediction using genetic programming, especially in the presence of regime change. This goal was achieved by the introduction of new modularity techniques that enabled the evolution of regime specific functionality. A regime indictor branch was incorporated into the GP algorithm to control the selection of regime specific implementations incorporated into the function definition branch.

Two techniques were proposed and illustrated: Automatically defined templates (ADT) and automatically acquired templates (AAT). These two approaches were compared against other common genetic programming paradigms, as well as against DyFor GP, the only other genetic programming approach found to consider regime change. Variations on how these two approaches couple program branches were also tested. Both approaches to ADT tightly couple the function and result producing branches, similar to coupling in ADF. Decoupled ADT allows the regime determining branch to evolve separately from the result producing branch. AAT only allows for the coupling between regime determining and result producing branches, as functions are shared across the entire program population and therefore remain independent from any individual result producing program.

ADT was the best performing approach in the majority of experiments performed and generally achieved superior performance to DyFor GP. Compared to DyFor GP, ADT was shown to be a more flexible approach, applicable to many prediction and regression scenarios,

while DyFor GP was found applicable only to sliding window prediction problems. ADT and AAT were more dependent on training conditions and performed best when initial training occurred across regime boundaries.

Decoupled ADT generally performed well, but not as good as coupled ADT. One potential problem with a decoupled regime determining branch is that evolved programs may number regimes differently. Result producing programs may therefore see the same actual regime numbered differently and therefore use a different implementation when a template method is called. While the overall fitness function is common to the entire population, each individual evolves separately along different paths. Tight coupling between program branches allows evolution to proceed in tandem towards a common implementation optimizing the governing fitness function.

ADT was shown to be superior to AAT due to ADT's permanent coupling between functions and result producing programs. AAT does not implement such coupling, limiting its performance. Kinnear (1994) discussed the inferior performance of module acquisition (the basis for AAT) relative to automatically defined functions (the basis for ADT). He proposed that this deficiency was due to ADFs relatively high structural regularity, described as the frequency of ADF calls and multiple uses of the same parameters. Structural regularity was seen to be lower in MA, due to the extracted modules immunity from any further modification such as crossover. Restricting evolution essentially decouples modules from the programs using them, as the former can no longer evolve based on the performance of the latter. A similar situation was encountered in the decoupled approaches tested in this dissertation.

The market data experiments confirmed what has been reported in prior studies ([Allen & Karjalainen, 1999], [Chen et al., 2008][16])—evolutionary approaches to market prediction have difficulty beating buy and hold when transaction costs are factored in. Taking transaction cost into account, no approach beat buy and hold on average in the tests performed. However, when transaction costs were ignored, evolutionary approaches beat buy and hold in the majority of cases.

Financial time series are largely stochastic. Even though GP may do well at modeling the underlying data generating process, those results may only be a close approximation to reality and may stop working abruptly at any time. GP prediction is also highly dependent on the parameter selection and data profile. Financial time series are also impacted by random events as well as investor psychology, as market participants continually adjust their evaluation regarding market conditions and directions. Luckily, genetic programming has an advantage in this respect due to its ability for automatic retraining over time.

**Implications**

This research contributed to the body of work on genetic programming by introducing and demonstrating several new modularity techniques that were shown to improve genetic programming performance compared to other approaches. This study was perhaps the first attempt at implementing software engineering techniques in genetic programming beyond the incorporation of language constructs seen in prior research.

---

[16] Chen et al. did find that GP performed consistently better in predicting the Taiwan market during the period of study, but was not consistently better for the other six markets analyzed, including the S&P 500 index.

This dissertation also has implications for market data prediction and financial investment. Many previous studies have shown that various technical indicators fail to beat buy and hold, when transaction cost are taken into account (Allen & Karjalainen, 1999); A result was confirmed in this dissertation. However, when transaction costs were not considered, the automatic approaches tested performed significantly better than a buy and hold approach. With the ever decreasing transaction costs, effectively zero in some cases, automated market prediction algorithms are now a more justifiable alternative to buy and hold or other manual trading approaches—if investors can achieve such low costs.

## Recommendations for Future Work

This section includes several promising areas for future research and extension of the methodology presented and implemented in this dissertation.

### Adaptive training and dynamic training generations.

As the underlying data generation process changes over time, existing or saved solutions may no longer correctly model the time series under analysis. Population fitness may degrade drastically immediately following a regime change. DyFor GP attempts to handle this scenario through a dynamically sizing analysis window based on prediction differentials. Another alternative approach simply to allocate additional training generations if prediction accuracy worsens. Currently, the number of training generations is a configurable ADT parameter. A simple extension would dynamically choose this parameter based on prediction performance and any defined lower and upper bounds.

A variation on adaptive training, dynamic training generations was proposed by Wagner & Michalewicz (2008) for use in DyFor GP. This feature was not considered for use in ADT or DyFor GP in the experiments performed in this dissertation, as it does not specifically concern

regime handling. In the dynamic generations approach, half of the population introduced and propagated to the next generation after each prediction are newly initialized individuals, while half are selected from the best performing individuals of the prior generations. Additional training, similar to though more extensive than what was proposed for adaptive training, is therefore needed on the newly introduced individuals. While increasing computation cost, this approach was shown to improve performance (ibid.) and should be considered for incorporation into ADT.

**Optimization of parameters.**

Many researchers have noted the susceptibility of evolutionary algorithms to the choice of initial parameters and the susceptibility of prediction algorithms to the choice of analysis periods. This research generally chose parameters to best replicate prior benchmark experiments. Further modification (ex. lower program size limits) might lead to more interpretable results, especially in the market data tests. However, care must be taken so that results do not converge to trivial solutions too early, as better models of the underlying data generation process might take longer to uncover with a smaller program size. A balance must be found between overfitting of data and generality of solutions.

**Computational optimization.**

The approach presented used native Java to simulate an s-expression tree evaluation after a native Clojure implementation was found to perform poorly. The Java approach is still computationally expensive, as each program tree is fully evaluating at every evaluation point in a training window. Limiting tree size helps, but computational overhead still limited the amount of evaluation runs that could easily be achieved.

It has been noted that evolutionary representations, similar to what is seen in their genetic inspiration, DNA, develop introns (Angeline, 1993, p. 10)—sections of code that do not directly contribute to the fitness of an individual. For example, Figure 55 shows a tree representation of the equation $y + 2x - 2x$ which always equals $y$. The second two terms, represented by the subtraction node in the figure, can be effectively ignored during fitness evaluation, potentially saving computational resources at each step.

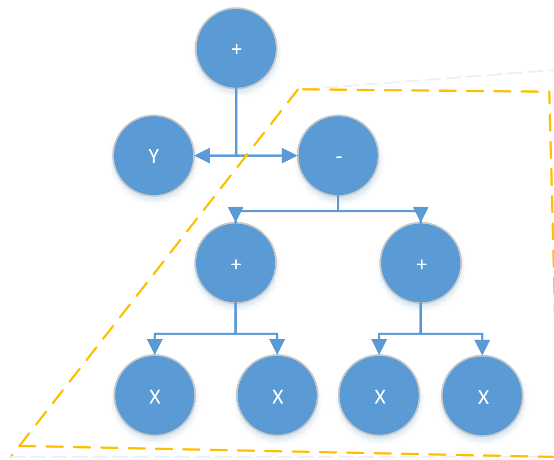



*Figure 55.* Program tree containing an intron. The expression represents the equation $y + 2x - 2x$. The outlined subtree has no impact on the expression result, which is always $y$.

It is probably not optimal, to search for introns prior to fitness evaluation; potential introns could, however, be noted during initial fitness evaluation and ignored during later stages. Even if intron detection is occasionally wrong, this technique could still contribute to improved performance as evolutionary algorithms are largely stochastic and approximate, and benefit from even minor improvements.

GP is highly parallel by nature. Incorporating parallelism into the GP framework, through concurrent fitness evaluation and genetic operations, would further improve performance. This enhancement requires suitable multicore, large memory systems to take advantage of parallel processing. More recent commodity distributed MapReduce/Hadoop frameworks, such as

Hadoop, make porting application code to a distributed computational network an obvious next step where performance is needed. A GP system running on Hadoop is described in is described in (Verma, Llorà, Goldberg, & Campbell, 2009).

**Improving market data prediction.**

While this dissertation did not have the primary goal of demonstrating market beating investment performance, the approach was shown to be valuable in task and further development could be tailored towards this objective. Two potential improvements are described below.

***Additional predictor series.***

This study limited predictor series to the target series itself. There is no reason why the vast volume of available financial data cannot also be used in prediction. The main problem when considering additional data is deciding which predictors are relevant. A secondary concern is any required cleansing and formatting of new data sources. The methodology developed in this dissertation can effectively handle any number of predictor series defined as input parameter. However, it is likely that simply making tens, hundreds, of even thousands of additional series available will negatively impact solution convergence and overwhelm the computation resources available. Clearly, an intelligent way to choose predictor series is required.

***Investment allocation.***

The methodology developed provided a long-flat indicator, fully invested or full out of the market. Other variations are possible, such as percentage allocations in and out of the market as well as allocations between various asset classes, such as bonds and commodities. In addition, the experiments didn’t consider short selling, a way to profit from an expected market downturn.

## Summary

This dissertation aimed to improve the prediction accuracy of non-linear and non-stationary time series in genetic programming. Artificial intelligence approaches are often used for such prediction problems where no deterministic solution is known to exist or where the series to be predicted is characterized by excess noise and the appearance of random behavior. These time series, especially in certain domains such as finance, are also thought to undergo abrupt changes to the underlying data generating process, known as regime change. Most methods of time series prediction, and almost all prior approaches that use genetic programming, do not consider regime change. Regime change was a primary consideration of this dissertation.

The dissertation goal was achieved by the introduction of new features specifically designed to enhance genetic programming modularity to enable regime specific processing and behavior. Prior research has demonstrated the benefits of incorporating programming language modularity features such as functions, recursion, and lambda expressions into the evolutionary process. Modularity can be further enhanced by incorporating software design patterns into the GP process. Such features will also yield evolved programs more similar to what a human programmer would produce, as software design patterns are based on observed human behavior and best practices.

Using the abstract template software design pattern as inspiration, two approaches to regime handling were introduced. The first approach, automatically defined templates (ADT), is an extension of automatically defined functions with the addition of regime specific implementations of each function. The second approach, automatically acquired templates (AAT), is an extension of module acquisition allowing regime specific implementations of each module in a shared library. Both approaches add a regime indicator branch to the canonical GP

algorithm. Regime indicator programs determine which regime specific template implementations are chosen during program fitness evaluation.

A Java based genetic programming was developed that incorporated the new approaches presented in this dissertation. The custom system also enabled detailed collection of experimental data. Other commonly seen GP alternatives, such as ADF, were also implemented to enable a better comparison of the proposed approach to existing methods.

Experiments were performed using both synthetically generated and real work time series including the S&P 500, a commonly benchmarked financial series index. By piecing together different time series, the synthetic series tests allowed the simulation and analysis of abrupt regime changes. The financial series tests exercised the new methodology on real world data without clear cut regime boundaries and characterized by randomness and noise on top of underlying and changing trends. The new methodologies were compared to commonly seen GP approaches, such as automatically defined functions. DyFor GP, the only evolutionary method found to consider regime change, was also included in the comparison and was used as a benchmark for measuring the regime handling performance of the new approaches. Several prior experimental studies concerning synthetic and financial series prediction were reproduced. The exact parameters used in these studies were replicated where and to the extent possible.

Experimental results showed that ADT consistently outperformed other genetic programming approaches, including DyFor GP, in most tests. ADT also significantly outperformed AAT in almost all experiments. Two versions of ADT, coupled and non-coupled, were implemented and tested. The coupled approach permanently joins together an individual regime detection program with a result producing program. The decoupled approach brings these two programs together at fitness calculation time. The coupled ADT approach also

outperformed decoupled ADT, due to more consistent indication of regimes and regime specific code. Therefore, only coupled ADT is recommended for future use and development.

Market data experiments looking at the S&P 500 index showed all evolutionary methods tested had difficulty beating a buy and hold approach when transaction costs were considered. However, when transaction costs were ignored, the majority of evolutionary methods beat the buy and hold benchmark, with coupled ADT performing best in the majority of tests. ADT and DyFor GP returned 16.8% and 15.7% respectively, for the periods 1999-2004 compared to a return of -1.89% achieved using a buy and hold strategy. The continual reduction in transaction costs as well as new trading platforms promising zero transaction cost make an automated approach more feasible for market investors.

Even though this research may uncover better methods than are currently available to predict financial markets, such an undertaking must be attempted with humility. If a single proven method was found and made public, it would likely be widely implemented rather quickly. Therefore, new methods can only gain temporary advantage over other established practices.

Luckily, or not, the market is influenced by the individual psychologies of countless participants whose beliefs continually change. Therefore, an algorithm must continually change and adapt through constant improvement and shorter and shorter time horizons. Such is the perilous task of attempting to beat the stock market.

# Appendix A

## Design Pattern Example

This section further explains the use and implementation of the template method and strategy patterns through an admittedly contrived, but simple to understand, example. Appendix B provides an additional, less introductory, example more relevant to the domain of this dissertation.

Navigation problems have been a applied with GP since early research, such as the artificial ant problem in (Koza, 1992, p. 147-162). This example considers the search for the fastest route through a simple Autocross course, shown in Figure A1. The goal of this problem is to navigate a vehicle through the course in the shortest time without crashing (hitting the wall or losing control). Fitness is evaluated based on the total time to complete the race. A crash evaluates to an appropriately high (lower is better) fitness score.

*Figure A1.* Sample racing domain.

Both vehicles may perform the following operations: accelerate (magnitude), brake (magnitude), left (magnitude), right (magnitude). A possible solution is simply:

Accelerate->left->right>brake

This solution fails when realistic physics are considered as the vehicle would likely lose control while trying to perform a 90 degree turn at full speed without first braking. In addition, the mechanics of controlling a motorcycle differ from an automobile.

A more plausible solution to the above navigation problem is the following.[17] A distinct Java class is created for each vehicle type and a "race" method is defined in each. The mechanics of controlling a motorcycle are more involved than controlling an automobile as individuated by large number of operations. The details of this difference are less important of the need to various detailed implementation for each motor vehicle type.

```
public class Automobile {
    public void race() {
        accelerate(100);    //0-100%
        brake(100);    //0-100%
        turnWheel(-90);    //Turn angle
        accelerate(100); //0-100%
        brake(100);    //0-100%
        turnWheel(90);    //Turn angle
        accelerate(100); //0-100%
        brake(100);    //0-100%
    }

    public void accelerate(double percent) {…}

    public void brake(double percent) {…}

    private void turnWheel(double direction) {…}
}
```

*Figure A2.* Initial implementation of Automobile class in autocross simulator example.

[17] More realistic physics are not considered this example scenario. It would also appear that the automobile would obviously win this race as it requires less cautious operations. The motorcycle's acceleration advantage can, however, overcome this. It is necessary to find the correct optimal parameters without crashing. This search problem is a type commonly applied to GP.

```
public class Motorcycle {
    public void race() {
        safeAccelerate(50);     //0-100%, additional checks
        safeAccelerate(100);     //0-100%, additional checks
        safeBrake(80);         //0-100%, additional checks
        pressHandlebars(-90);     //Turn angle
        lean(-45);         //lean angle
        safeAccelerate(50);     //0-100%, additional checks
        safeAccelerate(100);     //0-100%, additional checks
        safeBrake(80);         //0-100%, additional checks
        pressHandlebars(90);  //Turn angle
        lean(45);         //lean angle
        safeAccelerate(50);     //0-100%, additional checks
        safeAccelerate(100);     //0-100%, additional checks
        safeBrake(80);         //0-100%, additional checks
    }

    private void safeAccelerate(double percent {…}

    private void safeBrake(double percent) {…}

    private void lean(double percent {…}

    private void pressHandlebars(double degree {…}
}
```

*Figure A3.* Initial implementation of Motorcycle class in autocross simulator example.

The class model shown in Figure A3 can be simplified by incorporating reusable functions. Functional expressions are language constructs and have been incorporate into genetic programming using various approaches such as automatically defined functions (Koza, 1994). A functional approach to the problem might notice that the "turn" operation involves the same three repeated steps. A new function can be defined for the Automobile and Motorcycle classes and the main programs modified to incorporate this function.

```
private void turn(double direction){
    brake(100);
    turnWheel(direction);
    accelerate(100);
}
```

*Figure A4.* Turn method for Automobile class in autocross simulator example.

```
public void turn(double degree) {
    safeBrake(80);
    pressHandlebars(degree);
    lean(degree/2);
    safeAccelerate(50);
    safeAccelerate(100);
}
```

*Figure A5.* Turn method for Motorcycle class in autocross simulator example.

This change reduces the complexity of the race method in the main program branch.

```
public void race(){
    accelerate(100);
    turn(-90);
    turn(90);
    brake(100);
}
```

*Figure A6.* Race method for Automobile class in autocross simulator example.

```
public void race() {
    safeAccelerate(50);
    safeAccelerate(100);
    turn(90);
    turn(-90);
    safeBrake(80);
}
```

*Figure A7.* Race method for Motorcycle class in autocross simulator example.

A problem with the implementation at this point is that both vehicle classes use separate controller code. Both programs appear to contain some similarities but the actual steps and implementations differ.

Object-oriented (OO) design and design patterns can improve this design. An OO analysis may show that both the automobile and the motorcycle are actually types of motor vehicles, indicating an inheritance relationship. Furthermore, all motor vehicles have certain operations in common.

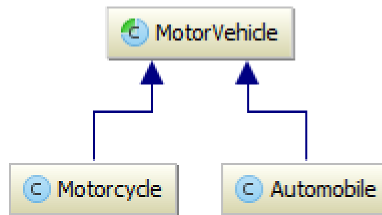



*Figure A8.* Object model showing one parent and two subclasses.

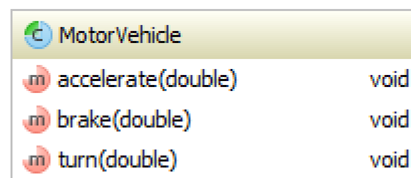



*Figure A9.* Methods in abstract parent class.

In Figure A8, MotorVehicle is defined as an abstract class. It doesn't exist in the real world but is instead an abstraction of similarly attributed items. Car and Motorcycles are concrete classes. These objects model real world entities. In OO design, abstract classes may contain methods and operations but may not be directly instantiated. Just as individual operations may be abstracted, compound operations, such as race(), can also abstracted.

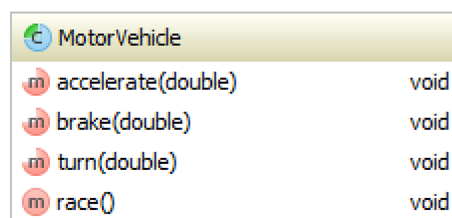



*Figure A10.* Additional methods in abstract parent class. The race methods in both subclasses are moved up into the parent.

```
public void race() {
    accelerate(100);
    turn(-90);
    turn(90);
    brake(100);
}
```

*Figure A11.* Race method in abstract class. Each step in this method is defined as abstract.

The race method in Figure A11 may be applied to any of the two MotorVehicle subclasses. Each will have a distinct implementation of the steps. The implementation details are defined for each concrete class.

The final class diagram, in Figure A12, shows the abstract methods and the implementations of each. Private, class specific methods are not shown. The final code for the Abstract and concrete classes are show in Figure A13 through Figure *A15*.

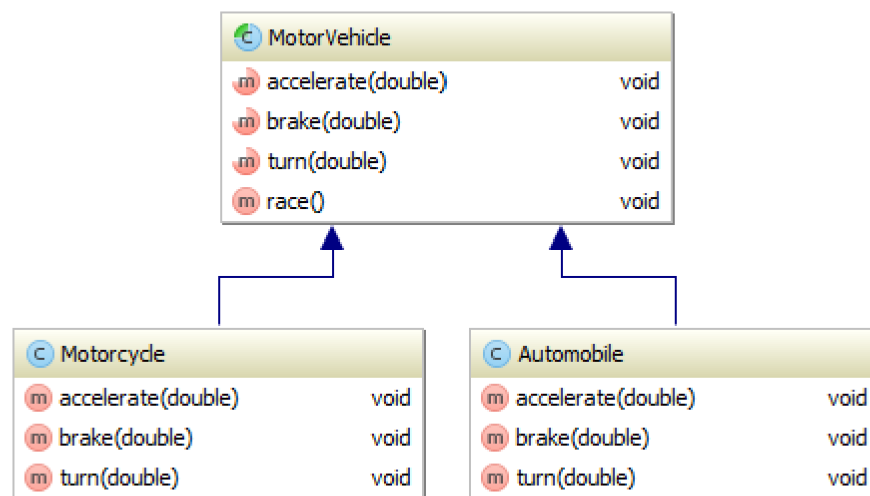



*Figure A12.* Full class diagram for abstract and concrete classes.

```
public abstract class MotorVehicle {
    public abstract void accelerate(double percent);

    public abstract void brake(double percent);

    public abstract void turn(double degree);

    public void race() {
        accelerate(100);
        turn(-90);
        turn(90);
        brake(100);
    }
}
```

*Figure A13.* Java code for abstract parent class.

```
public class Automobile extends MotorVehicle {
    @Override
    public void turn(double direction) {
        brake(100);
        turnWheel(direction);
        accelerate(100);
    }

    @Override
    public void accelerate(double percent {…}

    @Override
    public void brake(double percent) {…}

    private void turnWheel(double direction) {…}
}
```

*Figure A14.* Java code for Automobile concrete class.

```
public class Motorcycle extends MotorVehicle {

    @Override
    public void turn(double degree) {
        safeBrake(80);
        pressHandlebars(degree);
        lean(degree / 2);
        safeAccelerate(50);
        safeAccelerate(100);
    }

    @Override
    public void accelerate(double percent) {
        safeAccelerate(percent / 2);
    }

    private void safeAccelerate(double percen) {…}

    @Override
    public void brake(double percent) {
        safeBrake(percent * 0.8);
    }

    private void safeBrake(double percent {…}

    private void lean(double percent) {…}

    private void pressHandlebars(double degree {…}
}
```

*Figure A15.* Java code for Motorcycle concrete class.

The template method pattern defines an overall algorithm and invariant operations in an abstract class (the race function in MotorVehicle) and defers certain implementation steps to

concrete subclasses (accelerate, brake, and turn, as defined in Automobile and Motorcycle). Comparing the original program code to the code with added functions to the code with abstract methods shows the simplicity achieved through greater levels of abstraction, modularity, and composition.

Another design pattern, the strategy pattern, is similar in many ways to the template method pattern and also applicable to regime specific implementations. The strategy pattern allows alternative algorithm implementations to be selected and applied to a given task. This pattern accomplishes specific behavior not through inheritance but through composition, where varying behaviors, or strategies, are injected into a concrete class. In the racetrack example above, various approaches to performing the same task may be applied and used by each of the motor vehicle implementation. For example, vehicle braking could be either conservative or aggressive. An abstract class with an abstract brake method and two implementation approaches are defined.

Instead of controlling the algorithm from an abstract method as in the template pattern, the algorithm is controlled by a concrete class (the Automobile object). The automobile references an abstract braking strategy to perform the operation. The actual algorithm to perform braking is determined at runtime (either at instantiation, as indicated by the new class constructor, or modified at a later time prior to use).

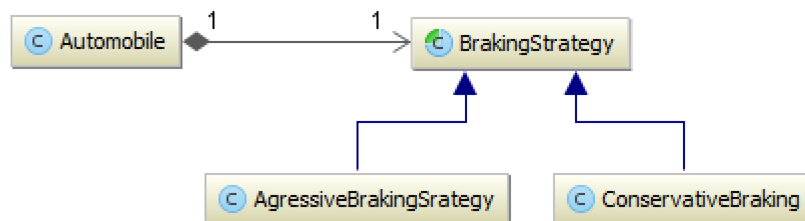



*Figure A16.* Object model for strategy pattern implementation.

```java
public class Automobile {
    private BrakingStrategy brakingStrategy;

    public Automobile(BrakingStrategy brakingStrategy) {
        this.brakingStrategy = brakingStrategy;
    }

    public void race() {
        accelerate(100);
        turn(-90);
        turn(90);
        brakingStrategy.brake(100);
    }

    public void accelerate(double percent) {
        //Implementation detail left out
    }

    private void turnWheel(double direction) {
        //Implementation detail left out
    }

    private void turn(double direction) {
        brakingStrategy.brake(100);
        turnWheel(direction);
        accelerate(100);
    }
}
```

*Figure A17.* Java code for applying strategy pattern in concrete Automobile class.

# Appendix B

## Stock Prediction Example

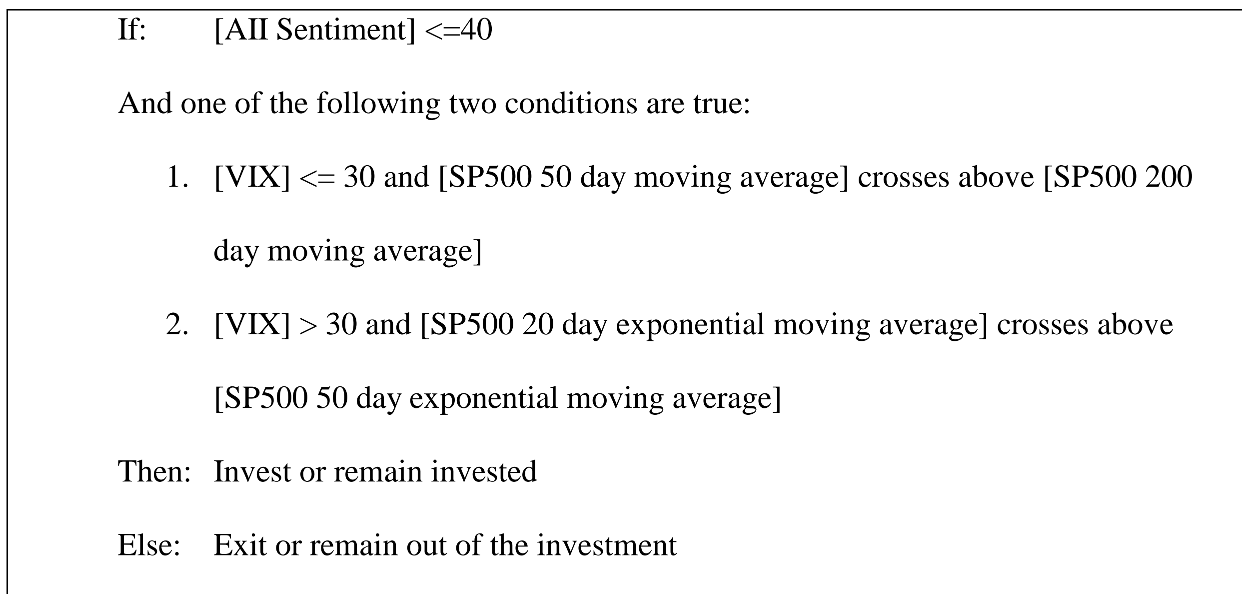
The following hypothetical example describes a financial model to guide investment decisions. The example doesn't presuppose genetic programming or other methodology, but is intended to illustrate the improvements possible by incorporate more modular, pattern oriented code into a program.

Suppose it is determined that moving average crossover is a good indication for buy/sell decisions. A moving average crossover occurs when two moving averages of different period on the same time series cross. A lower period moving average crossing above a higher moving period average is widely seen as a bullish signal (Sincere, 2011). However, it is discovered that a simple moving average is preferred in non-volatile market while an exponential moving average is preferred in a volatile one. Additional analysis has also shown that negative sentiment is a bullish signal (ibid.). A model to encompass this logic is created. If the model evaluates to true at any point in time, the model will invest or remain in the market. If the model evaluates to false, the model will stay out or exit the market. The market in this example is taken to be the S&P 500 index (S&P Dow Jones Indices LLC, 2016).

The model uses the following indicators:

1. [ma/ema]: S&P 500 moving average or exponential moving average. All moving average calculations below based on the S&P 500 index.
2. [VIX]: CBOE VIX volatility index (Chicago Board Options Exchange, 2014)
3. [AII Sentiment]: AAII Sentiment Survey (American Association of Individual Investors, 2014)

The complete model logic shown in Figure B1.


If: [AII Sentiment] <=40

And one of the following two conditions are true:

1. [VIX] <= 30 and [SP500 50 day moving average] crosses above [SP500 200 day moving average]
2. [VIX] > 30 and [SP500 20 day exponential moving average] crosses above [SP500 50 day exponential moving average]

Then: Invest or remain invested

Else: Exit or remain out of the investment

*Figure B1.* Example investment model logic.

**Java implementation.**

A Java program implementing the model described in above is shown in Figure B2. The program can be improved by applying the template method pattern. An analysis of the program shows that the sentiment condition is an invariant that must always be satisfied. A moving average calculation is always applied, but different moving averages are used as determined by the value of the VIX index. Therefore, two regimes can be determined by a single indicator encompassing the VIX related condition. Two regime specific implementations of an abstract algorithm are also needed.

The class diagram for the new version of the Java program is shown in the Figure B3. An abstract class, AbstractInvestor, is created to contain the invariant logic. Two regime specific implementations are created to provide the appropriate moving average calculation. A regime determination function is also incorporated to choose among the two possible regimes. The modified code is shown in Figure B4 through Figure B8.

```
public boolean evaluate() {

    if (sentiment(today) <= 40 &&
            vix(today) <= 30 &&
            ma(200, yesterday) <= ma(50, yesterday) &&
            ma(200, today) > ma(50, today)) {
        return true;
    } else if (sentiment(today) <= 40 &&
            vix(today) > 30 &&
            ema(20, yesterday) <= ema(50, yesterday) &&
            ema(20, today) > ema(50, today)) {
        return true;
    } else {
        return false;
    }
}
```

*Figure B2.* Java method for example investment decision

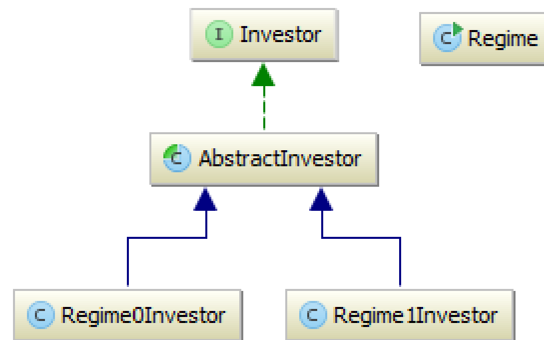



*Figure B3.* Class diagram for template method pattern implementation of investment decision example.

A regime determining program is shown in Figure B4. This program returns a binary string, 0 or 1 in this case, based on the single indicator base on the VIX index value.

```
public class Regime {
    public static int regime() {
//create binary result from single indicator
        String binaryString = indicator1() ? "1" : "0";
        return Integer.parseInt(binaryString, 2);
    }

    //Single indicator
    public static boolean indicator1() {
        return FinancialUtilities.vix(today) > 30;
    }
}
```

*Figure B4.* Regime indicator implementation.

Figure B5 defines an interface, which is simply the Boolean target expression. Figure B6 implements the interface in an abstract class. This class implements a template method pattern, incorporating invariant and regime specific logic. The regime specific logic must be defined as abstract in the Java class. Figure B7 defines two concrete classes implementing the regime specific logic. The actual algorithm is evaluated and regime selection is done in the following application program, shown in Figure B8.

```
public interface Investor {
    boolean invest();
}
```

*Figure B5.* Investment decision Java interface.

```
public abstract class AbstractInvestor implements Investor {
    @Override
    public boolean invest() {
        return FinancialUtilities.sentiment(today) <= 40 //invariant
                && investRegime();       //abstract - regime specific
    }

    public abstract boolean investRegime(); //Must be overridden
}
```

*Figure B6.* Template method Java implementation.

```
public class Regime0Investor extends AbstractInvestor {
    @Override
    public boolean investRegime() {
        return (
                ma(200, yesterday) <= ma(50, yesterday) &&
                        ma(200, today) > ma(50, today));
    }
}

public class Regime1Investor extends AbstractInvestor {
    @Override
    public boolean investRegime() {
        return (ema(20, yesterday) <= ema(50, yesterday) &&
                ema(20, today) > ema(50, today));
    }
}
```

*Figure B7.* Regime specific logic implementing abstract template methods.

The modular version, while providing the same ultimate result as the prior version, is easier to understand and better highlights the actual underlying model logic. Such understanding

is even more important for automated programming techniques such as genetic program, where the underlying pattern or logic may not easily be determined.

```
public class evaluator {
    public static void main(String[] args) {
        Investor template;//placeholder for concrete template
        int regimeNumber = Regime.regime(); //determine regime
        //inject appropriate template implementation
        if (regimeNumber == 0) {
            template = new Regime0Investor();
        } else {
            template = new Regime1Investor();
        }
        Boolean result = template.invest()); //run calculation
    }
}
```

*Figure B8.* Final program using template method pattern.

**Clojure implementation.**

The algorithm can also be implemented in Clojure[18], a JVM language similar to LISP. A LISP/S-expression representation is typical for representing GP trees. The initial, non-modular approach is shown in Figure B9.

```
(defn invest? []
  (or
    (and
      (<= (sentiment today) 40)
      (<= (vix today) 30)
      (<= (ma 200 yesterday) (ma 50 yesterday))
      (> (ma 200 today) (ma 50 today)))
    (and
      (<= (sentiment today 40))
      (> (vix today) 30)
      (<= (ema 20 yesterday) (ema 50 yesterday))
      (> (ema 20 today) (ema 50 today)))
    )
  )
```

*Figure B9.* Clojure implementation of investment decision example using a single function.

[18] Even through Clojure is not used as an evaluation language in this dissertation, the underlying Java implementation and program tree model a LISP/Clojure S-expression.

```
;  concrete regime specific implementation 1
(defn adt00 []

  (and
    (<= (ma 200 yesterday) (ma 50 yesterday))
    (> (ma 200 today) (ma 50 today)))
  )

;  concrete regime specific implementation 2
(defn adt01 []
  (and
    (<= (ema 20 yesterday) (ema 50 yesterday))
    (> (ema 20 today) (ema 50 today)))
  )

;choose concrete implementation based on regime selection logic
(defn regime [date]
  (if (> 30 (vix date))
    adt00
    adt01
    )
  )

;main expression
(defn investRegime? []
  (let [adt (regime today)] ;set concrete regime handler

    (and
      (<= (sentiment today) 40) ; invariant logic
      (adt) ;regime specific logic
      )
    )
  )
```

*Figure B10.* Clojure implementation of investment decision example incorporating modularity features.

This function can be expressed with more modular approaches is shown in Figure B10.

The overall algorithm can be evaluated in Clojure with the expression: (investRegime?).

# Appendix C

## Primitives

The following section describes the terminal and function primitives used in the genetic programming system and experiments created for this dissertation.

### Functions.

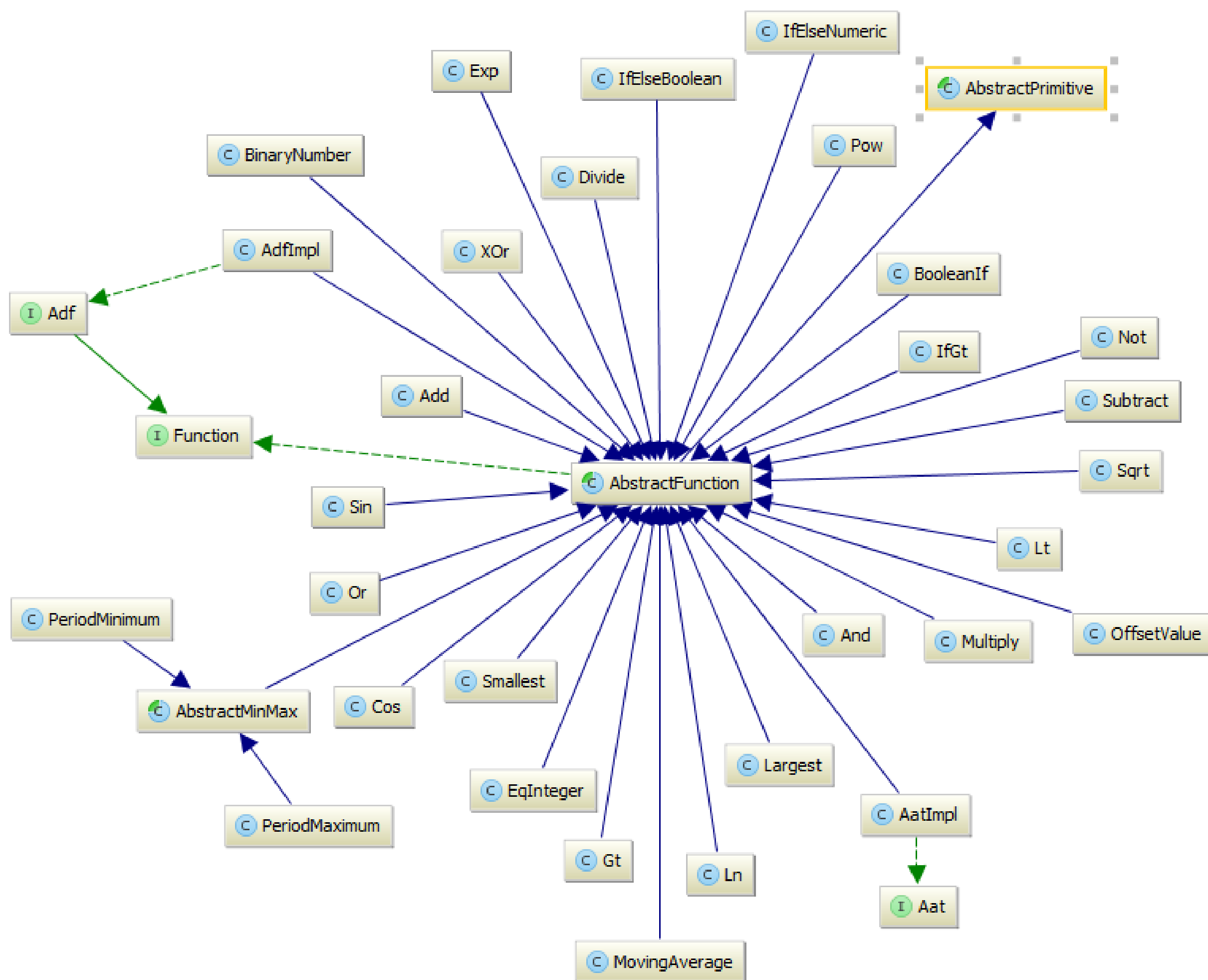



*Figure C1*. Primitive functions class diagram.

Table C1. Function Primitives

| Function | Data Type | Arity | Description |
|---|---|---|---|
| Add | N | 2 | Numeric Addition |
| And | B | 2 | Logical AND |
| BinaryNumber[a] | N | Variable | See regime determination |
| Cos | N | 1 | Cosine |
| Divide | N | 2 | Numeric division |
| EqInteger | B | 2 | Equality of integer representation of a numeric |
| Exp | N | 1 | Euler's number raised to a power |
| Gt | B | 2 | Numeric comparison |
| IfElseBoolean | B | 3 | Returns a Boolean based on logical condition |
| IfElseNumeric | B | 3 | Returns a Numeric based on logical condition |
| Largest | N | 2 | Returns largest value in a numeric comparison |
| Ln | N | 1 | Natural Logarithm |
| Lt | B | 2 | Logical less than |
| MovingAverage[b] | N | 1 | Parameterized moving average. |
| Multiply | N | 2 | Numeric multiplication |
| Not | B | 1 | Logical negation |
| OffsetValue | N | 1 | Returns the series value offset by a given numeric value |
| Or | B | 2 | Logical OR |
| PeriodMaximum | N | 1 | Returns the maximum value of a series within a given period |
| PeriodMinimum | N | 1 | Returns the minimum value of a series within a given period |
| Pow | N | 2 | Exponentiation |

| Function | Data Type | Arity | Description |
|---|---|---|---|
| Sin | N | 1 | Sine |
| Smallest | N | 2 | Returns smallest of two numeric values |
| Sqrt | N | 1 | Square Root |
| StdDev | N | 1 | Standard Deviation of values within a given numeric period |
| Subtract | N | 2 | Numeric subtraction |
| XOr | | 2 | Logical XOR |

*Note*. B=Boolean; N=Numeric.
[a]Binary string are used only in regime determination. Therefore, the length is dependent on the fixed number of regimes. [b]Integer component of parameter is used in case of a non-integer parameter.

**Terminals.**

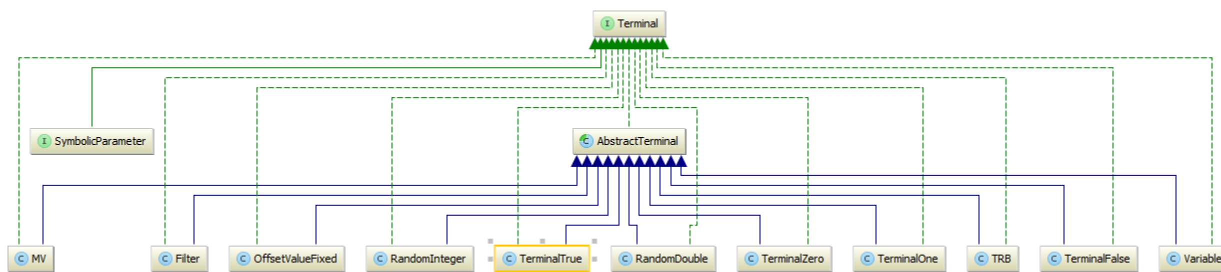



*Figure C2.* Primitive terminals class diagram.

Table C2. Terminal Primitives

| Terminal | Description |
| --- | --- |
| Filter[a] | Technical indicator taken from (Jin Li & Tsang, 1999) defined as current price minus the minimum price over a prior period. Period is function input value |
| MV[a] | Technical indicator taken from (Jin Li & Tsang, 1999) defined as current price minus the average price over a prior period. Period is function input value |
| OffsetValueFixed | Fixed offset on a named series. Period is program input value |
| RandomDouble | An immutable random value between 0 and 1 |
| RandomInteger | An immutable random value within a fixed range. Range is program input value. |
| TerminalFalse[b] | Immutable Boolean false value |
| TerminalOne [b] | Immutable numeric one value |
| TerminalTrue[b] | Immutable Boolean true value |
| TerminalZero[b] | Immutable numeric 0 value |
| TRB[a] | Technical indicator taken from (Jin Li & Tsang, 1999) defined as current price minus the maximum price over a prior period. Period is function input value |
| Variable | Represents the X value in a repression or prediction. X can be numeric or Date data type |

[a]These terminals were not used in this research, but are implemented and available in the developed system. These terminals are an example of compound functionality packaged as a non-reducible terminal. Alternatively, this could be evolved from lower level functions and terminals. [b]These terminals are often only used internally when no other primitive is available and a certain strong type is needed.

# Appendix D

## Program Parameters

This appendix provides a detailed description of the program parameters used for each experiment and a description of each parameter.

### Parameter descriptions.

Table D1. Common Parameters Used in Experiments

| Parameter | Default Value[a] | Description |
|---|---|---|
| allowTrivialPredictions[b] | true | Detect if a prediction program appears to converge on the prior actual value. Setting this value to false will result in a low fitness score for such a program. |
| applicationName | [linearRegressionApp, linearRegressionAppDyforGp, MarketPredictionApp, MarketProfitApp, MarketProfitAppDyforGp] | Type of program to run |
| crossoverPct | 0 | Crossover probability percentage |
| description | | Informational only |
| direction | [asc, desc] | Indicates direction of decreasing fitness values |
| elitist | false | Fittest individuals are automatically promoted (via reproduction) to next generation[c] |
| endTest | Series start +90% | Testing end window. Can be indicated in numeric or percent. |
| endTrain | Series start + 66.66 % | Training end window. Can be indicated in numeric or percent. |

| Parameter | Default Value[a] | Description |
|---|---|---|
| functions | | Available functions for result producing programs. Also used for regime determining programs if regimeFunctions parameter is not specified |
| logMetrics | false | Record metrics to database |
| maxDepth | unlimited | Maximum program depth |
| maxInitDepth | | Maximum depth for initial random population generation. This limit is also used for during mutation. |
| maxValidFitness[d] | unlimited | Set an upper limit on fitness scores. Any score over this value will be set to the fitness limit. |
| maxPredictionGenerations | 1 | Used with adaptive training. Determines the maximum number of training generations that can be run prior to each prediction step |
| maxSize | unlimited | Maximum nodes for an individual. |
| maxTotalNodes | unlimited | Maximum total number of nodes. If this parameter is used, population size may fluctuate. |
| meanSquaredError | true | Use mean square (true) or mean error (false) when calculating fitness in numeric prediction runs. |
| mutationPct | 0 | Mutation probability percentage |
| predictionGenerations[e] | 1 | Number of training runs executed at each prediction iteration. |

| Parameter | Default Value[a] | Description |
|---|---|---|
| predictionWindow | 1 | Predictions are made at this offset past the current window end point. |
| printTrainingProgram | false | Log the best training program after each generation to standard output. |
| populationSize | | Initial and maximum population size. If maximumTotalNodes parameter is used, this will be used as the initial population size |
| predictors[f] | | Set of target series to use as predictors |
| predictionStep | 1 | Number of steps to move sliding window after each prediction generation |
| programType | [prediction, regression] | Type of program run. |
| returnType | [number, boolean] | Data type returned by generated programs |
| riskFreeReturn[f] | | Use this series to calculate investment returns when not invested in the market |
| selectionStrategy[g] | [tournamentSelectionStrategy] | Selection strategy used to choose individuals for next generation |
| seriesEnd | 0 | Used to limit data in target series. Can be input as actual data point or percentage. |
| seriesStart | 100% | Used to limit data in target series. Can be input as actual data point or percentage. |
| signals[f] | 1 | Number of consecutive signals needed to act on an investment decision. |

| Parameter | Default Value[a] | Description |
|---|---|---|
| stagnationLimit | | Abort further training if a more fit individual is not found in this many generations |
| startTest | 0 | Testing start window. Can be input as actual data point or percentage. |
| startTrain | 33.33% | Training start window. Can be input as actual data point or percentage. |
| target | | Target series to predict. |
| terminals | | Terminal set available to evolved programs |
| testingGenerations | 1 | Incorporate multiple training/ testing cycles. The best tested individual over all cycles will be used for the subsequent prediction phase. |
| tournamentSize | 2 | Number of programs included in a tournament |
| trainingGenerations | 1 | Number of training generations. |
| trainingWindow | | Training window size. If not entered, the entire target series is used. |
| transactionCost[f] | 0 | Ply a fixed or percentage cost to each investment bought or sold |
| trainingStep[h] | 1 | Number of data points to move forward after each fitness evaluation over the training window. |
| useAverageError | false | Use average error (true) or total error (false) in fitness evaluations involving multiple comparisons. |

| Parameter | Default Value[a] | Description |
|---|---|---|
| useAverageFitnessSelector | false | For four way tournaments only, use average (true) or best (false) of two fitness calculations. |
| useAdaptiveTraining | false | Increase the training generations during prediction phase as prediction decreases in accuracy. |
| visualize | false | display real time graphics |

[a] Allowable range or values are shown in brackets. Default values within allowable ranges are marked with an asterisk. Parameters with default values specified are not required. [b]Trivial predictions describe the case where the predicted value $y(x) = y(x-1)$, where equality is defined as values within a given tolerance. In the current implementation, trivial predictions are taken to be cases where 95% of predicted values are within 0.0001. [c]The fittest individual from both the result producing branch and, if applicable, the regime determining branch are copied to the next generation via reproduction. The latter case is only applicable where the regime population is completely decoupled from the result producing population. [d] Maximum allowable fitness is generally only needed where fitness decreases with higher fitness values, as in symbolic regression, where the fitness of the worst performing individuals can approach infinity. This situation can skew population statistics or cause program aborts. [e]A value of 0 for this parameter will execute all predictions with no additional training after the initial training period. [f]Relevant to market prediction only. [g] Tournament selection is the only currently implemented selection strategy. Fitness proportional selection is another selection strategy that could be included. Random selection can be implemented by setting a tournament size of 1. [h]Larger training steps can be used to sample values from the target series in order to optimize fitness evaluation.

Table D2. ADT/AAT Parameters

| Parameter | Default Value[a] | Description |
|---|---|---|
| regimes | | Preset number of regimes |
| regimeFunctions | Result producing branch functions | Functions available to regime determining programs. |
| regimePopulationSize | result producing branch populationsize | Steady state regime population size. Only applicable for decoupled regime generation. If maximumTotalNodes parameter is used, this is the initial regime population size. |

[a] Parameters with default values specified are not required.

Table D3. ADT Parameters

| Parameter | Default Value[a] | Description |
|---|---|---|
| adfArity | | The arities used for each defined ADF and ADT template. |

[a] Parameters with default values specified are not required.

Table D4. AAT Parameters

| Parameter | Default Value[a] | Description |
|---|---|---|
| compressionPct | 0[b] | Compression Percentage |
| expansionPct | 0 | Expansion Percentage |
| minimumCompressionSize | 0 | Disallow compressions that would result in program size less than a minimum. Compression is performed again in such a case. |

[a] Parameters with default values specified are not required. [b]While technically allowable, a value of 0 for compression would result in no library functions created.

Table D5. DyFor GP Parameters

| Parameter | Default Value[a] | Description |
|---|---|---|
| maxWindowSize | | Maximum sliding window size |
| minWindowSize | | Minimum sliding window size |
| N | | Number of consecutive increases or decreases in prediction accuracy before adjusting sliding windows[b] |
| predictionSize | 1 | Predict this many points ahead of end of sliding window |
| resetOnNoTrend | false | Reset N if small and large window fitness scores are equal. |
| saveoff | 1 | number of programs to save off |
| startWindowSize | | Start sliding window size |
| windowDifference | | Difference between small and large windows |
| windowSlide | | Number of points to slide windows. |

*Note*. Typical DyFor GP Parameters as described in (Wagner & Michalewicz, 2008).
[a] Parameters with default values specified are not required. [b]Indicates a regime change.

**Parameter files.**

This section provides sample parameter files used in the described experiments. Windows batch and Unix shell script formats are both used. Information on downloading the full set of parameter files is given in Appendix I.

***LGOZLG.***

```
java  -Xmx7G -Djava.awt.headless=true  -cp /home/ubuntu/target/nova-1.2-SNAPSHOT.jar
com.infoblazer.gp.Application \
--description="LGOZLG ADT Prediction"  \
--tournamentSize=4                     \
--logMetrics=true                      \
--visualize=false                         \
--allowTrivialPredictions=false           \
--useAdaptiveTraining=false                \
--maxPredictionGenerations=5                \
--printTrainingProgram=false                 \
--maxPrediction=10                           \
--minPrediction=-10                          \
--regimes=2                                  \
--adfArity="1,2"                             \
--startTrain=150                             \
--endTrain=250                               \
--startTest=150                              \
--endTest=250                                \
--returnType=number                          \
--applicationName=linearRegressionApp        \
--elitist=true                               \
--programType=Prediction                     \
--target=LGOZLG                              \
--direction=asc                              \
--populationSize=3000                        \
--regimePopulationSize=3000                  \
--trainingWindow=110                         \
--maxInitDepth=5                             \
--maxDepth=10                                \
--fourWayPct=25                              \
--mutationPct=10                             \
--crossoverPct=90                            \
--trainingGenerations=41                     \
--selectionStrategy=tournamentSelectionStrategy \
--functions="add,subtract,multiply,divide,sin,cos,sqrt,exp,ln" \
--
regimeFunctions="add,subtract,multiply,divide,and,not,gt,offsetValue,periodMinimum,p
eriodMaximum,stdDev,movingAverage" \
--terminals="randomInteger(-1 110),offsetValueFixed(LGOZLG
1),offsetValueFixed(LGOZLG 2)"
```

*Figure D1.* ADT LGOZLG parameters

```
java  -Xmx7G -Djava.awt.headless=true  -cp /home/ubuntu/target/nova-1.2-SNAPSHOT.jar
com.infoblazer.gp.Application \
--description="LGOZLG ADT OMNI Prediction"  \
--tournamentSize=4                     \
--logMetrics=true                      \
--visualize=false                         \
--allowTrivialPredictions=false           \
--useAdaptiveTraining=false                \
--maxPredictionGenerations=5                \
--printTrainingProgram=false                 \
--maxPrediction=10                           \
--minPrediction=-10                          \
--regimes=2                                  \
--adfArity="1,2"                             \
--startTrain=150                             \
--endTrain=250                               \
--startTest=150                              \
--endTest=250                                \
--returnType=number                          \
--applicationName=linearRegressionApp        \
--elitist=true                               \
--programType=Prediction                     \
--target=LGOZLG                              \
--direction=asc                              \
--populationSize=3000                        \
--regimePopulationSize=1                  \
--trainingWindow=110                         \
--maxInitDepth=5                             \
--maxDepth=10                                \
--fourWayPct=25                              \
--mutationPct=10                             \
--crossoverPct=90                            \
--trainingGenerations=41                     \
--selectionStrategy=tournamentSelectionStrategy \
--functions="add,subtract,multiply,divide,sin,cos,sqrt,exp,ln" \
--regimeFunctions="regimeLGOZLG"                               \
--terminals="randomInteger(-1 110),offsetValueFixed(LGOZLG
1),offsetValueFixed(LGOZLG 2)"
```

*Figure D2.* ADT Omni LGOZLG parameters.

```
java  -Xmx7G -Djava.awt.headless=true  -cp /home/ubuntu/target/nova-1.2-SNAPSHOT.jar
com.infoblazer.gp.Application \
--description="LGOZLG AAT Prediction" \
--tournamentSize=4  \
--logMetrics=true \
--allowTrivialPredictions=false  \
--useAdaptiveTraining=false     \
--maxPredictionGenerations=5   \
--maxPrediction=10             \
--minPrediction=-10             \
--regimes=2       \
--gcFrequency=20 \
--startTrain=150                             \
--endTrain=250                               \
--startTest=150                              \
--endTest=250                                \
--returnType=number  \
--applicationName=linearRegressionApp \
--elitist=true              \
```

```
--programType=Prediction    \
--target=LGOZLG             \
--direction=asc             \
--populationSize=3000     \
--regimePopulationSize=3000 \
--trainingWindow=110        \
--maxInitDepth=5          \
--maxDepth=10               \
--mutationPct=10      \
--crossoverPct=80       \
--compressionPct=5       \
--minimumCompressionSize=5 \
--expansionPct=5          \
--trainingGenerations=41   \
--selectionStrategy=tournamentSelectionStrategy                  \
--functions="add,subtract,multiply,divide,sin,cos,sqrt,exp,ln"    \
--
regimeFunctions="add,subtract,multiply,divide,and,not,gt,offsetValue,periodMinimum,p
eriodMaximum,stdDev,movingAverage" \
--terminals="randomInteger(-1 110),offsetValueFixed(LGOZLG
1),offsetValueFixed(LGOZLG 2)"
```

*Figure D3.* AAT LGOZLG parameters.

```
java  -Xmx7G -Djava.awt.headless=true  -cp /home/ubuntu/target/nova-1.2-SNAPSHOT.jar
com.infoblazer.gp.Application \
--description="LGOZLG AAT Omni Prediction" \
--tournamentSize=4  \
--logMetrics=true \
--allowTrivialPredictions=false  \
--useAdaptiveTraining=false     \
--maxPredictionGenerations=5   \
--maxPrediction=10             \
--minPrediction=-10             \
--regimes=2       \
--gcFrequency=20 \
--startTrain=150                            \
--endTrain=250                              \
--startTest=150                             \
--endTest=250                               \
--returnType=number  \
--applicationName=linearRegressionApp \
--elitist=true              \
--programType=Prediction    \
--target=LGOZLG             \
--direction=asc             \
--populationSize=3000     \
--regimePopulationSize=1 \
--trainingWindow=110        \
--maxInitDepth=5          \
--maxDepth=10               \
--mutationPct=10      \
--crossoverPct=80       \
--compressionPct=5       \
--minimumCompressionSize=5 \
--expansionPct=5          \
--trainingGenerations=41   \
--selectionStrategy=tournamentSelectionStrategy                  \
--functions="add,subtract,multiply,divide,sin,cos,sqrt,exp,ln"    \
```

```
--regimeFunctions="regimeLGOZLG"      \
--terminals="randomInteger(-1 110),offsetValueFixed(LGOZLG
1),offsetValueFixed(LGOZLG 2)"
```

*Figure D4.* AAT Omni LGOZLG parameters.

```
java  -Xmx7G -Djava.awt.headless=true  -cp /home/ubuntu/target/nova-1.2-SNAPSHOT.jar
com.infoblazer.gp.Application \
--description="LGOZLG Dyfor" \
--tournamentSize=4  \
--logMetrics=true \
--allowTrivialPredictions=false  \
--useAdaptiveTraining=false \
--maxPrediction=10             \
--minPrediction=-10             \
--startTrain=150   \
--endTrain=250    \
--startTest=150    \
--endTest=250     \
--returnType=Number \
--applicationName=linearRegressionDyforGp  \
--programType=Prediction  \
--populationSize=3000    \
--maxInitDepth=5         \
--maxDepth=10               \
--mutationPct=10          \
--crossoverPct=90         \
--target=LGOZLG           \
--trainingGenerations=41   \
--selectionStrategy=tournamentSelectionStrategy \
--direction=asc \
--N=3          \
--saveoff=10     \
--maxWindowSize=200   \
--predictionSize=1    \
--minWindowSize=20    \
--windowSlide=1       \
--windowDifference=20 \
--startWindowSize=80  \
--functions="add,subtract,multiply,divide,sin,cos,sqrt,exp,ln" \
--terminals="randomInteger(-1 110),offsetValueFixed(LGOZLG
1),offsetValueFixed(LGOZLG 2)"
```

*Figure D5.* DyFor GP LGOZLG parameters.

***MGHENMG.***

```
java  -Xmx7G -Djava.awt.headless=true  -cp /home/ubuntu/target/nova-1.2-SNAPSHOT.jar
com.infoblazer.gp.Application \
--description="MGHENMG ADT Prediction" \
--tournamentSize=4                     \
--logMetrics=true                      \
--allowTrivialPredictions=false        \
--useAdaptiveTraining=false            \
--maxPredictionGenerations=5           \
--maxPrediction=10                     \
```

```
--minPrediction=-10                     \
--regimes=2                             \
--adfArity="1,2"                        \
--startTrain=150                        \
--endTrain=250                          \
--startTest=150                         \
--endTest=250                           \
--returnType=number                     \
--applicationName=linearRegressionApp   \
--elitist=true                          \
--programType=Prediction                \
--target=MGHENMG                        \
--direction=asc                         \
--populationSize=3000                   \
--regimePopulationSize=3000             \
--trainingWindow=110                    \
--maxInitDepth=5                        \
--maxDepth=10                           \
--mutationPct=10                        \
--crossoverPct=90                       \
--fourWayPct=25                         \
--trainingGenerations=41                \
--selectionStrategy=tournamentSelectionStrategy \
--functions="add,subtract,multiply,divide,sin,cos,sqrt,exp,ln" \
--
regimeFunctions="add,subtract,multiply,divide,and,not,gt,offsetValue,periodMinimum,p
eriodMaximum,stdDev,movingAverage" \
--terminals="randomInteger(-1 110),offsetValueFixed(MGHENMG
1),offsetValueFixed(MGHENMG 2),offsetValueFixed(MGHENMG 3),offsetValueFixed(MGHENMG
4),offsetValueFixed(MGHENMG 5),offsetValueFixed(MGHENMG 6),offsetValueFixed(MGHENMG
7),offsetValueFixed(MGHENMG 8),offsetValueFixed(MGHENMG 9),offsetValueFixed(MGHENMG
10),offsetValueFixed(MGHENMG 11),offsetValueFixed(MGHENMG
12),offsetValueFixed(MGHENMG 13),offsetValueFixed(MGHENMG
14),offsetValueFixed(MGHENMG 15),offsetValueFixed(MGHENMG
16),offsetValueFixed(MGHENMG 17),offsetValueFixed(MGHENMG
18),offsetValueFixed(MGHENMG 19),offsetValueFixed(MGHENMG
20),offsetValueFixed(MGHENMG 21),offsetValueFixed(MGHENMG
22),offsetValueFixed(MGHENMG 23),offsetValueFixed(MGHENMG
24),offsetValueFixed(MGHENMG 25),offsetValueFixed(MGHENMG
26),offsetValueFixed(MGHENMG 27),offsetValueFixed(MGHENMG
28),offsetValueFixed(MGHENMG 29),offsetValueFixed(MGHENMG
30),offsetValueFixed(MGHENMG 31)"
```

*Figure D6.* ADT MGHENMG parameters.

```
java  -Xmx7G -Djava.awt.headless=true  -cp /home/ubuntu/target/nova-1.2-SNAPSHOT.jar
com.infoblazer.gp.Application \
--description="MGHENMG ADT Prediction Omni" \
--tournamentSize=4                      \
--logMetrics=true                       \
--allowTrivialPredictions=false         \
--useAdaptiveTraining=false             \
--maxPredictionGenerations=5            \
--maxPrediction=10                      \
--minPrediction=-10                     \
--regimes=2                             \
--adfArity="1,2"                        \
--startTrain=150                        \
--endTrain=250                          \
```

```
--startTest=150                        \
--endTest=250                          \
--returnType=number                    \
--applicationName=linearRegressionApp  \
--elitist=true                         \
--programType=Prediction               \
--target=MGHENMG                       \
--direction=asc                        \
--populationSize=3000                  \
--regimePopulationSize=1            \
--trainingWindow=110                   \
--maxInitDepth=5                       \
--maxDepth=10                          \
--mutationPct=10                       \
--crossoverPct=90                      \
--fourWayPct=25                        \
--trainingGenerations=41               \
--selectionStrategy=tournamentSelectionStrategy \
--functions="add,subtract,multiply,divide,sin,cos,sqrt,exp,ln" \
--regimeFunctions="regimeMGHENMG"   \
--terminals="randomInteger(-1 110),offsetValueFixed(MGHENMG
1),offsetValueFixed(MGHENMG 2),offsetValueFixed(MGHENMG 3),offsetValueFixed(MGHENMG
4),offsetValueFixed(MGHENMG 5),offsetValueFixed(MGHENMG 6),offsetValueFixed(MGHENMG
7),offsetValueFixed(MGHENMG 8),offsetValueFixed(MGHENMG 9),offsetValueFixed(MGHENMG
10),offsetValueFixed(MGHENMG 11),offsetValueFixed(MGHENMG
12),offsetValueFixed(MGHENMG 13),offsetValueFixed(MGHENMG
14),offsetValueFixed(MGHENMG 15),offsetValueFixed(MGHENMG
16),offsetValueFixed(MGHENMG 17),offsetValueFixed(MGHENMG
18),offsetValueFixed(MGHENMG 19),offsetValueFixed(MGHENMG
20),offsetValueFixed(MGHENMG 21),offsetValueFixed(MGHENMG
22),offsetValueFixed(MGHENMG 23),offsetValueFixed(MGHENMG
24),offsetValueFixed(MGHENMG 25),offsetValueFixed(MGHENMG
26),offsetValueFixed(MGHENMG 27),offsetValueFixed(MGHENMG
28),offsetValueFixed(MGHENMG 29),offsetValueFixed(MGHENMG
30),offsetValueFixed(MGHENMG 31)"
```

*Figure D7.* ADT Omni MGHENMG parameters.

```
java  -Xmx7G -Djava.awt.headless=true  -cp /home/ubuntu/target/nova-1.2-SNAPSHOT.jar
com.infoblazer.gp.Application \
--description="MGHENMG AAT Prediction" \
--tournamentSize=4                     \
--logMetrics=true                      \
--allowTrivialPredictions=false        \
--useAdaptiveTraining=false            \
--maxPredictionGenerations=5           \
--maxPrediction=10                     \
--minPrediction=-10                    \
--regimes=2                            \
--gcFrequency=20                       \
--startTrain=150                       \
--endTrain=250                         \
--startTest=150                        \
--endTest=250                          \
--returnType=number                    \
--applicationName=linearRegressionApp  \
--elitist=true                         \
--programType=Prediction               \
--target=MGHENMG                       \
```

```
--direction=asc                       \
--populationSize=3000                 \
--regimePopulationSize=3000           \
--trainingWindow=110                  \
--maxInitDepth=5                      \
--maxDepth=10                         \
--mutationPct=10                      \
--crossoverPct=80                     \
--compressionPct=5                    \
--fourWayPct=25                       \
--minimumCompressionSize=5            \
--expansionPct=5                      \
--trainingGenerations=41              \
--selectionStrategy=tournamentSelectionStrategy \
--functions="add,subtract,multiply,divide,sin,cos,sqrt,exp,ln" \
--
regimeFunctions="add,subtract,and,not,gt,offsetValue,periodMinimum,periodMaximum,std
Dev,movingAverage" \
--terminals="randomInteger(-1 110),offsetValueFixed(MGHENMG
1),offsetValueFixed(MGHENMG 2),offsetValueFixed(MGHENMG 3),offsetValueFixed(MGHENMG
4),offsetValueFixed(MGHENMG 5),offsetValueFixed(MGHENMG 6),offsetValueFixed(MGHENMG
7),offsetValueFixed(MGHENMG 8),offsetValueFixed(MGHENMG 9),offsetValueFixed(MGHENMG
10),offsetValueFixed(MGHENMG 11),offsetValueFixed(MGHENMG
12),offsetValueFixed(MGHENMG 13),offsetValueFixed(MGHENMG
14),offsetValueFixed(MGHENMG 15),offsetValueFixed(MGHENMG
16),offsetValueFixed(MGHENMG 17),offsetValueFixed(MGHENMG
18),offsetValueFixed(MGHENMG 19),offsetValueFixed(MGHENMG
20),offsetValueFixed(MGHENMG 21),offsetValueFixed(MGHENMG
22),offsetValueFixed(MGHENMG 23),offsetValueFixed(MGHENMG
24),offsetValueFixed(MGHENMG 25),offsetValueFixed(MGHENMG
26),offsetValueFixed(MGHENMG 27),offsetValueFixed(MGHENMG
28),offsetValueFixed(MGHENMG 29),offsetValueFixed(MGHENMG
30),offsetValueFixed(MGHENMG 31)"
```

*Figure D8.* AAT MGHENMG parameters.

```
java  -Xmx7G -Djava.awt.headless=true  -cp /home/ubuntu/target/nova-1.2-SNAPSHOT.jar
com.infoblazer.gp.Application \
--description="MGHENMG AAT Prediction Omni" \
--tournamentSize=4                    \
--logMetrics=true                     \
--allowTrivialPredictions=false       \
--useAdaptiveTraining=false           \
--maxPredictionGenerations=5          \
--maxPrediction=10                    \
--minPrediction=-10                   \
--regimes=2                           \
--gcFrequency=20                      \
--startTrain=150                      \
--endTrain=250                        \
--startTest=150                       \
--endTest=250                         \
--returnType=number                   \
--applicationName=linearRegressionApp  \
--elitist=true                        \
--programType=Prediction              \
--target=MGHENMG                      \
--direction=asc                       \
--populationSize=3000                 \
```

```
--regimePopulationSize=1            \
--trainingWindow=110                   \
--maxInitDepth=5                       \
--maxDepth=10                          \
--mutationPct=10                       \
--crossoverPct=80                      \
--compressionPct=5                     \
--fourWayPct=25                        \
--minimumCompressionSize=5             \
--expansionPct=5                       \
--trainingGenerations=41               \
--selectionStrategy=tournamentSelectionStrategy \
--functions="add,subtract,multiply,divide,sin,cos,sqrt,exp,ln" \
--regimeFunctions="regimeMGHENMG"   \
--terminals="randomInteger(-1 110),offsetValueFixed(MGHENMG
1),offsetValueFixed(MGHENMG 2),offsetValueFixed(MGHENMG 3),offsetValueFixed(MGHENMG
4),offsetValueFixed(MGHENMG 5),offsetValueFixed(MGHENMG 6),offsetValueFixed(MGHENMG
7),offsetValueFixed(MGHENMG 8),offsetValueFixed(MGHENMG 9),offsetValueFixed(MGHENMG
10),offsetValueFixed(MGHENMG 11),offsetValueFixed(MGHENMG
12),offsetValueFixed(MGHENMG 13),offsetValueFixed(MGHENMG
14),offsetValueFixed(MGHENMG 15),offsetValueFixed(MGHENMG
16),offsetValueFixed(MGHENMG 17),offsetValueFixed(MGHENMG
18),offsetValueFixed(MGHENMG 19),offsetValueFixed(MGHENMG
20),offsetValueFixed(MGHENMG 21),offsetValueFixed(MGHENMG
22),offsetValueFixed(MGHENMG 23),offsetValueFixed(MGHENMG
24),offsetValueFixed(MGHENMG 25),offsetValueFixed(MGHENMG
26),offsetValueFixed(MGHENMG 27),offsetValueFixed(MGHENMG
28),offsetValueFixed(MGHENMG 29),offsetValueFixed(MGHENMG
30),offsetValueFixed(MGHENMG 31)"
```

*Figure D9.* AAT Omni MGHENMG parameters.

```
java  -Xmx7G -Djava.awt.headless=true  -cp /home/ubuntu/target/nova-1.2-SNAPSHOT.jar
com.infoblazer.gp.Application \
--description="MGHENMG Dyfor" \
--tournamentSize=4          \
--logMetrics=true                    \
--allowTrivialPredictions=false \
--maxPrediction=10            \
--minPrediction=-10           \
--startTrain=150              \
--endTrain=250                \
--startTest=150               \
--endTest=250                 \
--returnType=Number           \
--applicationName=linearRegressionDyforGp \
--programType=Prediction                 \
--populationSize=3000                    \
--maxInitDepth=5                         \
--maxDepth=10                            \
--mutationPct=10                         \
--crossoverPct=90                        \
--target=MGHENMG                         \
--trainingGenerations=41                 \
--selectionStrategy=tournamentSelectionStrategy \
--direction=asc                                \
--N=3                                          \
--saveoff=10                                   \
--maxWindowSize=200                            \
```

```
--predictionSize=1                              \
--minWindowSize=20                              \
--windowSlide=1                                 \
--windowDifference=20                           \
--startWindowSize=80                            \
--
functions="sin,cos,sqrt,exp,ln,add,subtract,multiply,divide,and,not,gt,offsetValue,p
eriodMinimum,periodMaximum,stdDev,movingAverage" \
--terminals="randomInteger(-1 110),offsetValueFixed(MGHENMG
1),offsetValueFixed(MGHENMG 2),offsetValueFixed(MGHENMG 3),offsetValueFixed(MGHENMG
4),offsetValueFixed(MGHENMG 5),offsetValueFixed(MGHENMG 6),offsetValueFixed(MGHENMG
7),offsetValueFixed(MGHENMG 8),offsetValueFixed(MGHENMG 9),offsetValueFixed(MGHENMG
10),offsetValueFixed(MGHENMG 11),offsetValueFixed(MGHENMG
12),offsetValueFixed(MGHENMG 13),offsetValueFixed(MGHENMG
14),offsetValueFixed(MGHENMG 15),offsetValueFixed(MGHENMG
16),offsetValueFixed(MGHENMG 17),offsetValueFixed(MGHENMG
18),offsetValueFixed(MGHENMG 19),offsetValueFixed(MGHENMG
20),offsetValueFixed(MGHENMG 21),offsetValueFixed(MGHENMG
22),offsetValueFixed(MGHENMG 23),offsetValueFixed(MGHENMG
24),offsetValueFixed(MGHENMG 25),offsetValueFixed(MGHENMG
26),offsetValueFixed(MGHENMG 27),offsetValueFixed(MGHENMG
28),offsetValueFixed(MGHENMG 29),offsetValueFixed(MGHENMG
30),offsetValueFixed(MGHENMG 31)"
```

*Figure D10.* DyFor GP MGHENMG parameters.

***SINCOS.***

```
java  -Xmx7G -Djava.awt.headless=true  -cp /home/ubuntu/target/nova-1.2-SNAPSHOT.jar
com.infoblazer.gp.Application \
--description="SINCOS GP"       \
--maxValidFitness=10000000000   \
--tournamentSize=4              \
--logMetrics=true             \
--visualize=false             \
--startTrain=0                \
--endTrain=100%               \
--startTest=0                 \
--endTest=100%                \
--applicationName=linearRegressionApp \
--elitist=true                      \
--programType=LinearRegression      \
--returnType=Number                 \
--target=SINE                       \
--direction=asc                     \
--populationSize=5000               \
--maxInitDepth=5                    \
--maxDepth=10                       \
--mutationPct=10                    \
--crossoverPct=90                   \
--trainingGenerations=100         \
--testingGenerations=1            \
--selectionStrategy=tournamentSelectionStrategy \
--functions="add,subtract,multiply,divide,sin,cos,sqrt,exp,ln" \
--terminals="randomInteger(-1 100),variable(x)"
```

*Figure D11.* GP SINCOS parameters.

```
java  -Xmx7G -Djava.awt.headless=true  -cp /home/ubuntu/target/nova-1.2-SNAPSHOT.jar
com.infoblazer.gp.Application \
--description="SINCOS ADF"       \
--maxValidFitness=10000000000   \
--tournamentSize=4              \
--adfArity="1,2"                \
--logMetrics=true              \
--visualize=false              \
--startTrain=0                 \
--endTrain=100%                \
--startTest=0                  \
--endTest=100%                 \
--applicationName=linearRegressionApp \
--elitist=true                       \
--programType=LinearRegression       \
--returnType=Number                  \
--target=SINE                        \
--direction=asc                      \
--populationSize=5000                \
--maxInitDepth=5                     \
--maxDepth=10                        \
--mutationPct=10                     \
--crossoverPct=90                    \
--trainingGenerations=100         \
--testingGenerations=1            \
--selectionStrategy=tournamentSelectionStrategy \
--functions="add,subtract,multiply,divide,sin,cos,sqrt,exp,ln" \
--terminals="randomInteger(-1 100),variable(x)"
```

*Figure D12.* ADF SINCOS Parameters

```
java  -Xmx7G -Djava.awt.headless=true  -cp /home/ubuntu/target/nova-1.2-SNAPSHOT.jar
com.infoblazer.gp.Application \
--description="SINCOS ADT"       \
--maxValidFitness=10000000000   \
--tournamentSize=4              \
--adfArity="1,2"                \
--regimes=2                     \
--logMetrics=true              \
--visualize=false              \
--startTrain=0                 \
--endTrain=100%                \
--startTest=0                  \
--endTest=100%                 \
--applicationName=linearRegressionApp \
--elitist=true                       \
--programType=LinearRegression       \
--returnType=Number                  \
--target=SINE                        \
--direction=asc                      \
--populationSize=5000                \
--maxInitDepth=5                     \
--maxDepth=10                        \
--mutationPct=10                     \
--crossoverPct=90                    \
--fourWayPct=25                    \
```

```
--trainingGenerations=100          \
--testingGenerations=1             \
--selectionStrategy=tournamentSelectionStrategy \
--functions="add,subtract,multiply,divide,sin,cos,sqrt,exp,ln" \
--
regimeFunctions="and,not,gt,offsetValue,periodMinimum,periodMaximum,stdDev,movingAve
rage" \
--terminals="randomInteger(-1 100),variable(x)"
```

*Figure D13.* ADT SINCOS parameters.

```
java  -Xmx7G -Djava.awt.headless=true  -cp /home/ubuntu/target/nova-1.2-SNAPSHOT.jar
com.infoblazer.gp.Application \
--description="SINCOS AAT" \
--maxValidFitness=10000000000 \
--tournamentSize=4           \
--regimes=2                  \
--gcFrequency=10             \
--logMetrics=true            \
--visualize=false            \
--startTrain=0               \
--endTrain=100%              \
--startTest=0                \
--endTest=100%               \
--applicationName=linearRegressionApp \
--elitist=true                       \
--programType=LinearRegression       \
--returnType=Number                  \
--target=SINE                        \
--direction=asc                      \
--populationSize=5000                \
--maxInitDepth=5                     \
--maxDepth=10                        \
--mutationPct=10                     \
--crossoverPct=80                    \
--compressionPct=5                   \
--minimumCompressionSize=5           \
--expansionPct=5                     \
--fourWayPct=25                      \
--trainingGenerations=100            \
--testingGenerations=1               \
--selectionStrategy=tournamentSelectionStrategy \
--functions="add,subtract,multiply,divide,sin,cos,sqrt,exp,ln" \
--
regimeFunctions="and,not,gt,offsetValue,periodMinimum,periodMaximum,stdDev,movingAve
rage" \
--terminals="randomInteger(-1 100),variable(x)"
```

*Figure D14.* AAT SINCOS parameters.

***S&P 500 market data prediction.***

This section provides sample parameters used in the S&P 500 prediction experiments described in this dissertation. The samples include parameter files for the prediction period 1999-2000 including transaction costs. 1999-2004 ADT and DyFor GP input files are also provided.

```
java  -Xmx7G -Djava.awt.headless=true  -cp /nova/automatically-defined-
templates/target/nova-1.3-SNAPSHOT.jar com.infoblazer.gp.Application ^
--description="SP500 ADF"  ^
--printTrainingProgram=false   ^
--logMetrics=true   ^
--visualize=false       ^
--adfArity="2,2"        ^
--applicationName=marketProfitApp   ^
--programType=Profit                ^
--seriesStart=1/1/1989              ^
--seriesEnd=12/31/2000              ^
--startTrain=1/1/1989               ^
--endTrain=12/31/1993               ^
--startTest=1/1/1994                ^
--endTest=12/31/1998                ^
--target=SP500                      ^
--predictors=SP500.m250.1           ^
--direction=desc                             ^
--populationSize=500                         ^
--trainingGenerations=100                    ^
--stagnationLimit=50                         ^
--testingGenerations=1                       ^
--predictionStep=1                           ^
--predictionGenerations=0                    ^
--maxInitDepth=5                             ^
--maxDepth=10                                ^
--maxSize=100                                ^
--mutationPct=40                             ^
--crossoverPct=50                            ^
--tournamentSize=2                           ^
--selectionStrategy=tournamentSelectionStrategy    ^
--
functions="add,subtract,multiply,divide,gt,lt,and,or,not,offsetValue,ifElseBoolean,m
ovingAverage,periodMaximum,periodMinimum,norm"   ^
--terminals="randomInteger(0 250),randomDouble(0
2),terminalTrue,terminalFalse,offsetValueFixed(SP500.m250.1 0)"     ^
--returnType=Boolean ^
--trainingStep=1     ^
--signals=1          ^
--elitist=true       ^
--riskFreeReturn=TREAS3M  ^
--transactionCost=0.5PCT
```

*Figure D15.* ADF S&P 500 prediction sample parameter file

```
java  -Xmx7G -Djava.awt.headless=true  -cp /nova/automatically-defined-
templates/target/nova-1.3-SNAPSHOT.jar com.infoblazer.gp.Application ^
--description="SP500 ADT"  ^
--printTrainingProgram=false   ^
--logMetrics=true   ^
--visualize=false       ^
--adfArity="2,2"        ^
--regimes=2             ^
--applicationName=marketProfitApp   ^
--programType=Profit                ^
--seriesStart=1/1/1989              ^
--seriesEnd=12/31/2000              ^
--startTrain=1/1/1989               ^
--endTrain=12/31/1993               ^
--startTest=1/1/1994                ^
--endTest=12/31/1998                ^
--target=SP500                      ^
--predictors=SP500.m250.1           ^
--direction=desc                             ^
--populationSize=500                         ^
--trainingGenerations=100                    ^
--stagnationLimit=50                         ^
--testingGenerations=1                       ^
--predictionStep=1                           ^
--predictionGenerations=0                    ^
--maxInitDepth=5                             ^
--maxDepth=10                                ^
--maxSize=100                                ^
--mutationPct=40                             ^
--crossoverPct=50                            ^
--tournamentSize=2                           ^
--selectionStrategy=tournamentSelectionStrategy    ^
--
functions="add,subtract,multiply,divide,gt,lt,and,or,not,offsetValue,ifElseBoolean,m
ovingAverage,periodMaximum,periodMinimum,norm"   ^
--terminals="randomInteger(0 250),randomDouble(0
2),terminalTrue,terminalFalse,offsetValueFixed(SP500.m250.1 0)"     ^
--returnType=Boolean ^
--trainingStep=1     ^
--signals=1          ^
--elitist=true       ^
--riskFreeReturn=TREAS3M  ^
--transactionCost=0.5PCT
```

*Figure D16.* ADT S&P 500 prediction sample parameter file

```
java  -Xmx7G -Djava.awt.headless=true  -cp /nova/automatically-defined-
templates/target/nova-1.3-SNAPSHOT.jar com.infoblazer.gp.Application ^
--description="SP500 GP"  ^
--printTrainingProgram=false   ^
--logMetrics=true   ^
--visualize=false       ^
--applicationName=marketProfitApp   ^
--programType=Profit                ^
--seriesStart=1/1/1989              ^
--seriesEnd=12/31/2000              ^
--startTrain=1/1/1989               ^
```

```
--endTrain=12/31/1993            ^
--startTest=1/1/1994             ^
--endTest=12/31/1998             ^
--target=SP500                   ^
--predictors=SP500.m250.1        ^
--direction=desc                          ^
--populationSize=500                      ^
--trainingGenerations=100                 ^
--stagnationLimit=50                      ^
--testingGenerations=1                    ^
--predictionStep=1                        ^
--predictionGenerations=0                 ^
--maxInitDepth=5                         ^
--maxDepth=10                             ^
--maxSize=100                             ^
--mutationPct=40                          ^
--crossoverPct=50                         ^
--tournamentSize=2                        ^
--selectionStrategy=tournamentSelectionStrategy    ^
--
functions="add,subtract,multiply,divide,gt,lt,and,or,not,offsetValue,ifElseBoolean,m
ovingAverage,periodMaximum,periodMinimum,norm"   ^
--terminals="randomInteger(0 250),randomDouble(0
2),terminalTrue,terminalFalse,offsetValueFixed(SP500.m250.1 0)"     ^
--returnType=Boolean ^
--trainingStep=1     ^
--signals=1          ^
--elitist=true       ^
--riskFreeReturn=TREAS3M  ^
--transactionCost=0.5PCT
```

*Figure D17.* GP S&P 500 prediction sample parameter file

```
java  -Xmx7G -Djava.awt.headless=true  -cp /nova/automatically-defined-
templates/target/nova-1.3-SNAPSHOT.jar com.infoblazer.gp.Application ^
--description="SP500 ADT Sliding" ^
--printTrainingProgram=false      ^
--logMetrics=true               ^
--visualize=false                 ^
--regimes=2                       ^
--adfArity="2,2"                  ^
--applicationName=marketProfitApp  ^
--programType=Profit              ^
--seriesStart=1/1/1989             ^
--seriesEnd=12/31/2004             ^
--startTrain=1/1/1989              ^
--endTrain=12/31/1998              ^
--target=SP500                    ^
--predictors=SP500.m250.1          ^
--direction=desc                          ^
--populationSize=500                      ^
--trainingGenerations=50                  ^
--testingGenerations=1                    ^
--predictionStep=5                        ^
--predictionGenerations=1                 ^
--maxInitDepth=5                          ^
--maxDepth=10                             ^
```

```
--maxSize=100                       ^
--mutationPct=40                    ^
--crossoverPct=50                   ^
--tournamentSize=2                  ^
--trainingWindow=250                ^
--selectionStrategy=tournamentSelectionStrategy ^
--
functions="add,subtract,multiply,divide,gt,lt,and,or,not,offsetValue,ifElseBoolean,m
ovingAverage,periodMaximum,periodMinimum,norm" ^
--terminals="randomInteger(0 250),randomDouble(0
2),terminalTrue,terminalFalse,offsetValueFixed(SP500.m250.1 0)"
^
--returnType=Boolean ^
--trainingStep=5     ^
--signals=1          ^
--elitist=true       ^
--riskFreeReturn=TREAS3M  ^
--transactionCost=0.5PCT
```

*Figure D18.* ADT sliding window S&P 500 prediction sample parameter file

```
java  -Xmx7G -Djava.awt.headless=true  -cp /nova/automatically-defined-
templates/target/nova-1.3-SNAPSHOT.jar com.infoblazer.gp.Application ^
--description="SP500 Dyfor"  ^
--printTrainingProgram=false  ^
--logMetrics=true             ^
--visualize=false               ^
--applicationName=marketProfitAppDyforGp  ^
--programType=Profit                      ^
--seriesStart=1/1/1989            ^
--seriesEnd=12/31/2004            ^
--startTrain=1/1/1989             ^
--endTrain=12/31/1998             ^
--target=SP500                          ^
--predictors=SP500.m250.1         ^
--direction=desc                    ^
--populationSize=500                ^
--trainingGenerations=50            ^
--testingGenerations=1              ^
--predictionStep=5                  ^
--predictionGenerations=1           ^
--maxInitDepth=5                    ^
--maxDepth=10                       ^
--maxSize=100                       ^
--mutationPct=40                    ^
--crossoverPct=50                   ^
--tournamentSize=2                  ^
--trainingWindow=250                ^
--selectionStrategy=tournamentSelectionStrategy ^
--
functions="add,subtract,multiply,divide,gt,lt,and,or,not,offsetValue,ifElseBoolean,m
ovingAverage,periodMaximum,periodMinimum,norm" ^
--terminals="randomInteger(0 250),randomDouble(0
2),terminalTrue,terminalFalse,offsetValueFixed(SP500.m250.1 0)"
^
--N=3 ^
--saveoff=10 ^
```

```
--maxWindowSize=375 ^
--predictionSize=1  ^
--minWindowSize=125 ^
--windowDifference=40 ^
--startWindowSize=250 ^
--returnType=Boolean  ^
--trainingStep=5      ^
--signals=1           ^
--elitist=true        ^
--riskFreeReturn=TREAS3M  ^
--transactionCost=0.5PCT
```

*Figure D19.* DyFor GP S&P 500 prediction sample parameter file

# Appendix E

## Best Symbolic Regression Programs

The following are samples of the best programs found in the symbolic regression SINCOS tests.

```
(-
  (*
    (SIN
      (SIN
        (SIN
          (COS
            (sqrt
              (-
                (sqrt x) x))))))
    (sqrt
      (-
        (sqrt x)
        (- x 39))))
  (LOG
    (LOG
      (+
        (*
          (EXP
            (*
              (- x 68)
              (sqrt x)))
          (EXP
            (*
              (-
                (+ 6 x) 76) x)))
        (SIN
          (+
            (SIN
              (% x x))
            (COS 76)))))))
```

*Figure E1*. Best Regression Program recorded by GP approach.

```
main->
(*
  (sqrt x)
  (COS
    (*
      (sqrt
        (EXP
          (%
            (COS
              (LOG x))
            (EXP
              (*
                (- x 71)
                (sqrt x))))))
```

```
       (LOG
         (+
           (% 71
             (EXP
               (* 5 x)))
           (-
             (*
               (sqrt x)
               (COS
                 (sqrt x)))
             (sqrt x)))))))
adf0->
(LOG
  (EXP
    (LOG
      (LOG arg0))))
adf1->
(*
  (LOG
    (EXP
      (*
        (+
          (COS arg0)
          (sqrt
            (LOG arg0)))
        (LOG
          (SIN arg0))))))
  (SIN arg0))
```

*Figure E2.* Best symbolic regression program recorded by ADF approach.

```
main->
(-
  (-
    (-
      (adf0
        (*
          (LOG
            (LOG
              (% 7 x)))
          (LOG x)))
      (*
        (sqrt
          (-
            (- x
              (sqrt 8))
            (SIN
              (- x
                (sqrt 8)))))
        (adf0
          (LOG
            (EXP
              (- x
                (sqrt x)))))))
    (SIN
      (+
        (LOG x) x)))
  (adf0
    (*
      (*
```

```
        (+
          (* x x)
          (+
            (-
              (* 16
                (- 85 43))
              (sqrt x))
            (SIN
              (sqrt x))))
        (SIN
          (sqrt 8)))
      (sqrt x))))

adf0->
(sqrt
  (LOG
    (LOG 0)))

adf1->
(-
  (EXP
    (+
      (-
        (EXP
          (+ 1
            (SIN
              (+ arg0 arg0))))
        (EXP arg1))
      (*
        (EXP
          (EXP 0)) 1)))
  (EXP
    (EXP arg1)))

main->
(BinaryNumber
  (>
    (adf0 0)
    (adf0 x)))

adf0->
(SIN
  (SIN
    (COS
      (stdev
        (movingAverage
          (periodMinimum SINE
            (stdev
              (periodMinimum SINE
                (PeriodMaximum SINE arg0)))))))))

adf1->
(% 0
  (- 0
    (periodMinimum SINE
      (offsetValue SINE
        (PeriodMaximum SINE
          (PeriodMaximum SINE arg0))))))
```

*Figure E3*. Best symbolic regression program recorded by ADT approach.

```
main->
(*
  (AAT13480
    [1 :
     (SIN
       (+
         (*
           (SIN 59)
           (+
             (COS 11) x))
         (*
           (SIN 59)
           (%
             (- x
               (sqrt
                 (+ x 13))) 13))))]
    )
  (LOG
    (*
      (-
        (-
          (%
            (+
              (COS 11) x)
            (SIN
              (*
                (SIN 59)
                (% x 13)))) 74)
        (sqrt 59))
      (%
        (+
          (COS
            (SIN
              (EXP
                (+ x 31))))
          (- 80
            (- x 74)))
        (SIN
          (SIN
            (COS
              (LOG x))))))))


main->
(BinaryNumber true)
```

*Figure E4*. Best symbolic regression program recorded by AAT approach.

```
main->
(adf1 x
  (+
    (%
      (%
        (COS 2)
        (adf1
          (COS
```

```
            (-
              (sqrt x)
              (- x
                (adf1 x 46)))) x))
      (adf1
        (-
          (EXP
            (+
              (sqrt 25)
              (EXP 47)))
          (COS
            (-
              (sqrt x)
              (- x
                (adf1 x x))))) 46))
    (sqrt
      (adf1 x
        (+
          (%
            (COS
              (COS
                (sqrt x)))
            (adf1 x x))
          (sqrt
            (LOG x)))))))

adf0->
(COS
  (+
    (+
      (LOG 1)
      (LOG
        (+ 1 arg0)))
    (+
      (+
        (LOG
          (+
            (LOG 1)
            (LOG
              (+ 0 1))))
        (LOG
          (+ 0
            (+ 0 1)))) 0)))

adf1->
(+
  (LOG
    (+
      (* arg1 arg1)
      (SIN arg0)))
  (-
    (+
      (* arg1 arg1)
      (SIN
        (sqrt arg0)))
    (LOG
      (+ arg1
        (-
          (+
            (+
              (* arg1 arg1)
```

```
              (SIN
                (sqrt arg0)))
            (SIN
              (LOG
                (* arg1 arg1))))
          (sqrt arg0))))))

main->
(BinaryNumber
  (>
    (stdev
      (stdev
        (PeriodMaximum SINE
          (stdev
            (stdev
              (PeriodMaximum SINE
                (periodMinimum SINE x)))))))
    (movingAverage
      (periodMinimum SINE
        (PeriodMaximum SINE
          (PeriodMaximum SINE x))))))

adf0->
(EXP 1)

adf1->
(%
  (offsetValue SINE
    (movingAverage
      (offsetValue SINE 1)))
  (offsetValue SINE 0))
```

*Figure E5.* Best symbolic regression program recorded by coupled ADT approach.

```
main->
(*
  (SIN
    (COS
      (+
        (COS
          (sqrt
            (+
              (+
                (LOG 89) x) 54)))
        (+
          (COS
            (sqrt
              (+ x 54)))
          (LOG
            (LOG 37))))))
  (+
    (+
      (AAT6424
        [1 :
         (LOG 89)]
        [2 :
         (LOG x)]
        )
      (+
```

```
        (COS
          (sqrt
            (+ x
              (+ 75
                (sqrt x)))))
        (+
          (COS
            (sqrt
              (+ x 75)))
          (LOG
            (LOG
              (+
                (SIN 54) x))))))
    (SIN
      (+
        (LOG
          (*
            (*
              (- 90 x)
              (COS
                (% x 54)))
            (LOG
              (SIN
                (% 13 75)))))
        (COS
          (+
            (SIN
              (% 13 37))
            (+
              (+
                (SIN 54)
                (sqrt x))
              (COS
                (% x 54))))))))

main->
(BinaryNumber
  (>
    (offsetValue SINE
      (offsetValue SINE
        (PeriodMaximum SINE 16)))
    (periodMinimum SINE
      (stdev
        (offsetValue SINE
          (periodMinimum SINE 23))))))
```

*Figure E6*. Best symbolic regression program recorded by coupled AAT approach.

# Appendix F

## MGHENMG Initial Random Parameters

The MGHENMG series requires 31 offset values in order to determine y(x). Following the procedure described in (Wagner & Michalewicz, 2008), an array of length 1200 was initialized. The first 31 values were randomly generated and are shown in Table F1. The remainder of the series was generated using the formula shown in (19. The final 200 points of this series was used as the initial 200-point segment of the MGHENMG series. The remaining two sections were calculated directly from the formulas and prior series values.

Table F1. MGHENMG Initialization Parameters

| X | Y | X | Y |
|---|---|---|---|
| 1 | 0.8440964138066118 | 24 | 0.47020747316392664 |
| 2 | 0.9532661055224593 | 25 | 0.5455742442961969 |
| 3 | 0.06636753866027945 | 26 | 0.859267154912505 |
| 4 | 0.8620971393379627 | 27 | 0.5342809689449174 |
| 5 | 0.2100850778694464 | 28 | 0.3258840447272048 |
| 6 | 0.4894196903422636 | 29 | 0.5628455094036957 |
| 7 | 0.11588139131258257 | 30 | 0.46854285555526787 |
| 8 | 0.47264417635200906 | 31 | 0.6234535818110518 |
| 9 | 0.08408151444136325 | | |
| 10 | 0.5657337444733351 | | |
| 11 | 0.695537452821834 | | |
| 12 | 0.6371301993098292 | | |
| 13 | 0.11797947216718663 | | |
| 14 | 0.6643411155833825 | | |
| 15 | 0.35853611038911015 | | |
| 16 | 0.8617986087216263 | | |
| 17 | 0.028791106927193777 | | |
| 18 | 0.8136196388158929 | | |
| 19 | 0.01620324342793167 | | |
| 20 | 0.6569433423303102 | | |
| 21 | 0.2871394142441621 | | |
| 22 | 0.10838147285823774 | | |
| 23 | 0.43001491262576674 | | |

# Appendix G

## Full Results

This section provides charted results from the experiments described in Chapter 4. Information on downloading the raw data in Microsoft Access format is given in Appendix I.

### SINCOS.

Average fitness is provided in Figure 39 as is not reproduced in this section.

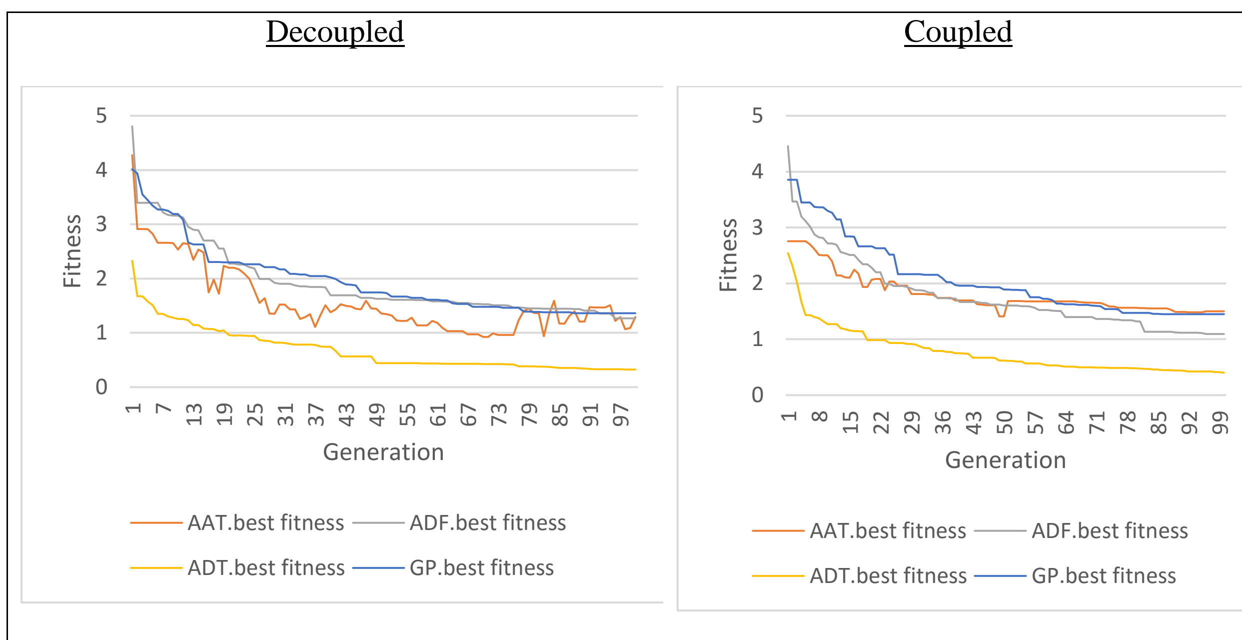
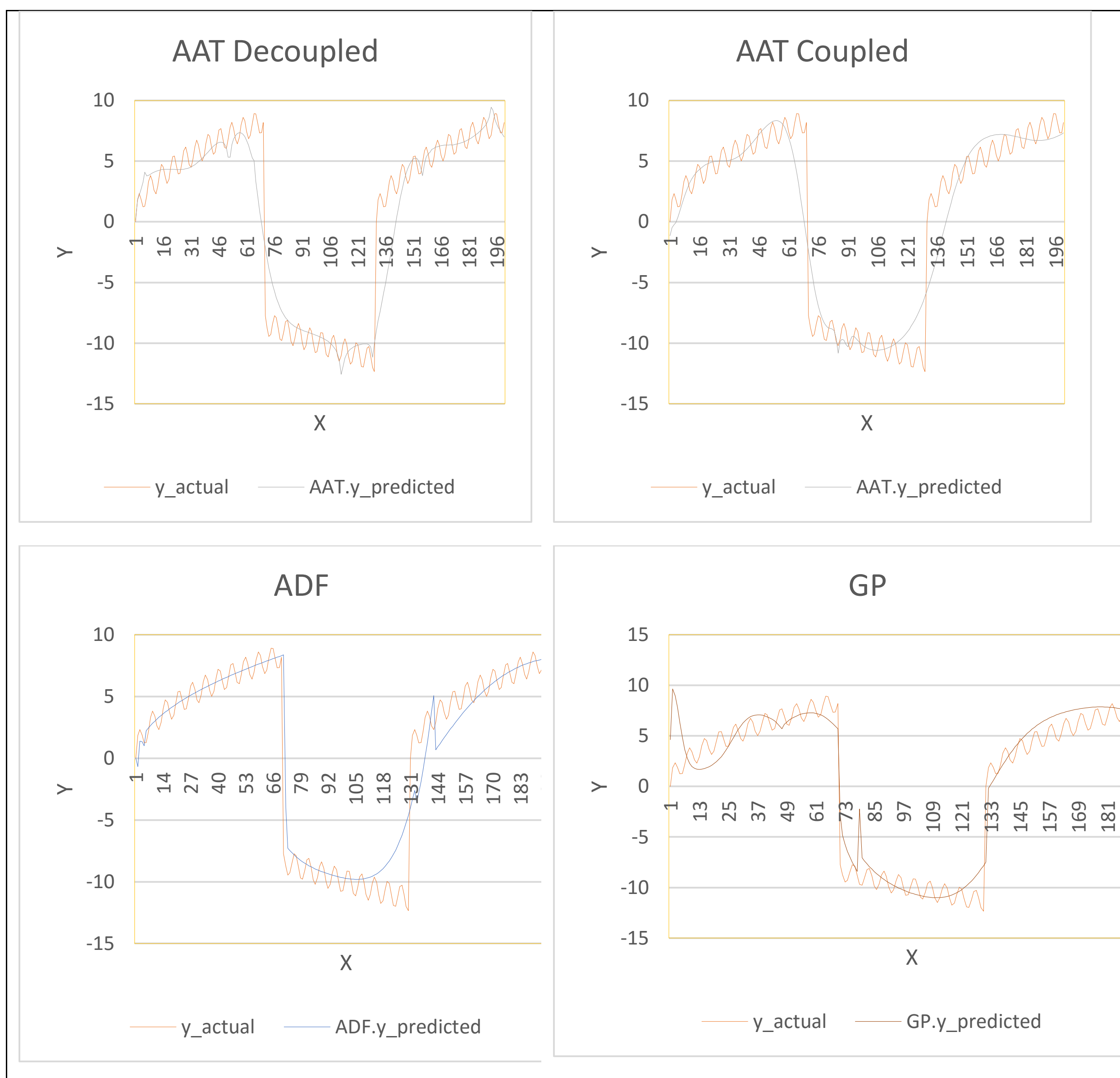



*Figure G1*. Best fitness in symbolic regression SINCOS experiments.


AAT Decoupled
10
5
0
-5
-10
-15
Y
1 16 31 46 61 76 91 106 121 136 151 166 181 196
X
y_actual
AAT.y_predicted
AAT Coupled
10
5
0
-5
-10
-15
Y
1 16 31 46 61 76 91 106 121 136 151 166 181 196
X
y_actual
AAT.y_predicted
ADF
10
5
0
-5
-10
-15
Y
1 14 27 40 53 66 79 92 105 118 131 144 157 170 183
X
y_actual
ADF.y_predicted
GP
15
10
5
0
-5
-10
-15
Y
1 13 25 37 49 61 73 85 97 109 121 133 145 157 169 181
X
y_actual
GP.y_predicted

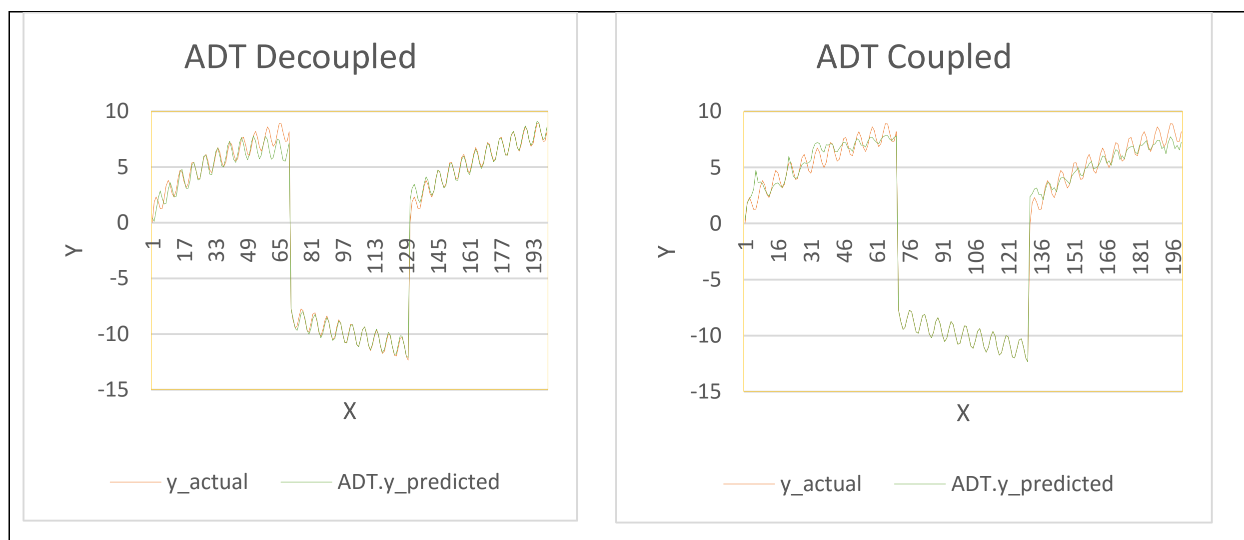



*Figure G2.* Best regressions in SINCOS experiments. Coupling does not affect GP and ADF.

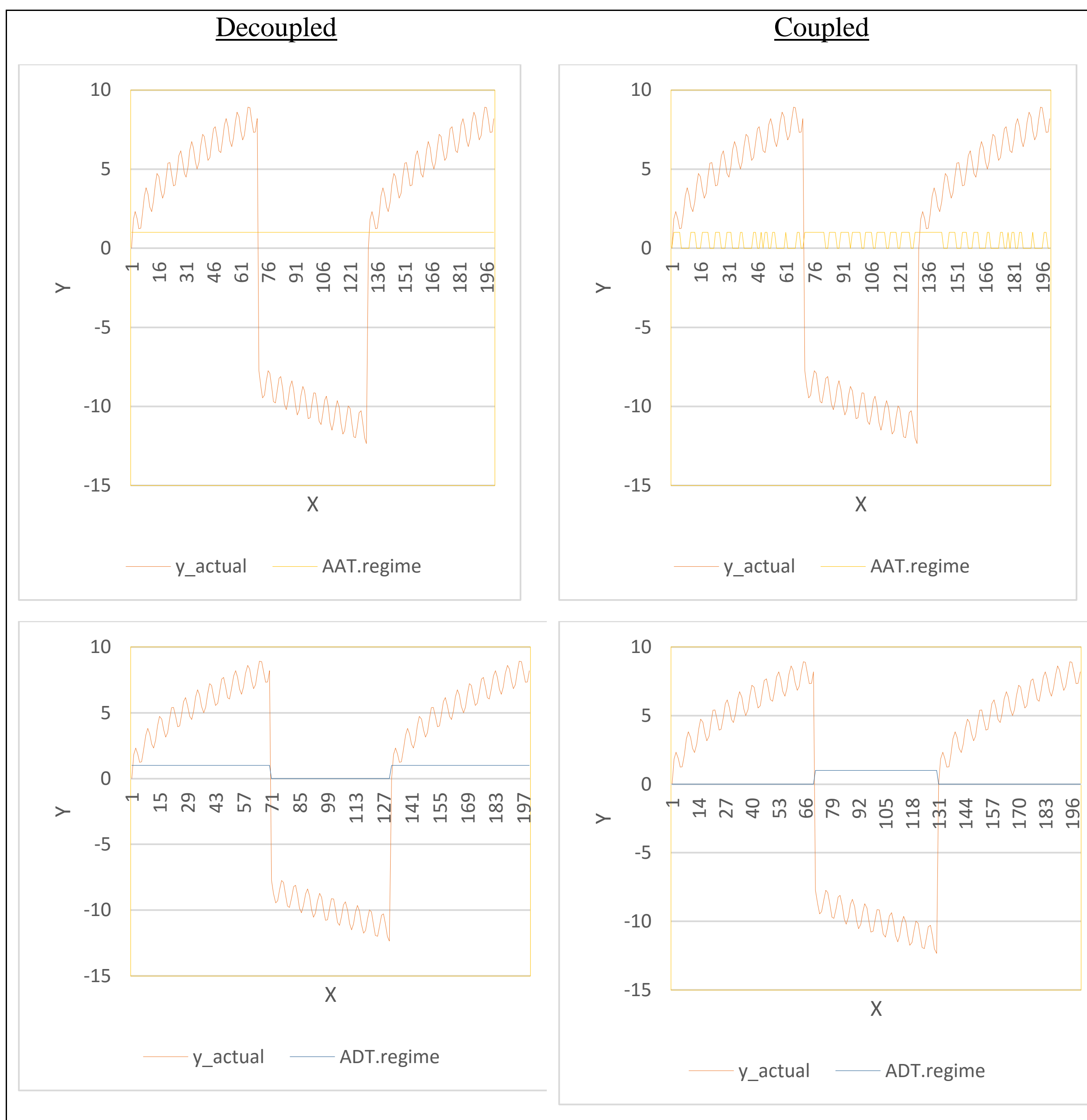



*Figure G3*. Best regime regressions in SINCOS experiments. ADT successfully determined the likely regime in both cases. The difference in numeric regime value is not relevant.

**LGOZLG.**

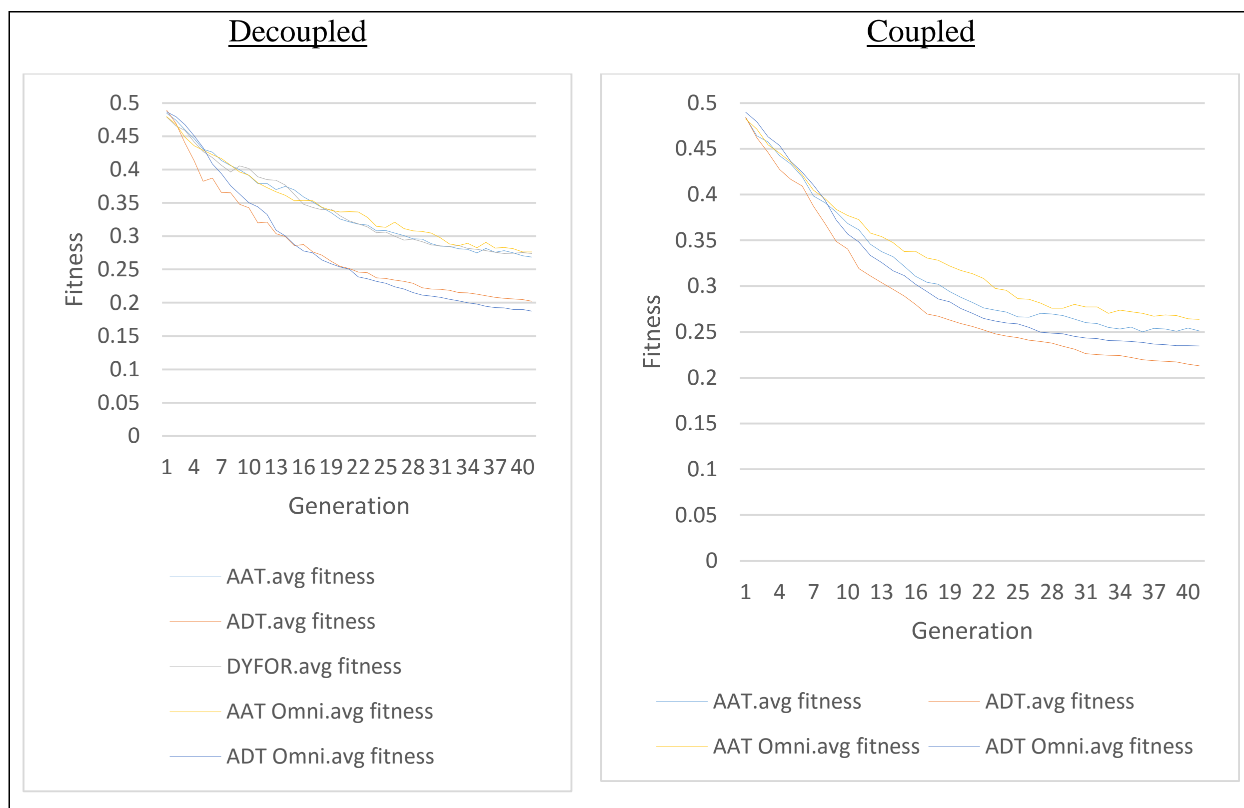



*Figure G4.* Average training fitness in LGOZLG experiments.

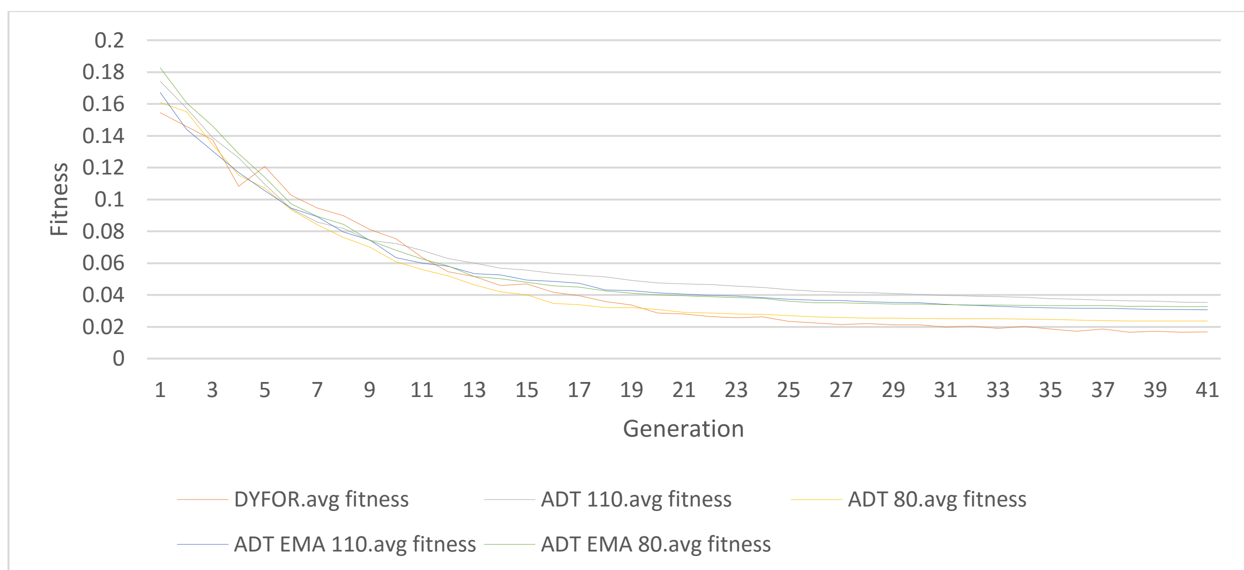
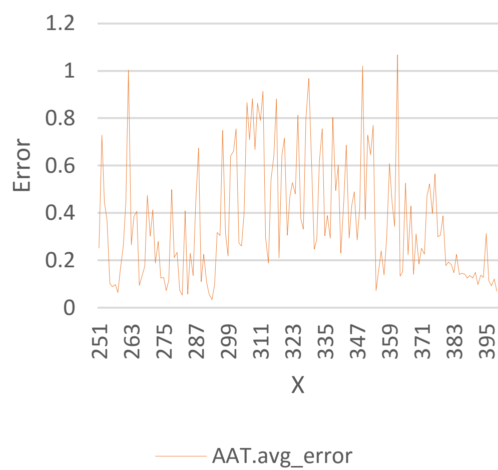
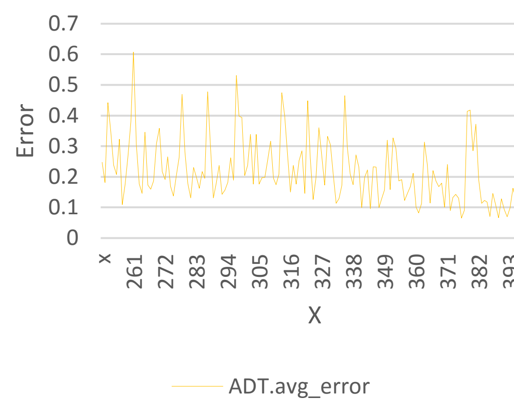
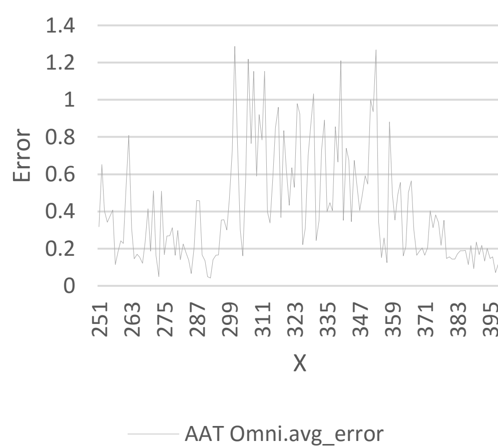
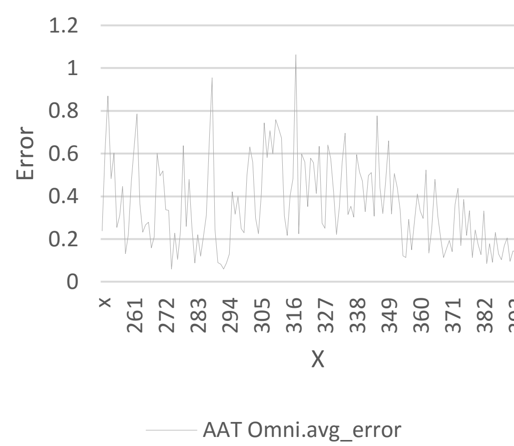



*Figure G5.* Average training fitness in LGOZLG 1-100 experiments.

| Decoupled | Coupled |
|---|---|
|   | |
| |   |
|   |   |

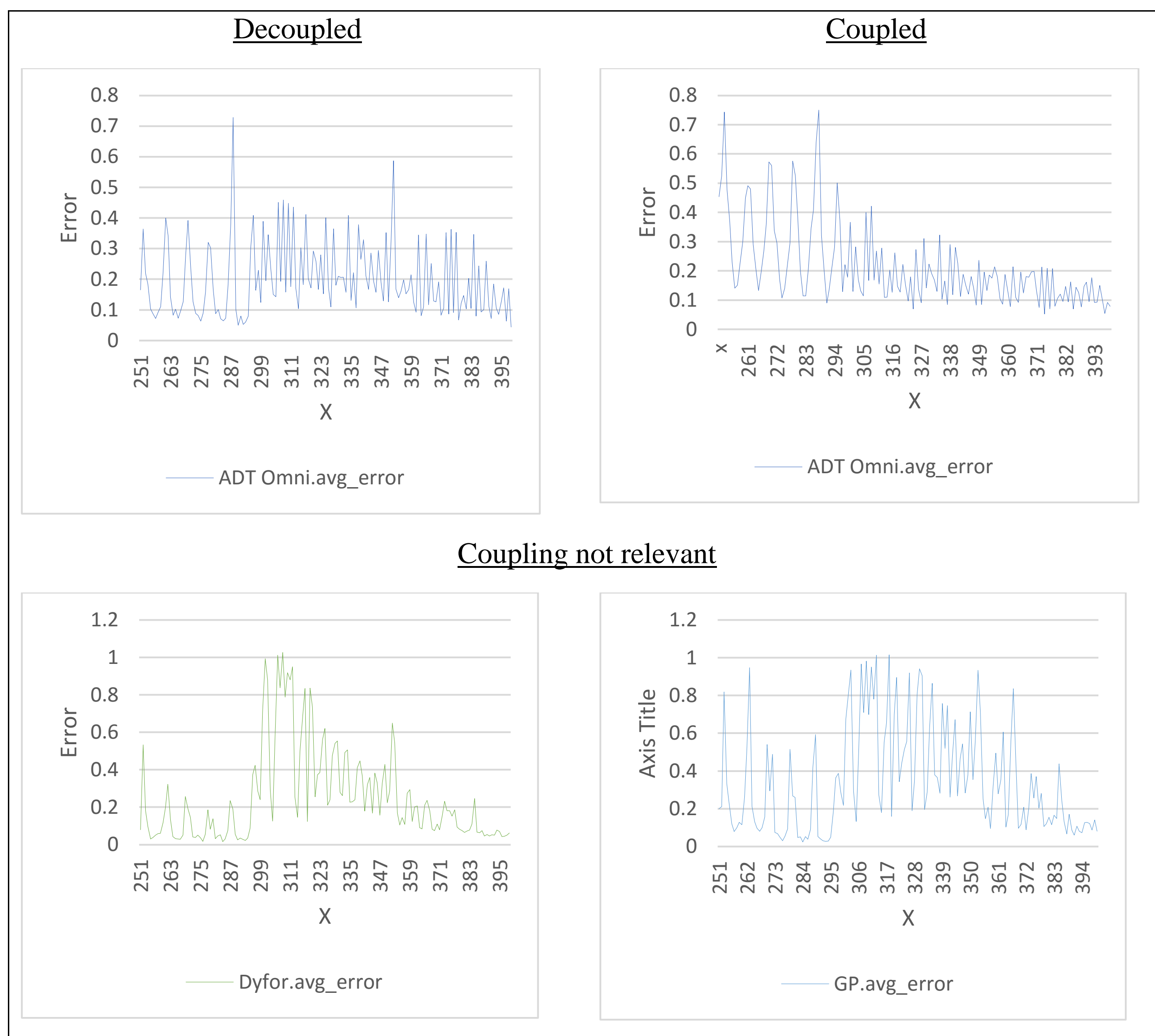



*Figure G6.* Average error in LGOZLG experiments. Coupling is not relevant for DyFor GP and GP results.

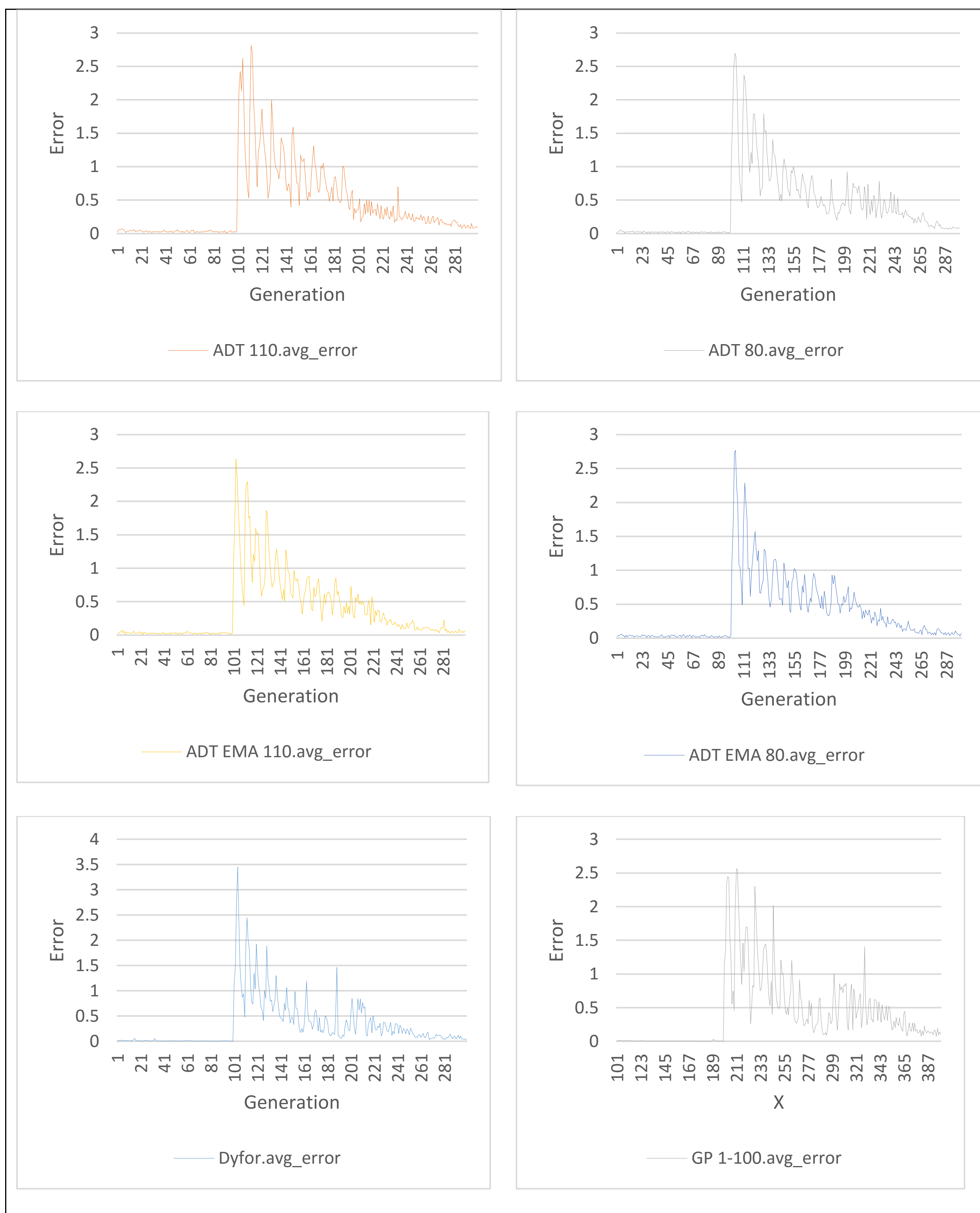



*Figure G7.* Average error in LGOZLG 1-100 experiments.

| Decoupled | Coupled |
|---|---|

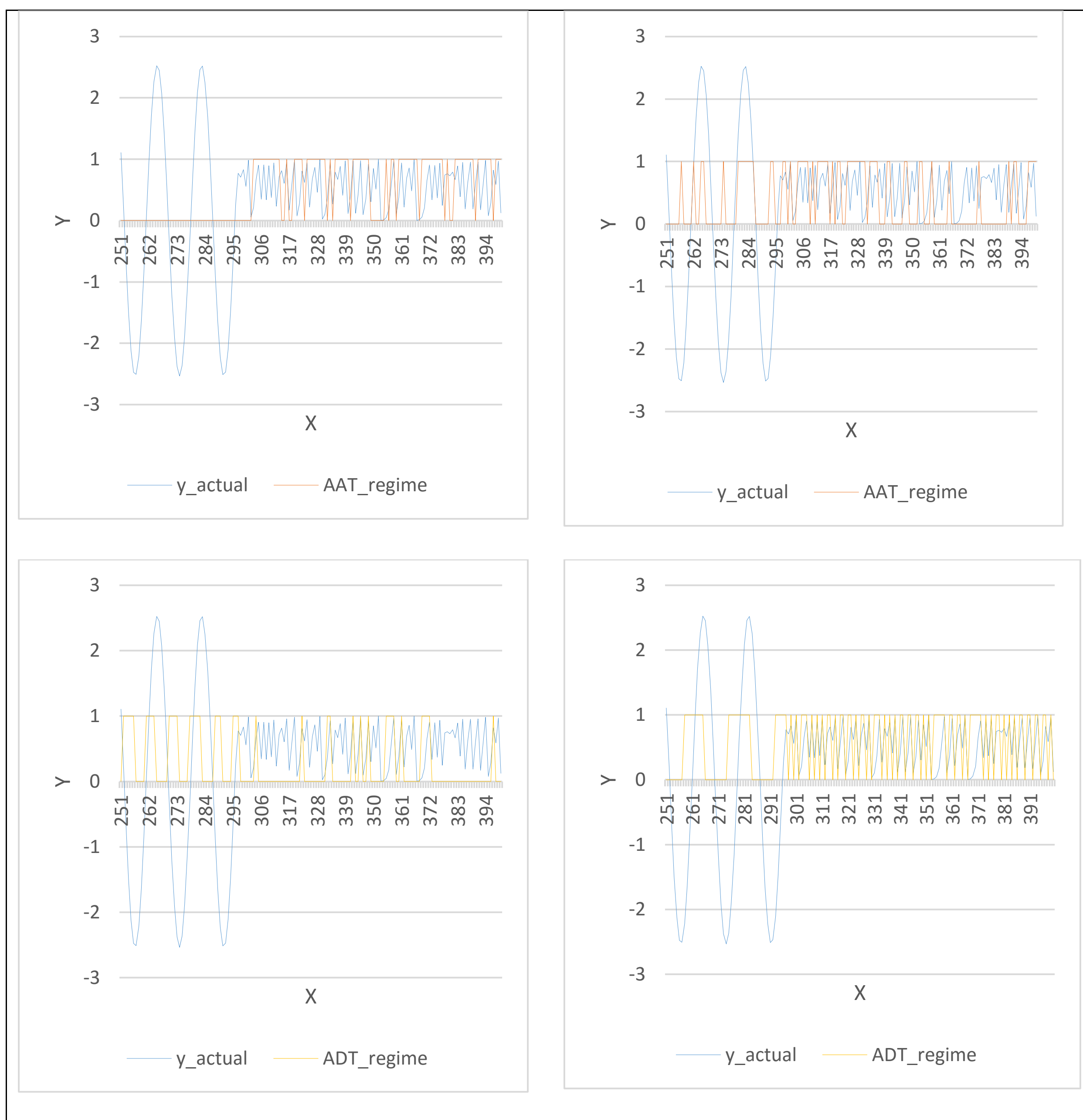



*Figure G8.* LGOZLG regime determination by best performing individuals.

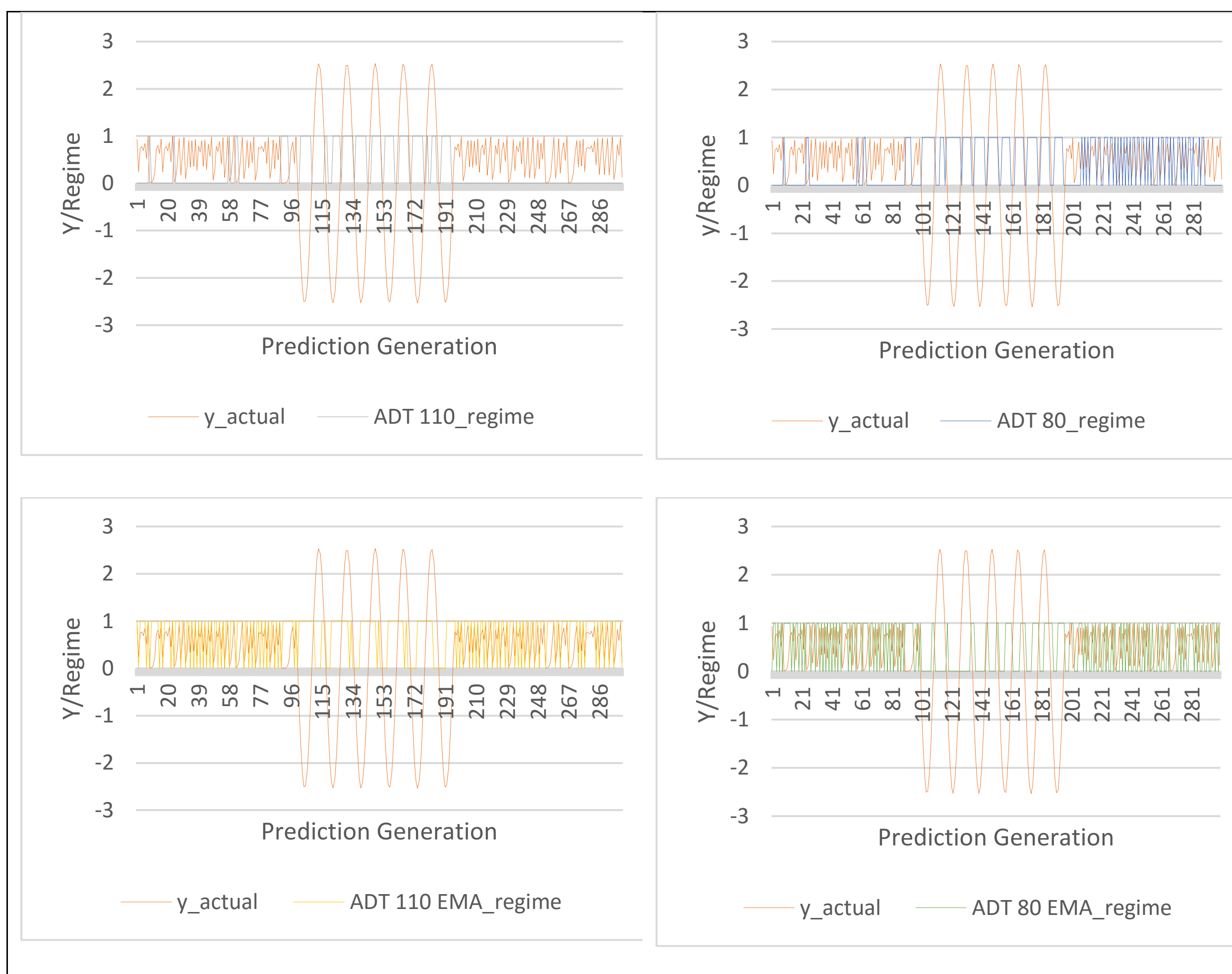
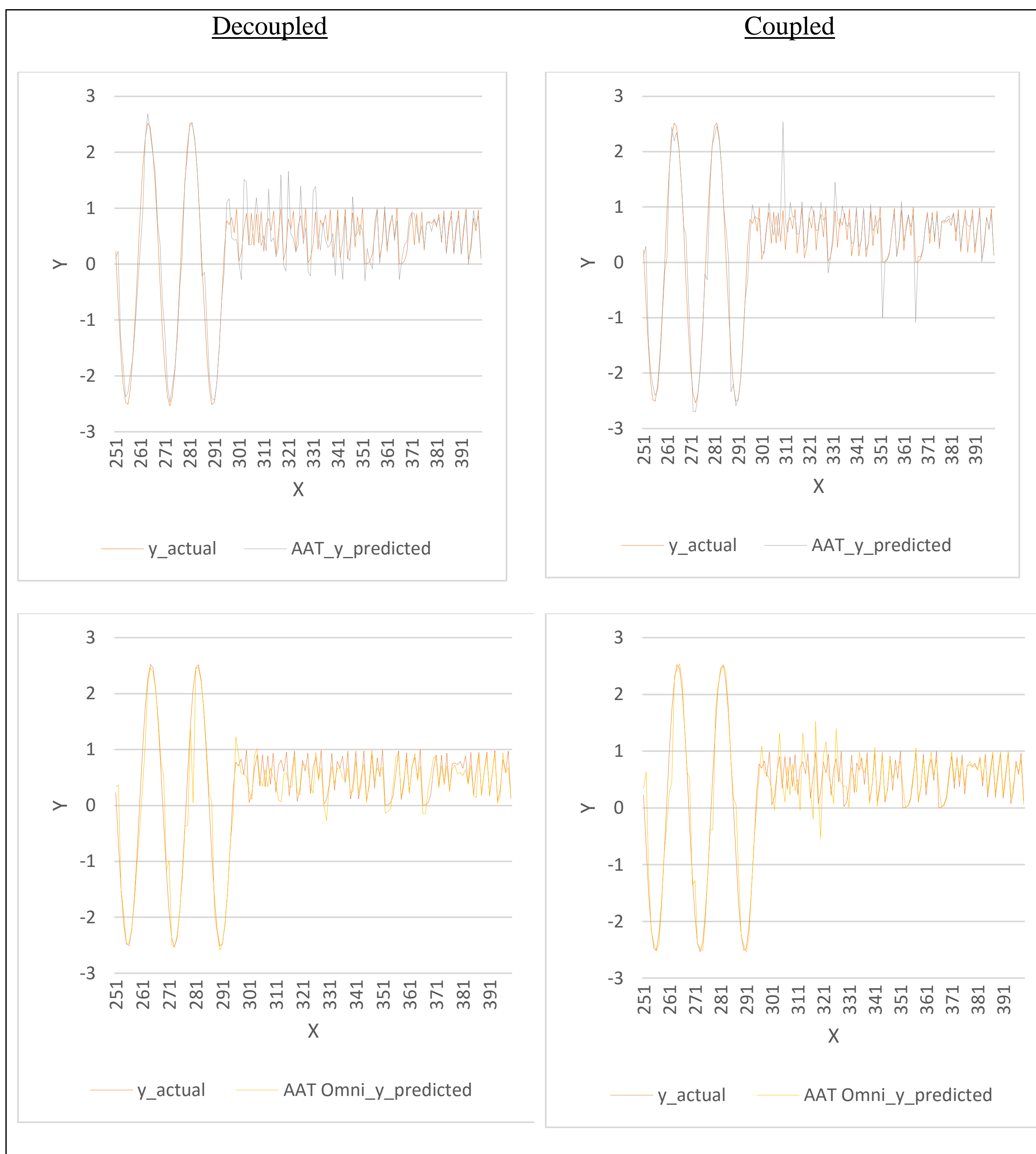
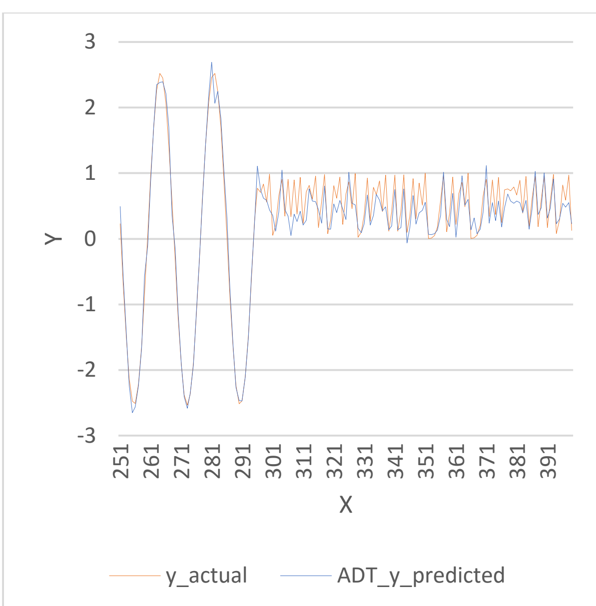
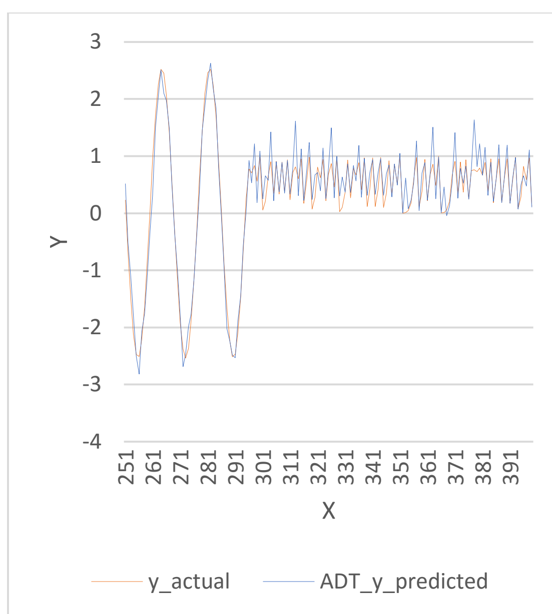


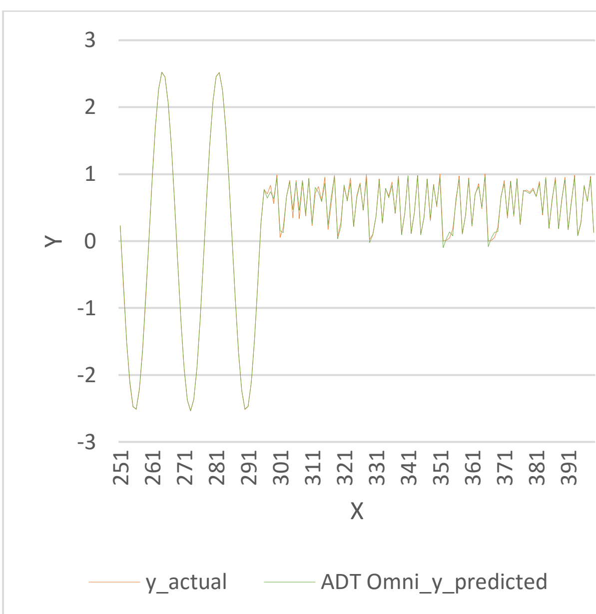

*Figure G9.* LGOZLG 1-100 regime determination by best performing individuals.

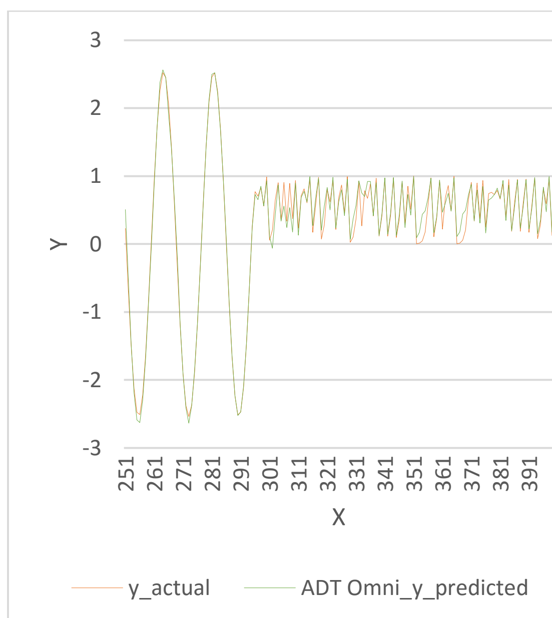

Decoupled
Coupled
3
2
1
0
-1
-2
-3
Y
X
251
261
271
281
291
301
311
321
331
341
351
361
371
381
391
y_actual
AAT_y_predicted
AAT Omni_y_predicted

**Decoupled** | **Coupled**



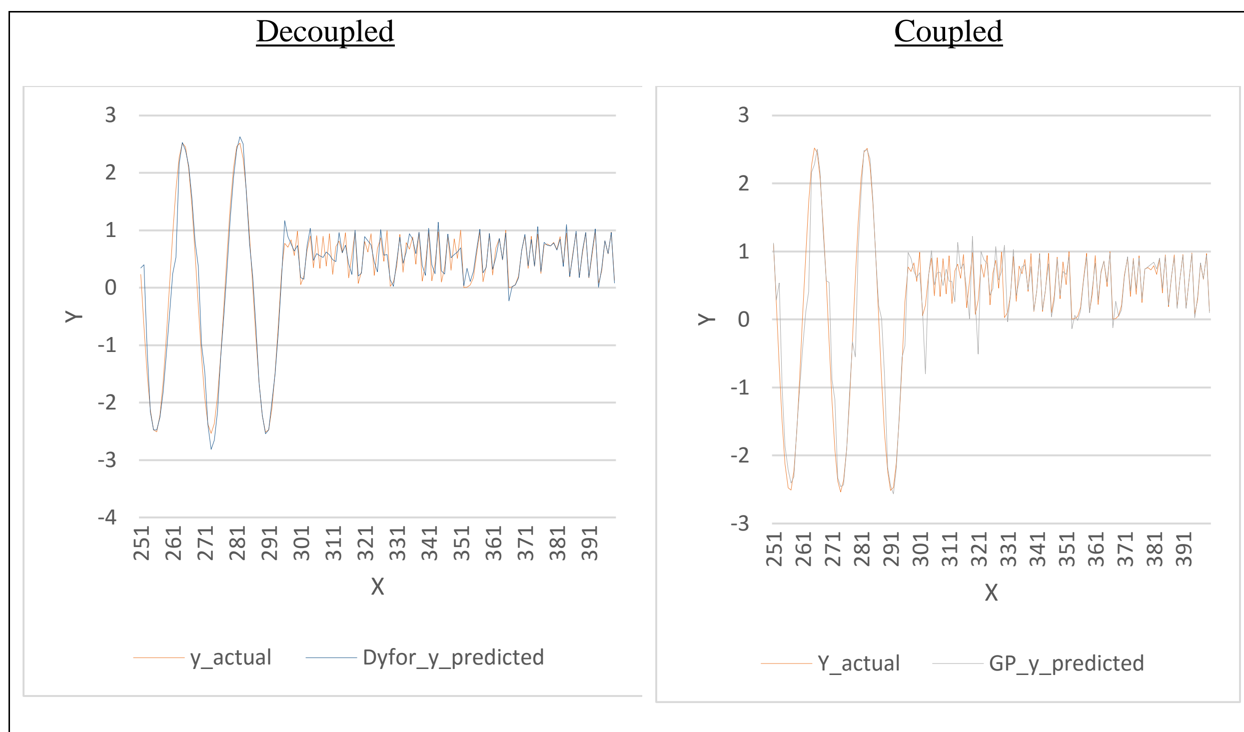
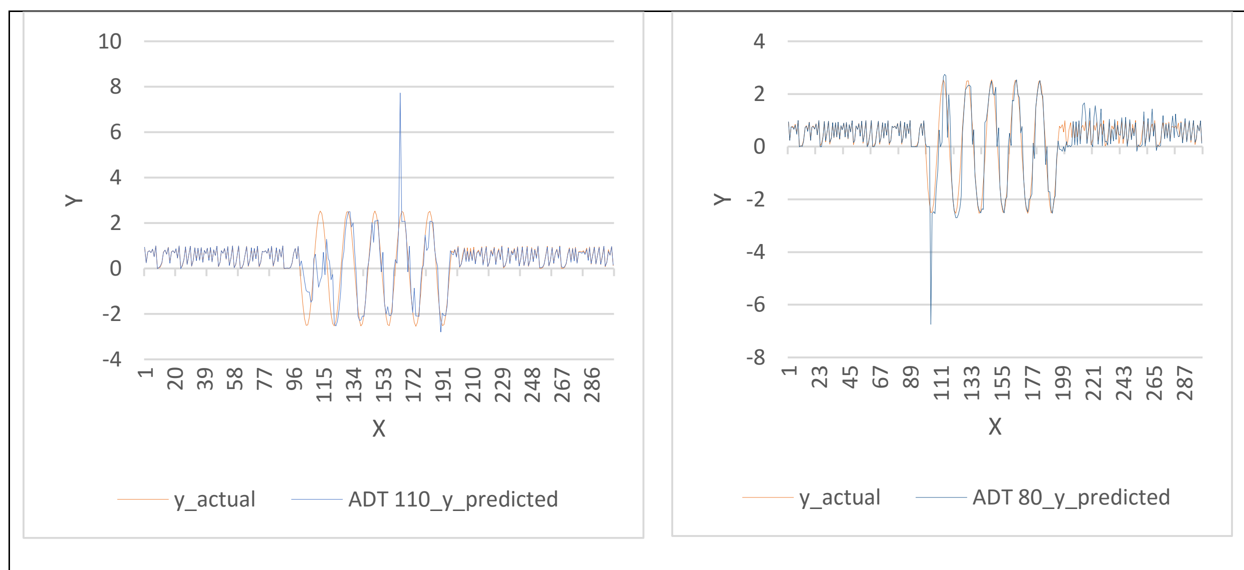



*Figure G10.* LGOZLG best prediction.



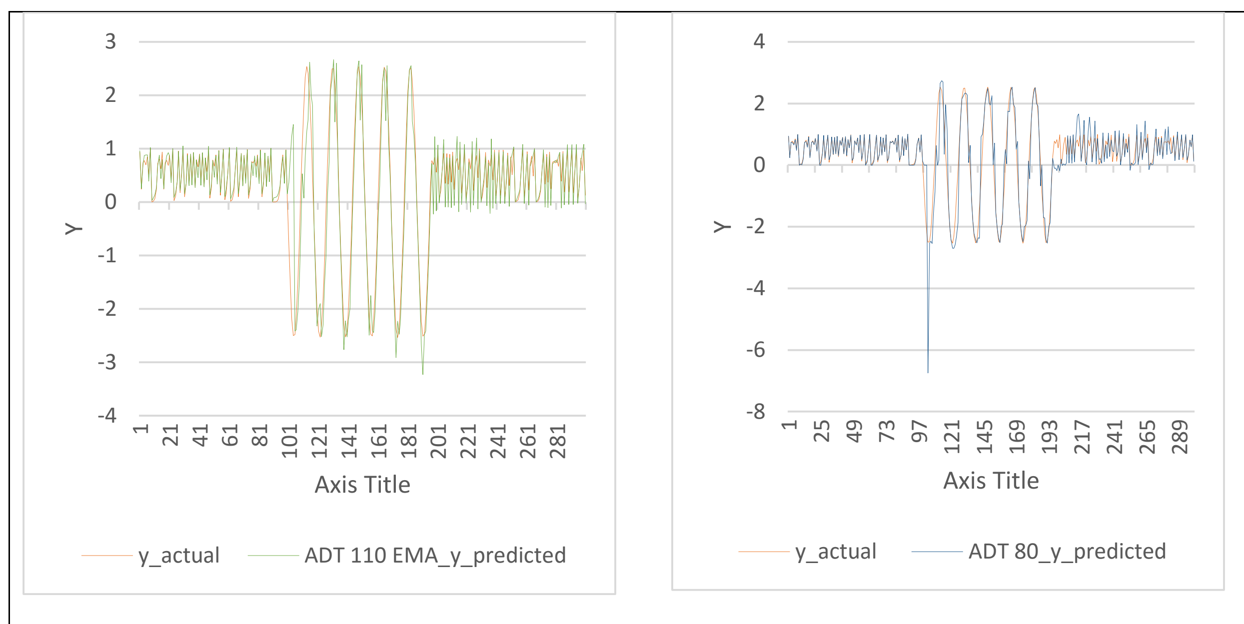



*Figure G11*. LGOZLG 1-100 best prediction.

**MGHENMG.**

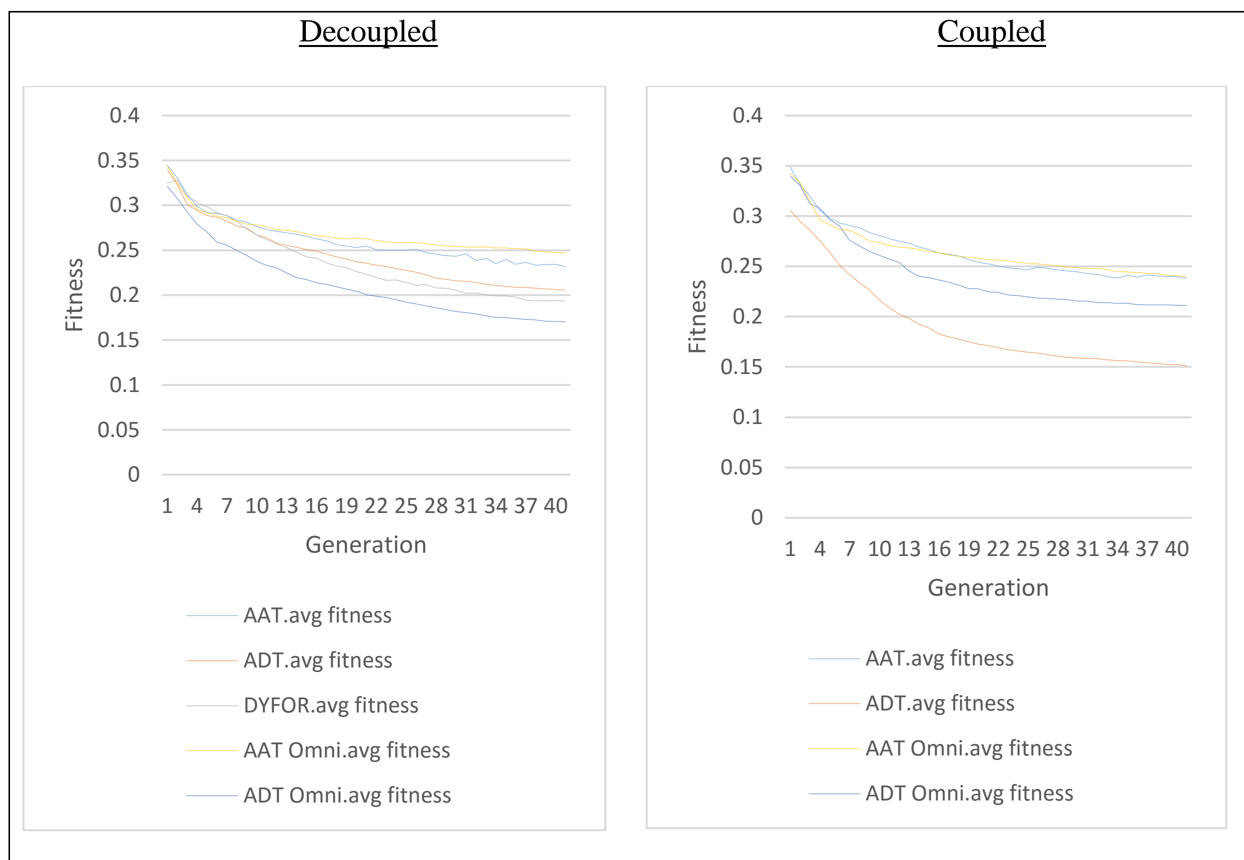



*Figure G12*. MGHENMG average training fitness.

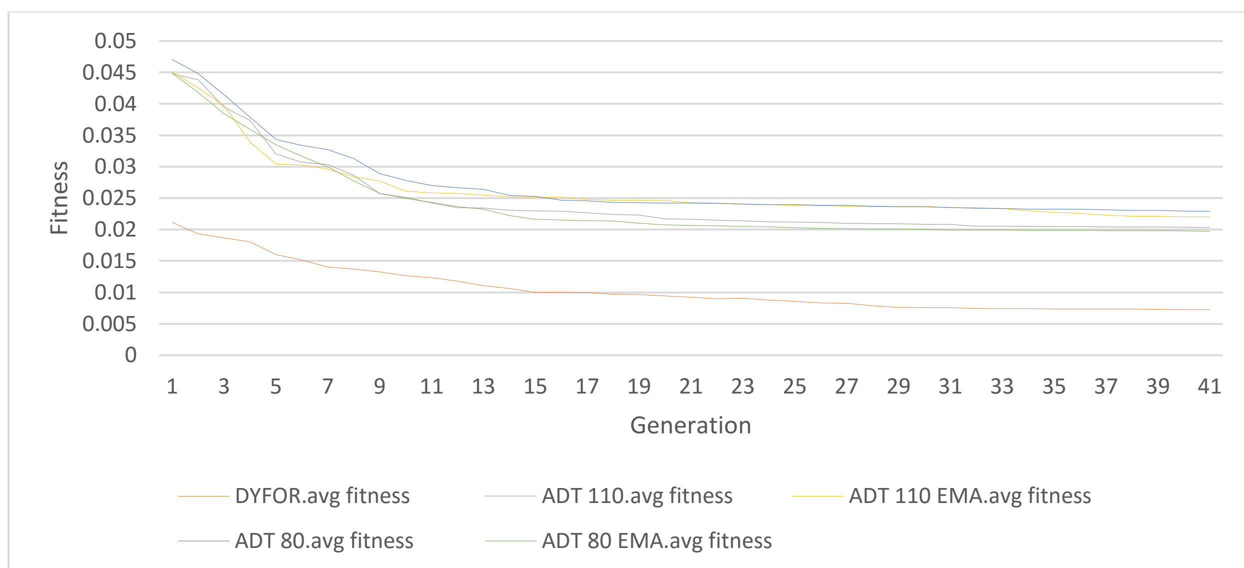



*Figure G13*. MGHENMG 30-130 average training fitness.

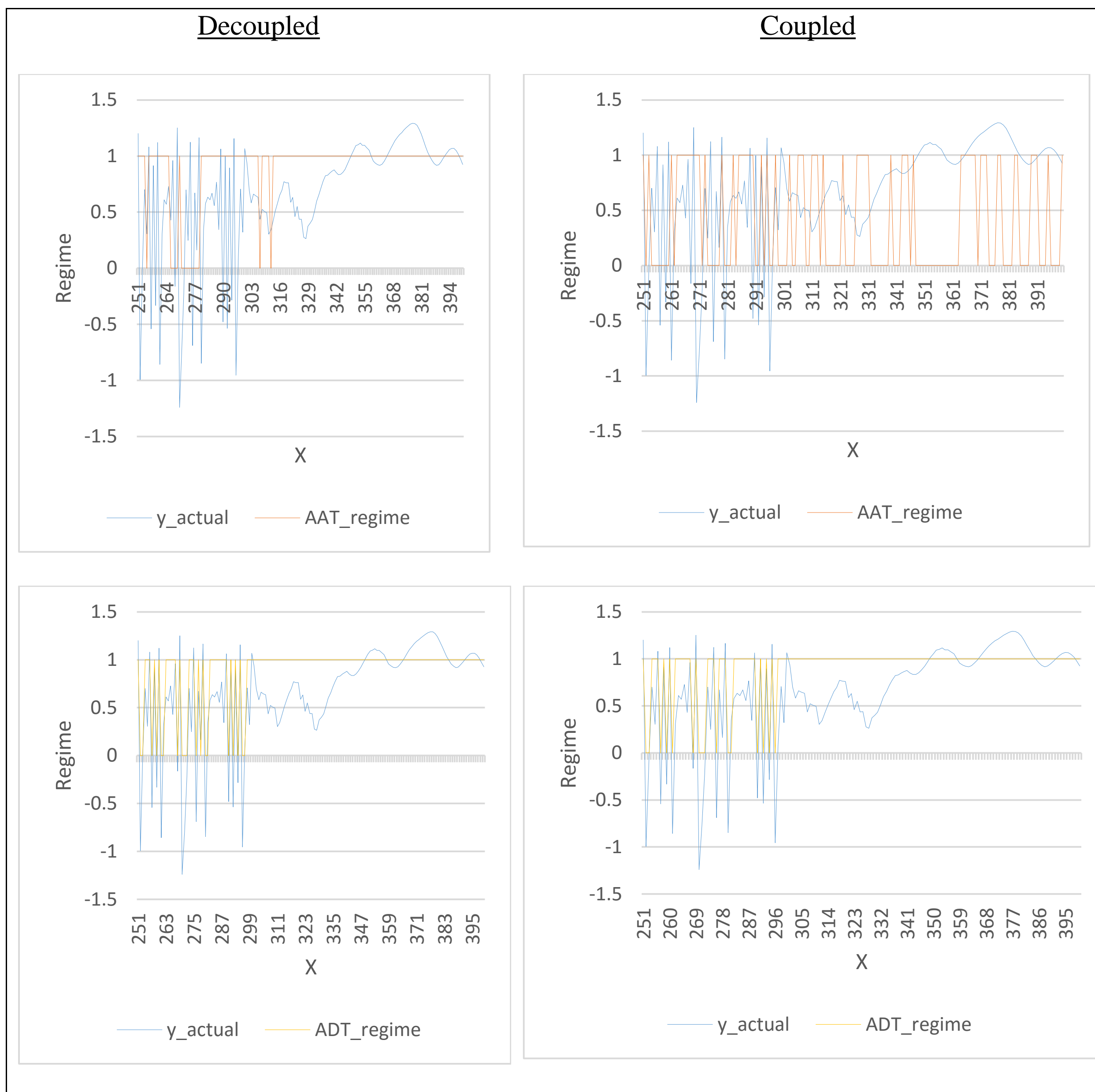



*Figure G14.* MGHENMG regime determination by best performing individuals.

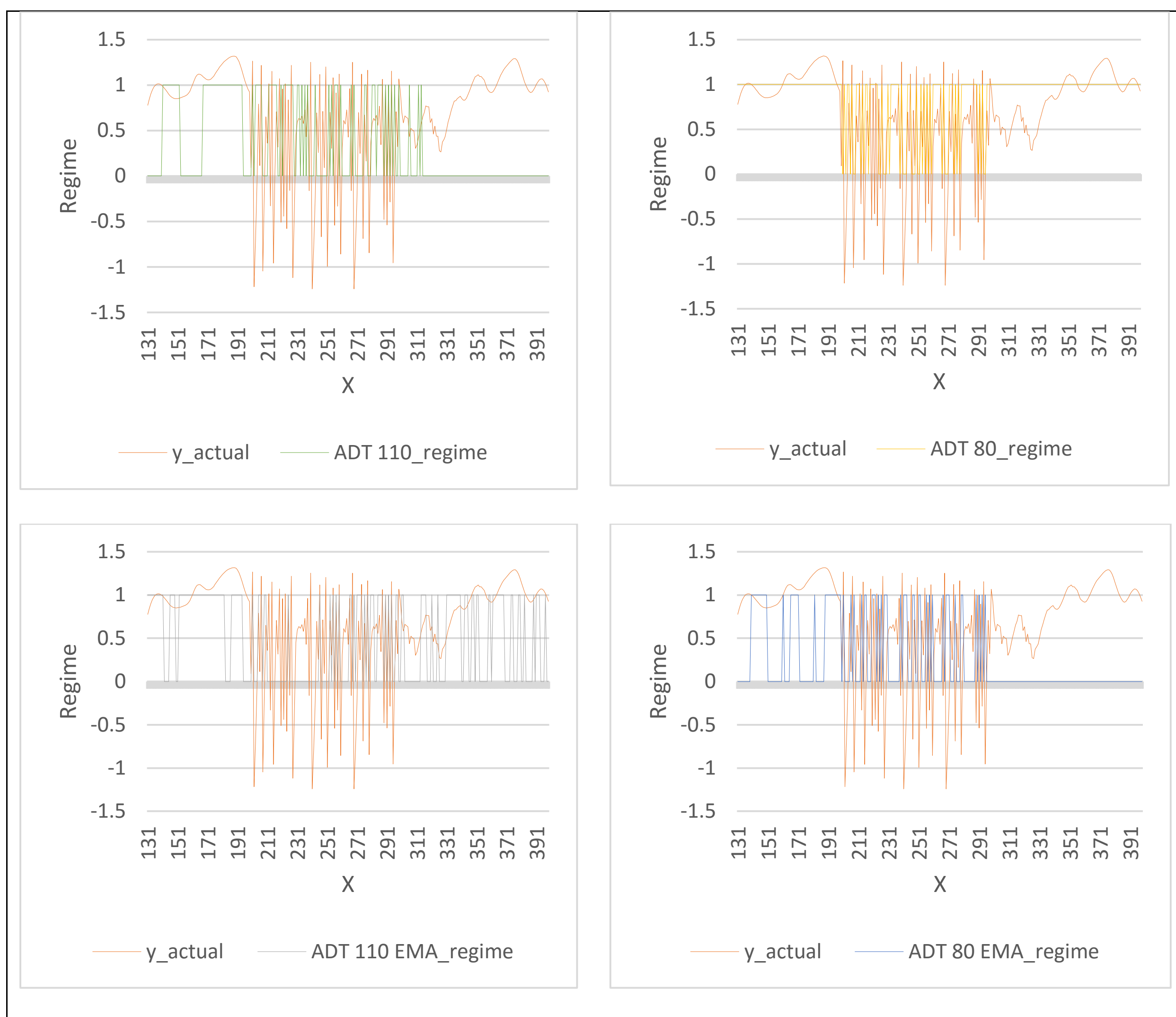



*Figure G15*. MGHEMMG 1-100 regime determination by best performing individuals.

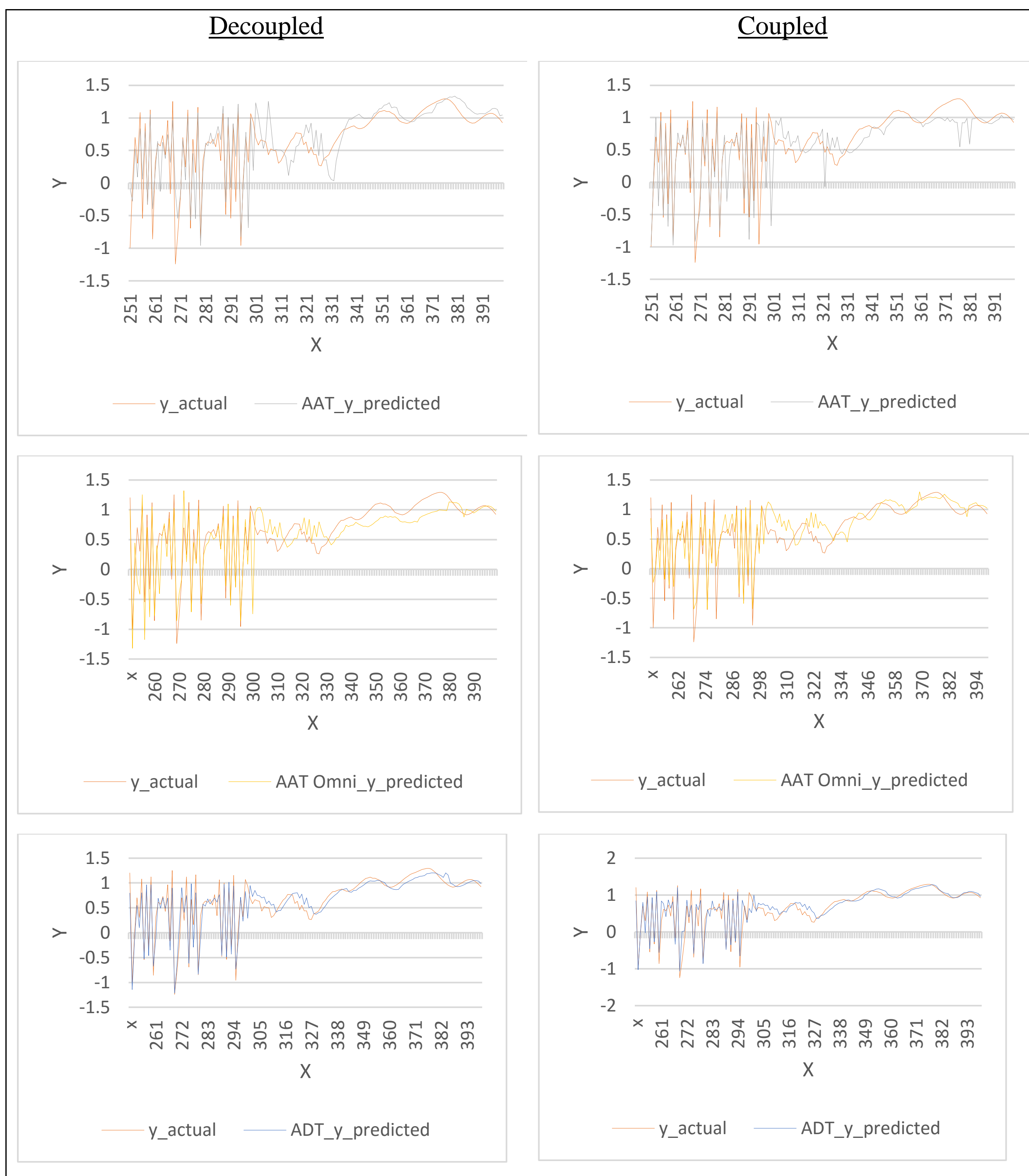



*Figure G16.* MGHENMG best prediction.

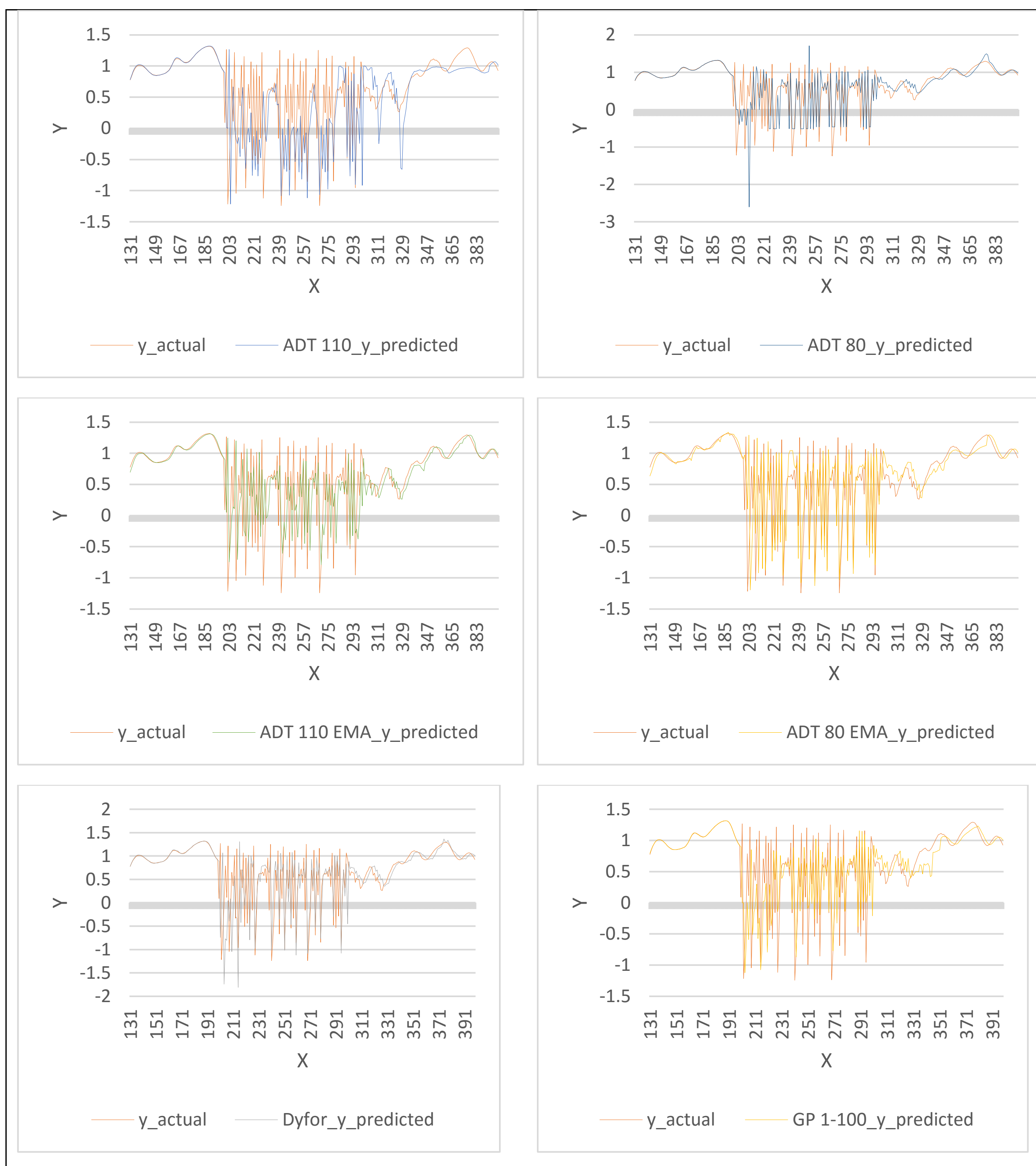
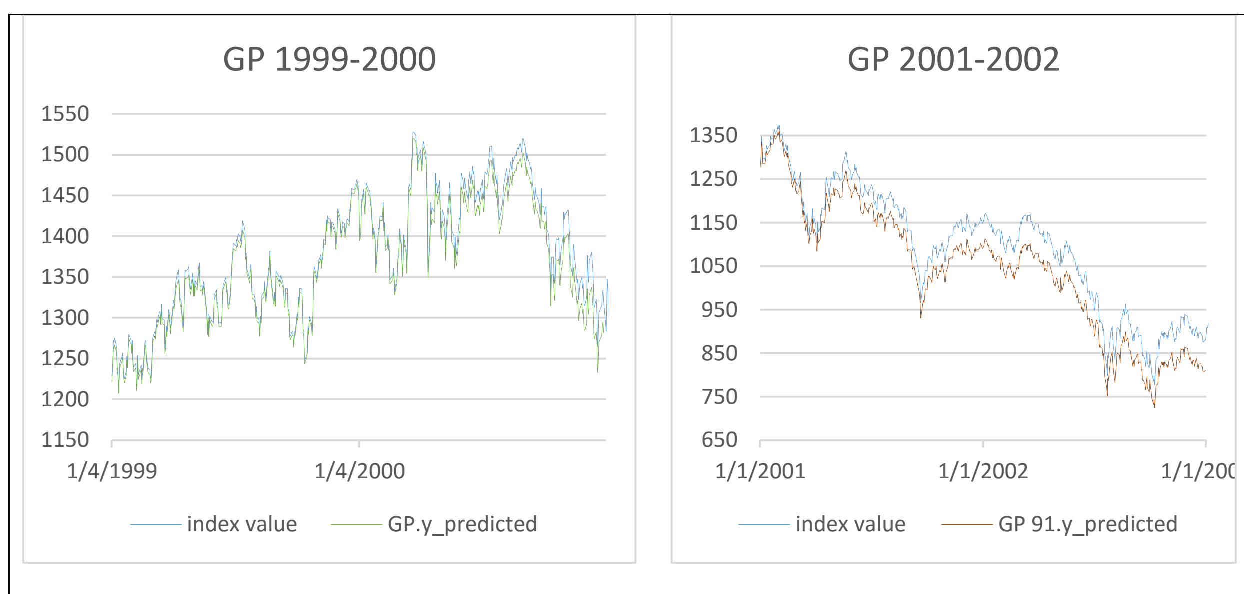
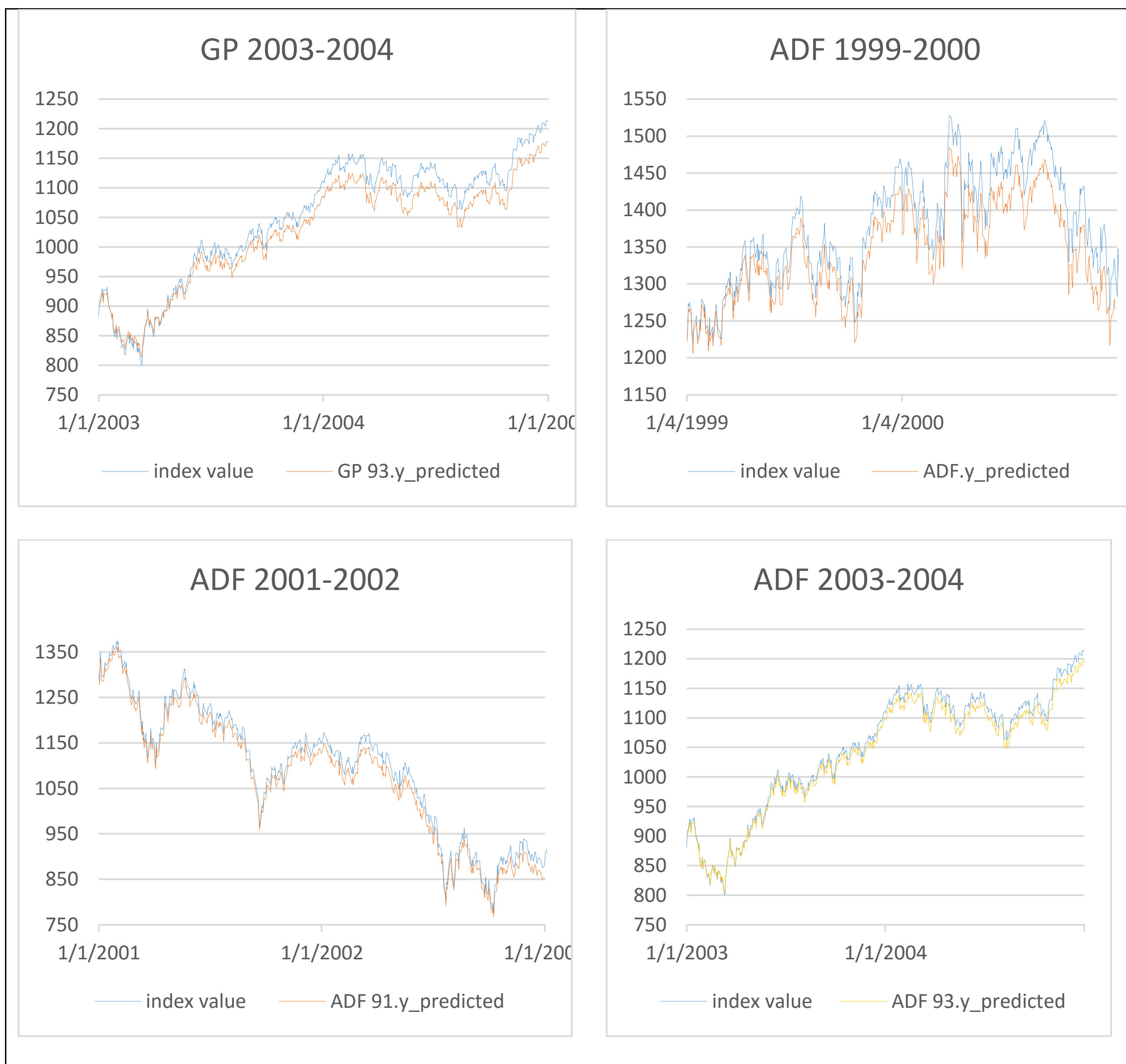



*Figure G17*. MGHENMG 1-100 best prediction.

**S&P 500 market data prediction.**

The following section includes the full results of all market data tests performed. Analysis periods correspond to the one of the three periods shown in Figure 48. Chart legend series titles without years indicate period 1, series with a suffix 91 correspond to period 2, and series with a suffix 93 indicate to period 3. Sliding window series test encompass the full analysis period. For example, in Figure G18 below, the series GP 91.y_predicted refers to predictions made using GP method for the second training period shown in Figure 48.




GP 2003-2004
1250
1200
1150
1100
1050
1000
950
900
850
800
750
1/1/2003
1/1/2004
index value
GP 93.y_predicted
ADF 1999-2000
1550
1500
1450
1400
1350
1300
1250
1200
1150
1/4/1999
1/4/2000
index value
ADF.y_predicted
ADF 2001-2002
1350
1250
1150
1050
950
850
750
1/1/2001
1/1/2002
index value
ADF 91.y_predicted
ADF 2003-2004
1250
1200
1150
1100
1050
1000
950
900
850
800
750
1/1/2003
1/1/2004
index value
ADF 93.y_predicted

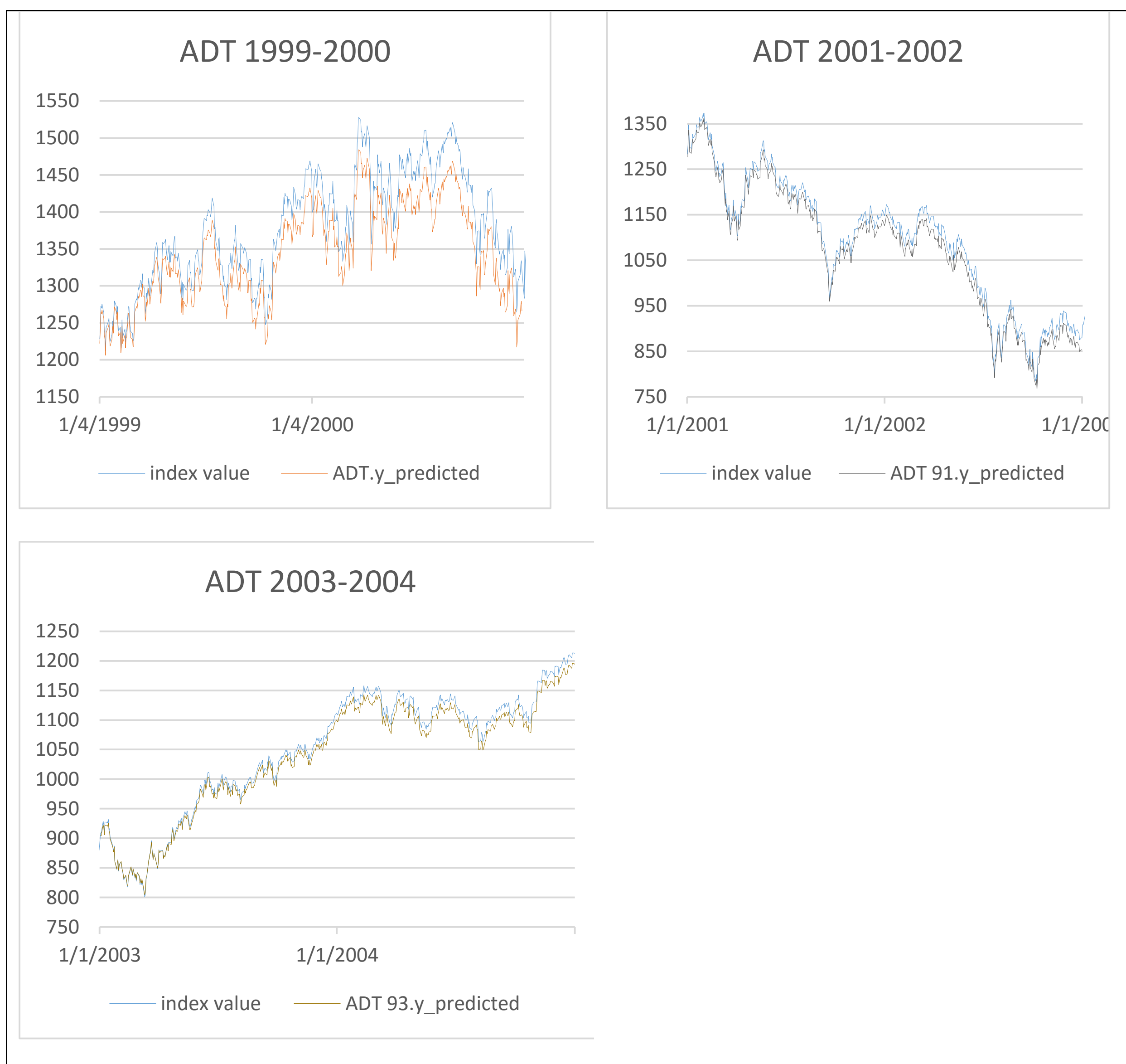


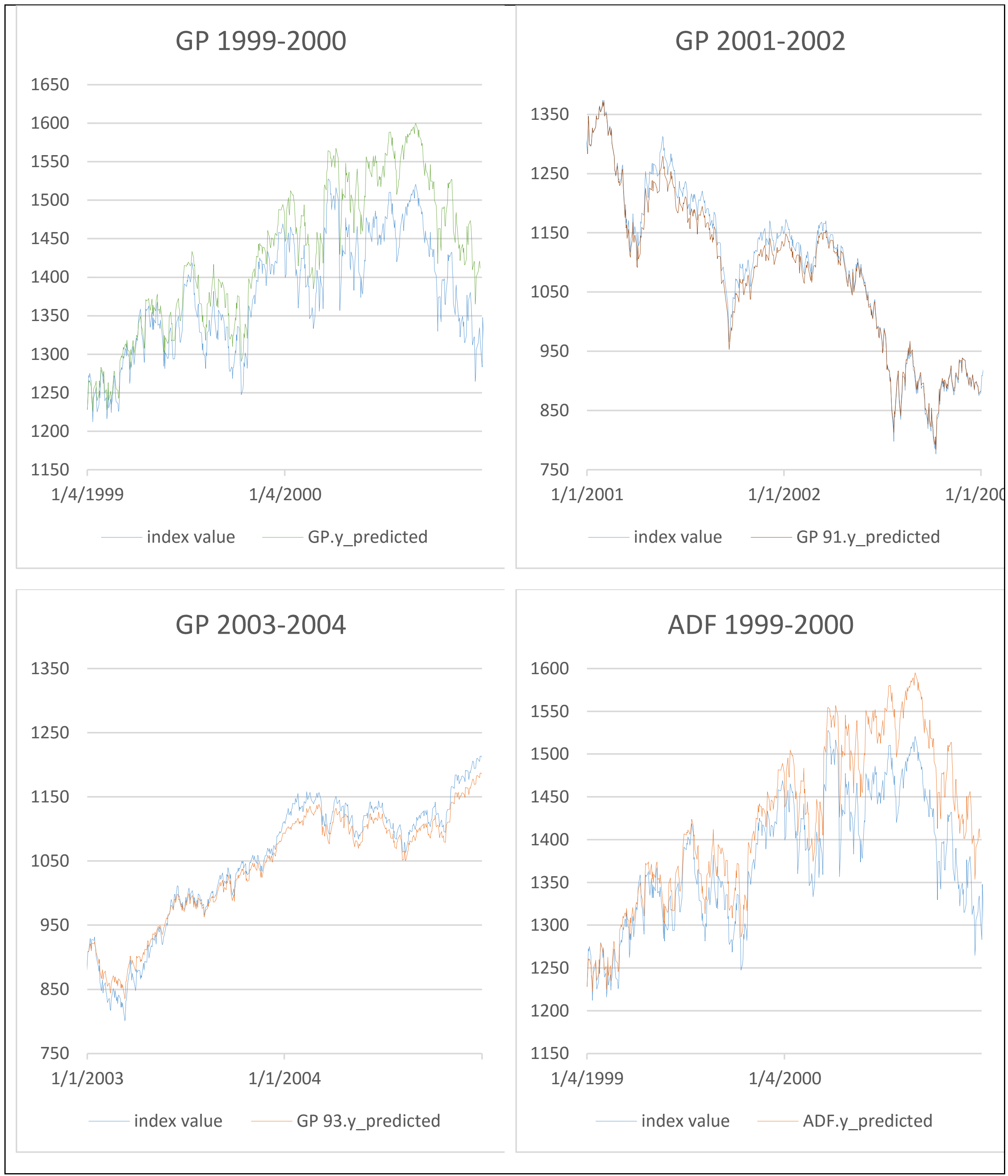

*Figure G18.* S&P 500 investment performance.

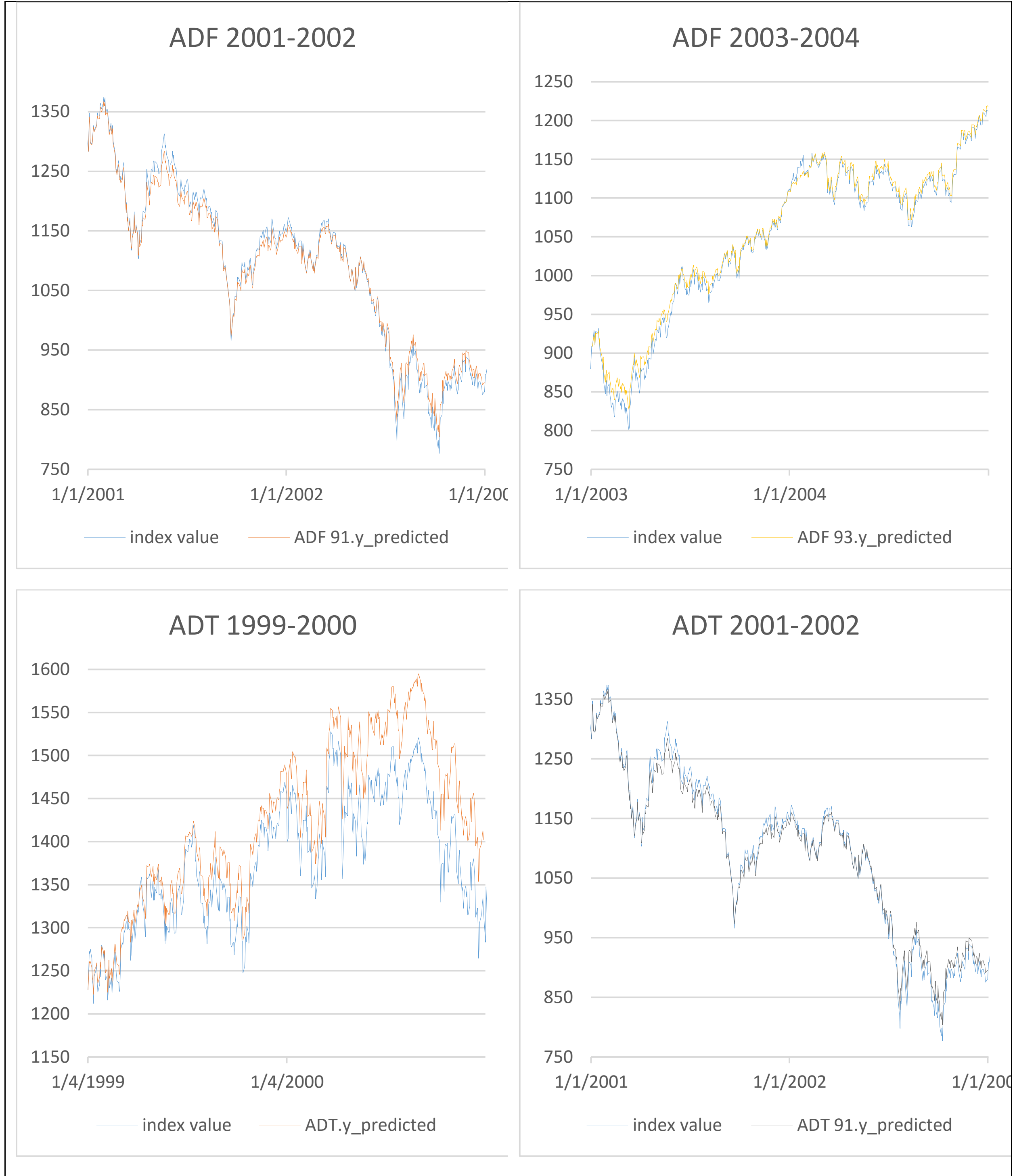
*Figure G19.* Investment performance in S&P 500 sliding window experiments.

Table G1. Number of Trades Executed, Including Transaction Costs

| Method | Mean | Std. Dev. | Min. | Max |
|---|---|---|---|---|
| | | 1999-2000 | | |
| GP | 7.14 | 8.55 | 1 | 38 |
| ADF | 8.22 | 13.57 | 1 | 81 |
| ADT | 5.28 | 6.47 | 1 | 21 |
| | | 2001-2002 | | |
| GP | 20.6 | 53.21 | 1 | 223 |
| ADF | 10.48 | 35.13 | 1 | 179 |
| ADT | 25.42 | 57.51 | 1 | 217 |
| | | 2003-2004 | | |
| GP | 4.44 | 6.21 | 1 | 33 |
| ADF | 2.72 | 4.57 | 1 | 25 |
| ADT | 1.56 | 2.10 | 1 | 11 |
| | | 1999-2004 | | |
| ADT | 45.38 | 14.13 | 16 | 86 |
| DyFor GP | 50.12 | 9.75 | 36 | 77 |

*Note*. Each two-year period has approximately 500 trading day opportunities. ADT and DyFor GP 1999-2004 evaluated approximately 100 weekly trading opportunities.


GP 1999-2000
1650
1600
1550
1500
1450
1400
1350
1300
1250
1200
1150
1/4/1999
1/4/2000
index value
GP.y_predicted
GP 2001-2002
1350
1250
1150
1050
950
850
750
1/1/2001
1/1/2002
index value
GP 91.y_predicted
GP 2003-2004
1350
1250
1150
1050
950
850
750
1/1/2003
1/1/2004
index value
GP 93.y_predicted
ADF 1999-2000
1600
1550
1500
1450
1400
1350
1300
1250
1200
1150
1/4/1999
1/4/2000
index value
ADF.y_predicted


ADF 2001-2002
1350
1250
1150
1050
950
850
750
1/1/2001
1/1/2002
index value
ADF 91.y_predicted
ADF 2003-2004
1250
1200
1150
1100
1050
1000
950
900
850
800
750
1/1/2003
1/1/2004
index value
ADF 93.y_predicted
ADT 1999-2000
1600
1550
1500
1450
1400
1350
1300
1250
1200
1150
1/4/1999
1/4/2000
index value
ADT.y_predicted
ADT 2001-2002
1350
1250
1150
1050
950
850
750
1/1/2001
1/1/2002
index value
ADT 91.y_predicted

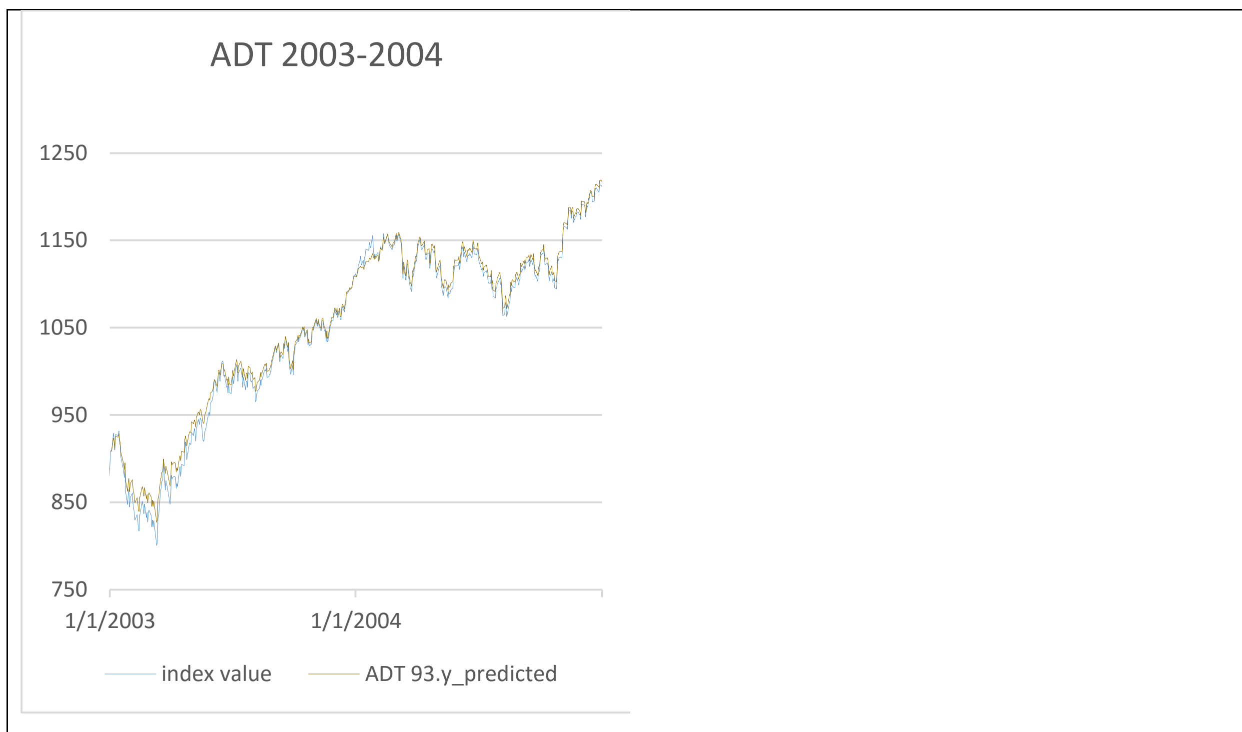



*Figure G20*. S&P 500 investment performance with transaction costs ignored.

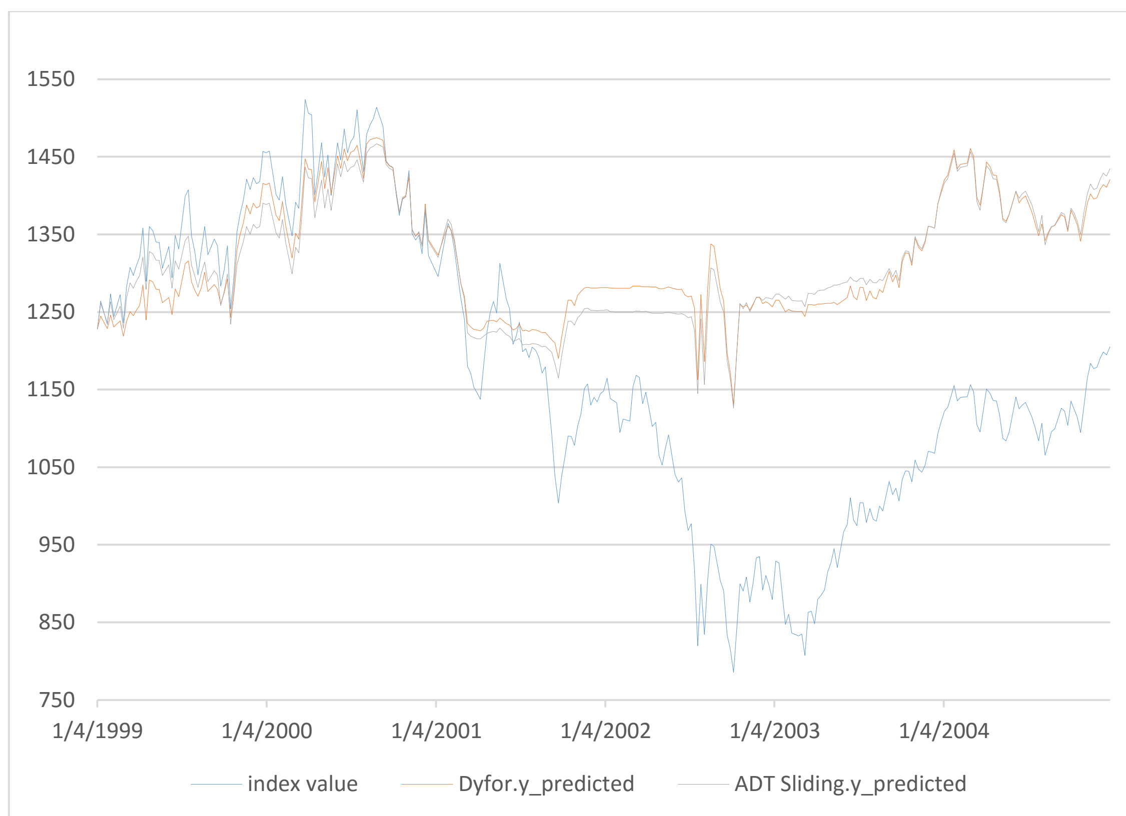



*Figure G21.* Investment performance in S&P 500 sliding window experiments with transaction costs ignored. Both ADT and DyFor GP achieve statistically significant higher returns than buy and hold.

Table G2. Number of Trades Executed, Not Including Transaction Costs

| Method | Mean | Std. Dev. | Min | Max |
|---|---|---|---|---|
| | | 1999-2000 | | |
| GP | 67.80 | 49.84 | 8 | 181 |
| ADF | 68.08 | 56.68 | 2 | 225 |
| ADT | 70.90 | 67.79 | 2 | 251 |
| | | 2001-2002 | | |
| GP | 45.19 | 54.59 | 4 | 231 |
| ADF | 53.56 | 58.39 | 6 | 258 |
| ADT | 54.44 | 58.80 | 6 | 229 |
| | | 2003-2004 | | |
| GP | 33.56 | 36.68 | 1 | 206 |
| ADF | 41.52 | 51.96 | 4 | 279 |
| ADT | 53.38 | 60.79 | 4 | 281 |
| | | 1999-2004 | | |
| ADT | 58.50 | 16.84 | 32 | 111 |
| DyFor GP | 60.28 | 12.45 | 36 | 87 |

*Note*. Each two-year period has approximately 500 trading day opportunities. ADT and DyFor GP 1999-2004 evaluated approximately 100 weekly trading opportunities.

# Appendix H

## Best S&P 500 Prediction Programs

This section will examine the best runs out of 50 trials for the best performing methods attempted in the market prediction experiment. The performance of ADT will also be included in case this was not the winning strategy. The best results achieved for each method and approach are listed in Table 17 through Table 20. These results should not be seen as realistically achievable; they were selected with full prior knowledge. Instead, the results illustrate the relative performance of the results generated by different evolutionary algorithms.

### Transaction costs considered.

This section highlights the best series out of 50 trials using a 0.5% transaction cost and summarized in Table 17.

#### *1999-2000 with transaction costs.*

Table H1. Best Return, 1999-2000, With Transaction Costs

| Best Approach | GP |
|---|---|
| Best Return | 0.1197 |
| ADT Return | 0.1042 |
| Buy and Hold Return | 0.0751 |

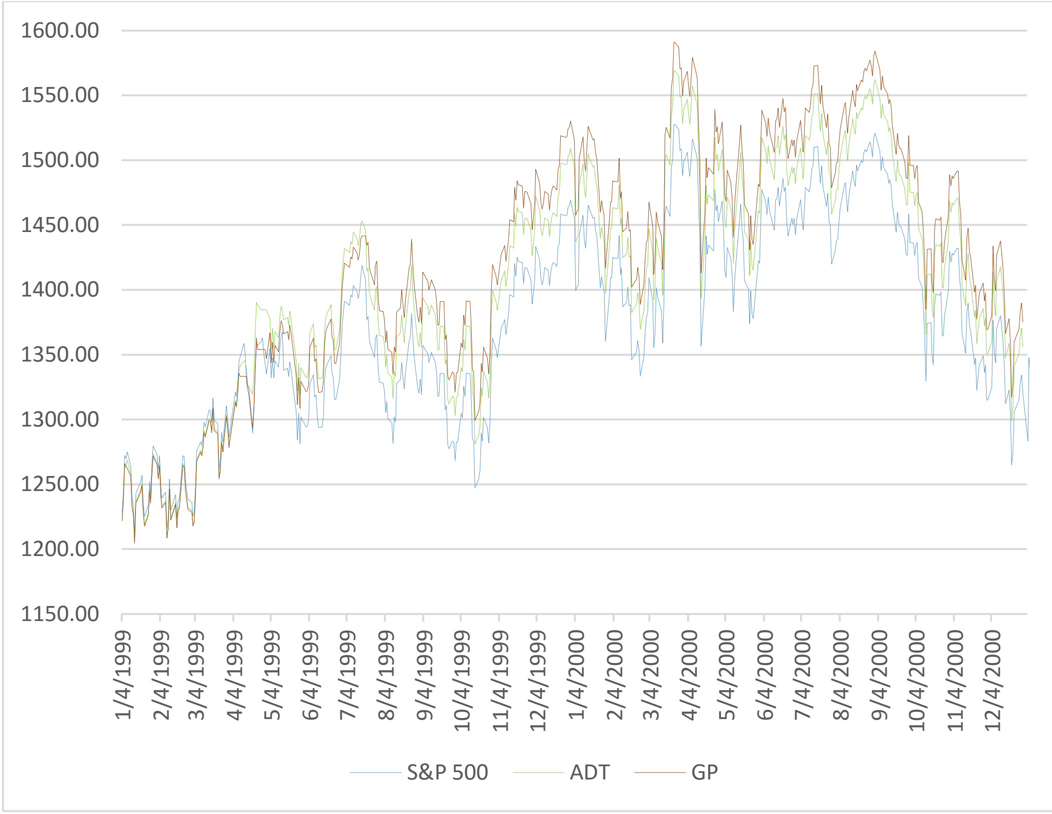



*Figure H1*. Best performing approach for 1999-2000 period with transaction costs. ADT is included for comparison.

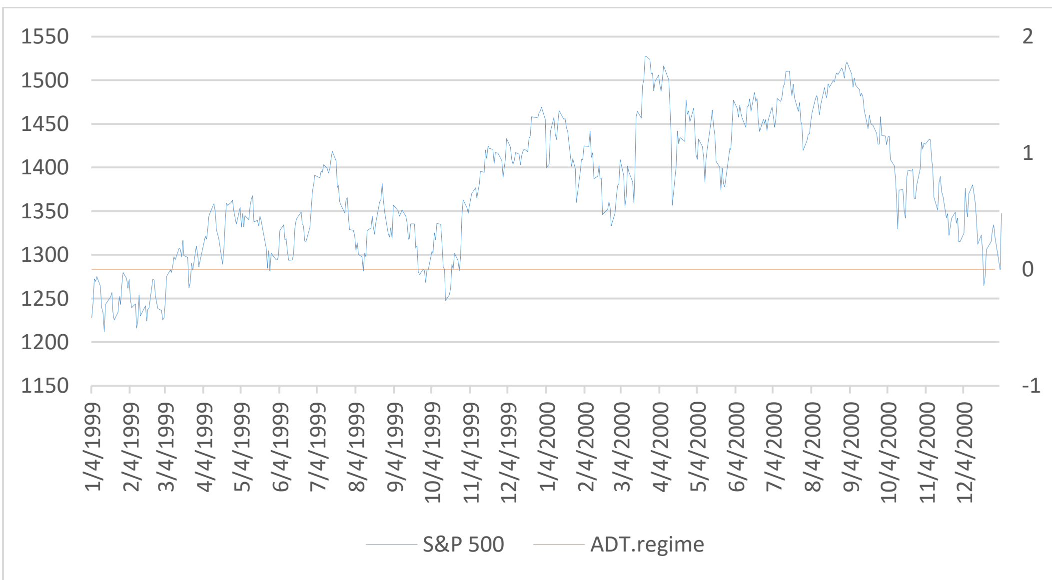



*Figure H2*. Regime predicted by best performing ADT individual for 1999-2000 period with transaction costs. The S&P 500 index is plotted on the primary axis.

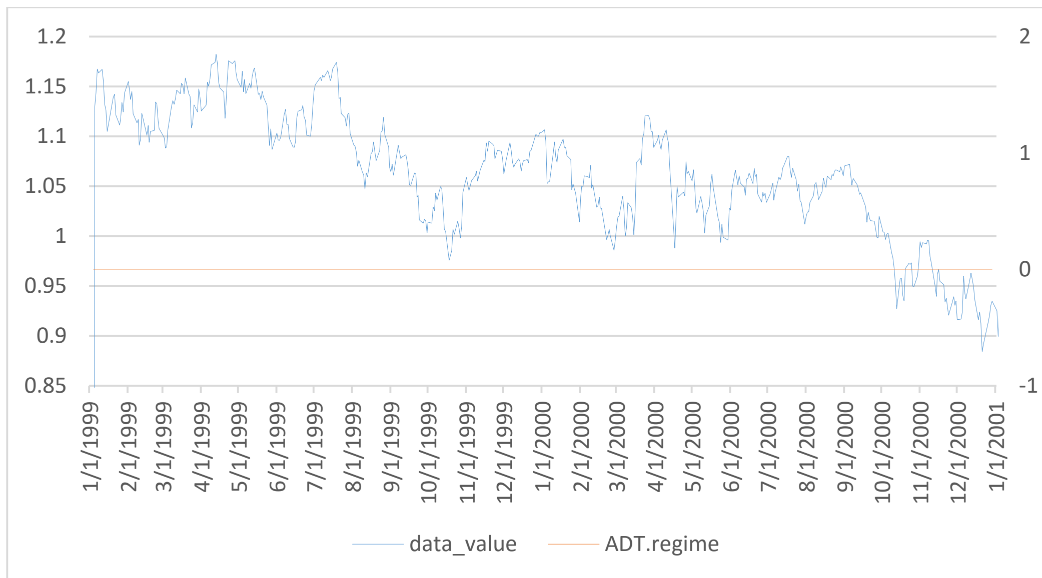



*Figure H3*. Predicted regime of best performing ADT individual for 1999-2000 period with transaction costs. The normalized S&P 500 index is plotted on the primary axis.

```
main->
(> 1.1602981190800095
  (movingAverage
    (+
      (movingAverage 125)
      (periodMinimum SP500.m250.1 0.3831607872563503))))
```

*Figure H4*. Best Performing GP Program for 1999-2000 period with transaction costs.

```
main->
(if
  (if
    (>
      (adf0
        (not
          (not false)) 1.1570045433187506)
      (movingAverage
        (+
          (+
            (adf0 false 14)
            (PeriodMaximum SP500.m250.1 233))
          (movingAverage
            (Norm
              (adf1 true Offsetvalue 0)
              (Norm 14 1.1570045433187506))))))
    (>
      (adf0 false 1.1570045433187506)
      (PeriodMaximum SP500.m250.1 Offsetvalue 0))
    (not
      (> 79 1.1570045433187506)))
```

```
 (if
    (not
      (if false false false))
    (not
      (and true false))
    (<
      (offsetValue SP500.m250.1 1.7847441320462223)
      (periodMinimum SP500.m250.1 1.7864632063244332)))
  (<
    (Norm
      (adf1 true Offsetvalue 0)
      (Norm 14 129))
    (adf1
      (if true true false)
      (- Offsetvalue 0 66))))


adf0->
(Norm
  (movingAverage
    (+
      (* arg1
        (+ arg1
          (+ arg1 arg1)))
      (movingAverage arg1)))
  (Norm
    (+
      (PeriodMaximum SP500.m250.1 arg1)
      (+ arg1
        (movingAverage arg1)))
    (periodMinimum SP500.m250.1
      (movingAverage arg1))))


adf1->
(+
  (periodMinimum SP500.m250.1
    (movingAverage
      (offsetValue SP500.m250.1
        (Norm
          (%
            (+ arg1 arg1) 1)
          (movingAverage 0)))))
  (Norm
    (offsetValue SP500.m250.1
      (movingAverage 1))
    (%
      (movingAverage
        (movingAverage arg1))
      (Norm
        (*
          (+ arg1 1)
          (offsetValue SP500.m250.1 arg1))
        (offsetValue SP500.m250.1 arg1)))))
```

*Figure H5*. Best performing ADT program for 1999-2000 period with transaction costs.

```
main->
(BinaryNumber
  (>
    (*
      (Norm
        (%
          (offsetValue SP500.m250.1
            (movingAverage 1.2937969012317059)) Offsetvalue 0)
        (offsetValue SP500.m250.1
          (PeriodMaximum SP500.m250.1
            (periodMinimum SP500.m250.1
              (offsetValue SP500.m250.1
                (+ 0.43683082957359565 0.9780983999359141))))))
      (periodMinimum SP500.m250.1
        (+
          (-
            (%
              (periodMinimum SP500.m250.1 30)
              (periodMinimum SP500.m250.1 238))
            (*
              (movingAverage
                (adf0 false Offsetvalue 0))
              (PeriodMaximum SP500.m250.1 168)))
          (periodMinimum SP500.m250.1
            (PeriodMaximum SP500.m250.1
              (Norm 0.8429000760570815 83))))))
    (movingAverage
      (+
        (PeriodMaximum SP500.m250.1
          (periodMinimum SP500.m250.1
            (movingAverage
              (+
                (offsetValue SP500.m250.1 Offsetvalue 0)
                (* 0.5160408045881169 202)))))
        (* 212 0.52816557399602)))))

adf0->
(and
  (>
    (PeriodMaximum SP500.m250.1
      (* arg1 arg1))
    (-
      (+ arg1 arg1)
      (periodMinimum SP500.m250.1 arg1)))
  (and
    (<
      (Norm
        (periodMinimum SP500.m250.1
          (periodMinimum SP500.m250.1 arg1)) arg1)
      (movingAverage arg1))
    (and
      (<
        (Norm arg1
          (movingAverage arg1))
        (movingAverage
          (periodMinimum SP500.m250.1 arg1)))
      (> arg1 0))))
adf1->
(or
  (or
    (if
```

```
      (>
        (Norm arg1
          (offsetValue SP500.m250.1
            (PeriodMaximum SP500.m250.1 arg1)))
        (Norm
          (movingAverage
            (periodMinimum SP500.m250.1
              (offsetValue SP500.m250.1 arg1)))
          (periodMinimum SP500.m250.1
            (periodMinimum SP500.m250.1
              (periodMinimum SP500.m250.1
                (movingAverage arg1))))))
      (if arg0 arg0 arg0)
      (> 0 arg1))
    (if
      (if arg0 false arg0)
      (or arg0 arg0)
      (< arg1 arg1)))
  (if
    (if
      (not
        (not
          (if
            (if arg0 arg0 arg0)
            (or arg0 arg0)
            (or arg0 arg0))))
      (and arg0 true)
      (> arg1 arg1))
    (if
      (if arg0 arg0 false)
      (and arg0 arg0)
      (and arg0 arg0))
    (<
      (movingAverage arg1)
      (periodMinimum SP500.m250.1
        (periodMinimum SP500.m250.1
          (periodMinimum SP500.m250.1
            (-
              (movingAverage arg1)
              (% arg1 arg1))))))))
```

*Figure H6.* Best performing ADT regime program for 1999-2000 period with transaction costs.

***2001-2002 with transaction costs.***

Table H2. Best Return, 2001-2002, With Transaction Costs

| Best Approach | ADT |
|---|---|
| Best Return | -0.0134 |
| Buy and Hold Return | 0.0751 |

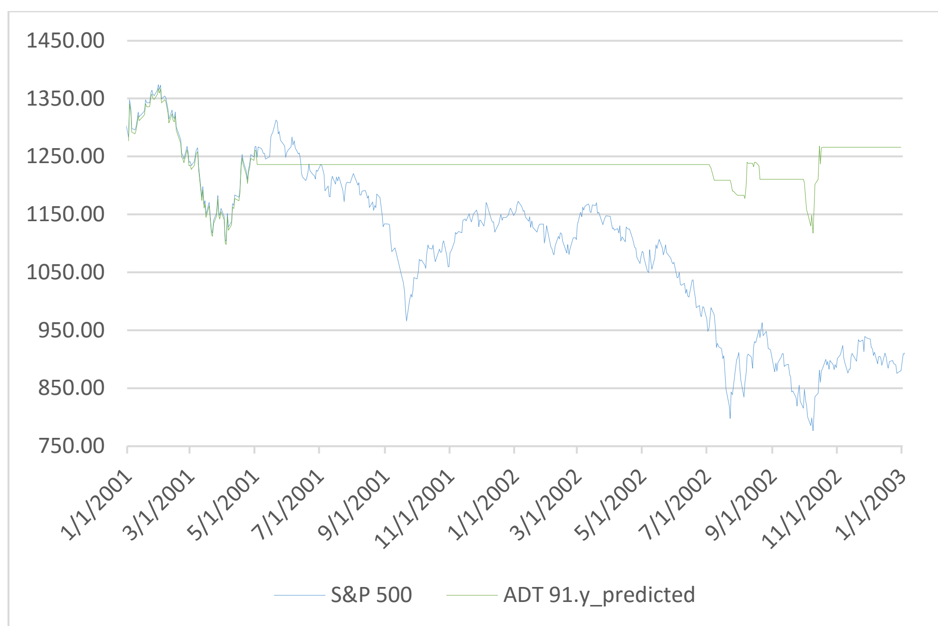



*Figure H7.* Best performing approach for 2001-2002 period including transaction costs.

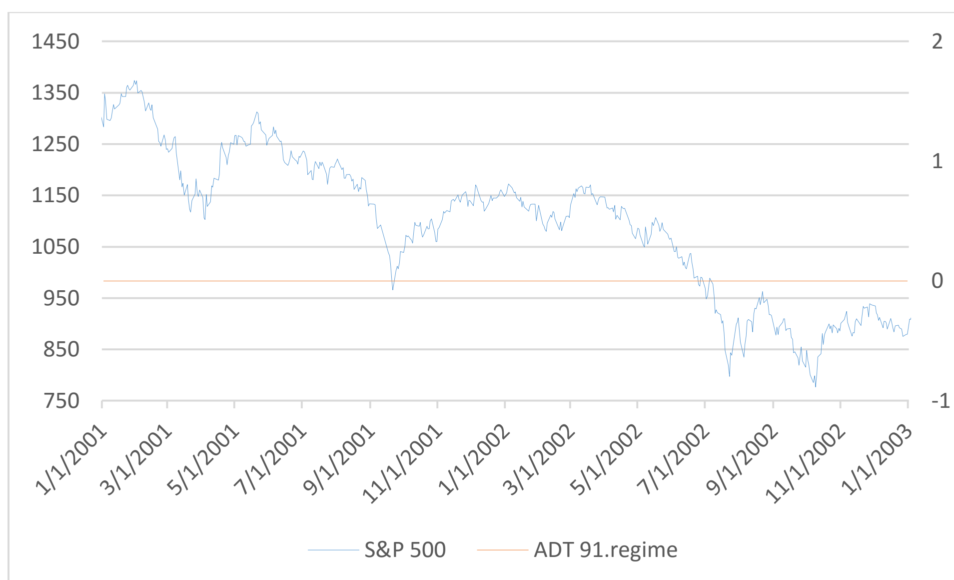



*Figure H8.* Regime predicted by best performing approach for 2001-2002 period including transaction costs.

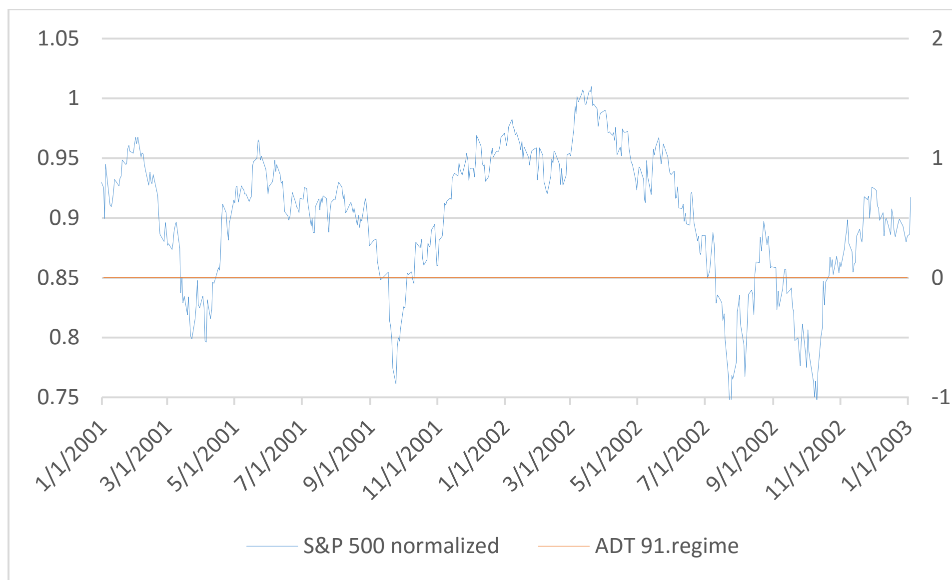



*Figure H9*. Regime predicted by best performing approach for 2001-2002 period including transaction costs, plotted against normalized target series.

```
main->
(adf1 false
  (+
    (+ Offsetvalue 0 0.7111180884746806)
    (+ 150 0.9819508139975657)))

adf0->
(or
  (if
    (or
      (< arg1 arg1)
      (if false true
        (if false true arg0)))
    (<
      (Norm arg1 arg1) arg1)
    (>
      (- arg1 arg1)
      (* arg1 arg1)))
  (and
    (if
      (and arg0 arg0)
      (or arg0 arg0)
      (if arg0 arg0 arg0))
    (>
      (% arg1 arg1)
      (PeriodMaximum SP500.m250.1 arg1))))

adf1->
(<
  (%
    (periodMinimum SP500.m250.1
      (PeriodMaximum SP500.m250.1 arg1))
    (*
      (offsetValue SP500.m250.1 arg1)
      (PeriodMaximum SP500.m250.1 arg1)))
```

```
  (movingAverage
    (PeriodMaximum SP500.m250.1
      (movingAverage arg1))))
```

*Figure H10.* Best performing ADT program for 2001-2002 period including transaction costs.

```
main->
(BinaryNumber
  (adf0
    (or
      (or
        (< 190 1.880980657798631)
        (if true false false))
      (if
        (>
          (offsetValue SP500.m250.1 247)
          (* 1.880980657798631 46))
        (adf0 false 108)
        (if true true true)))
    (PeriodMaximum SP500.m250.1
      (Norm
        (+ 0.7814113713749771 245)
        (% Offsetvalue 0 67)))))
adf0->
(and
  (and
    (not
      (> arg1
        (Norm
          (offsetValue SP500.m250.1 arg1) arg1)))
    (if
      (if arg0 arg0 arg0)
      (or arg0 arg0)
      (not true)))
  (not
    (if
      (not arg0)
      (<
        (offsetValue SP500.m250.1
          (Norm
            (offsetValue SP500.m250.1 arg1)
            (- arg1 arg1)))
        (offsetValue SP500.m250.1 arg1))
      (> arg1 arg1))))
adf1->
(-
  (+
    (+
      (movingAverage arg1)
      (- 1 arg1))
    (Norm
      (*
        (offsetValue SP500.m250.1
          (offsetValue SP500.m250.1 arg1))
        (movingAverage
          (offsetValue SP500.m250.1 arg1)))
      (%
        (-
```

```
          (% arg1 arg1)
          (offsetValue SP500.m250.1 arg1))
        (Norm
          (periodMinimum SP500.m250.1 0)
          (% arg1 arg1)))))
  (%
    (movingAverage
      (% arg1 arg1))
    (%
      (periodMinimum SP500.m250.1 arg1)
      (* arg1 arg1))))
```

*Figure H11.* Regime predicted by best performing ADT program for 2001-2002 period including transaction costs.

***2003-2004 with transaction costs.***

Table H3. Best Return, 2003-2004, With Transaction Costs

| | |
|---|---|
| Best Approach: | ADF |
| Best Return: | 0.3539 |
| ADT Return: | 0.3522 |
| Buy and Hold Return: | 0.3332 |

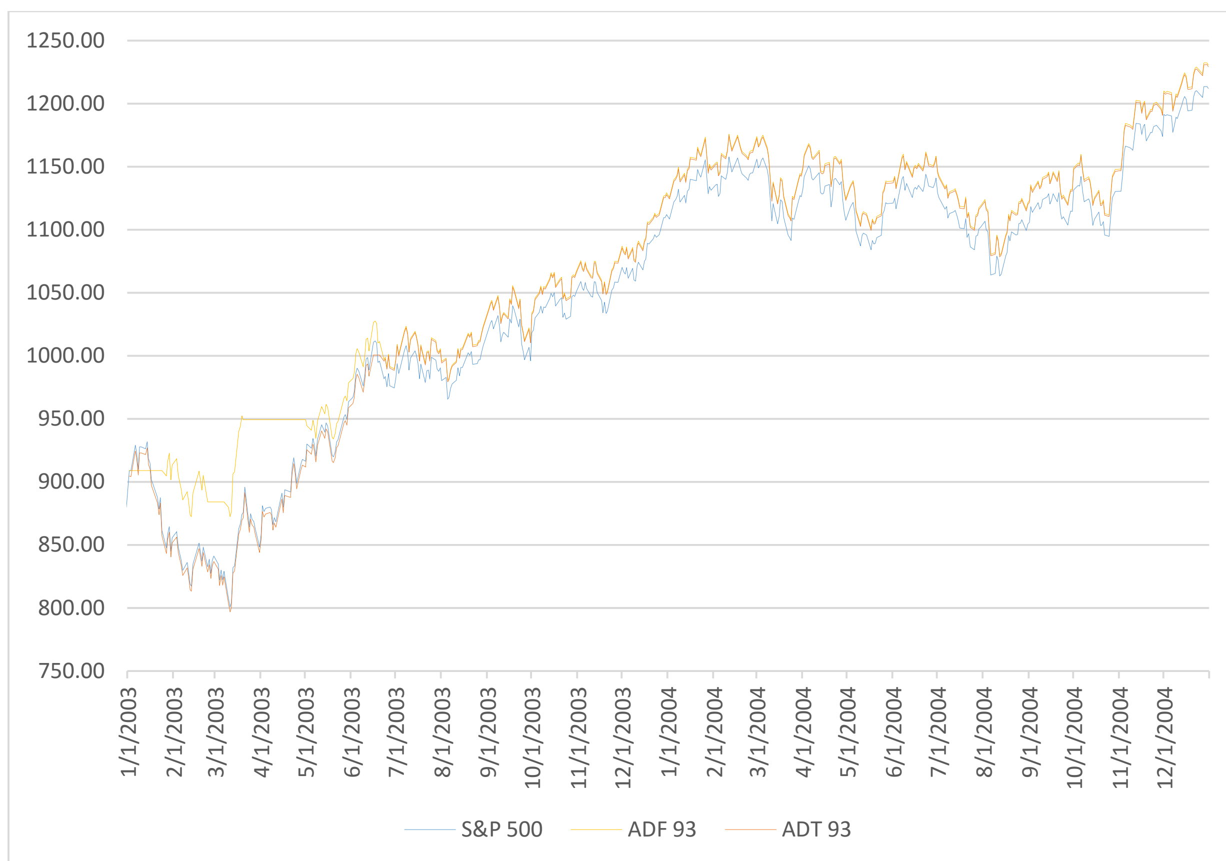



*Figure H12*. Best performing approach for 2003-2004 period with transaction costs.

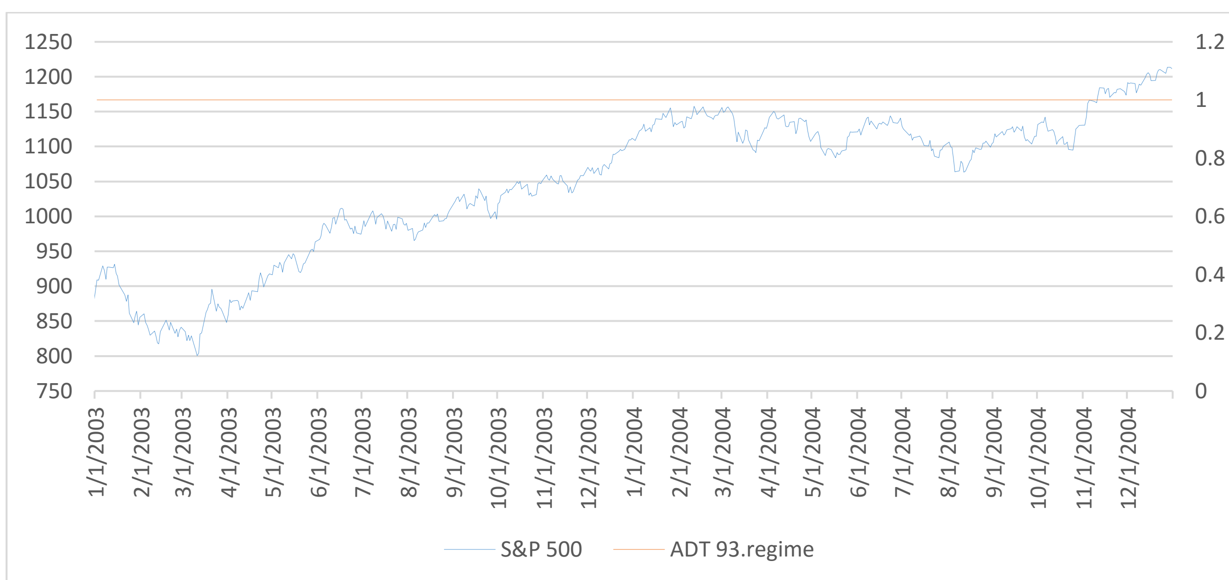



*Figure H13*. Regime predicted by best performing approach for 2003-2004 period with transaction costs.

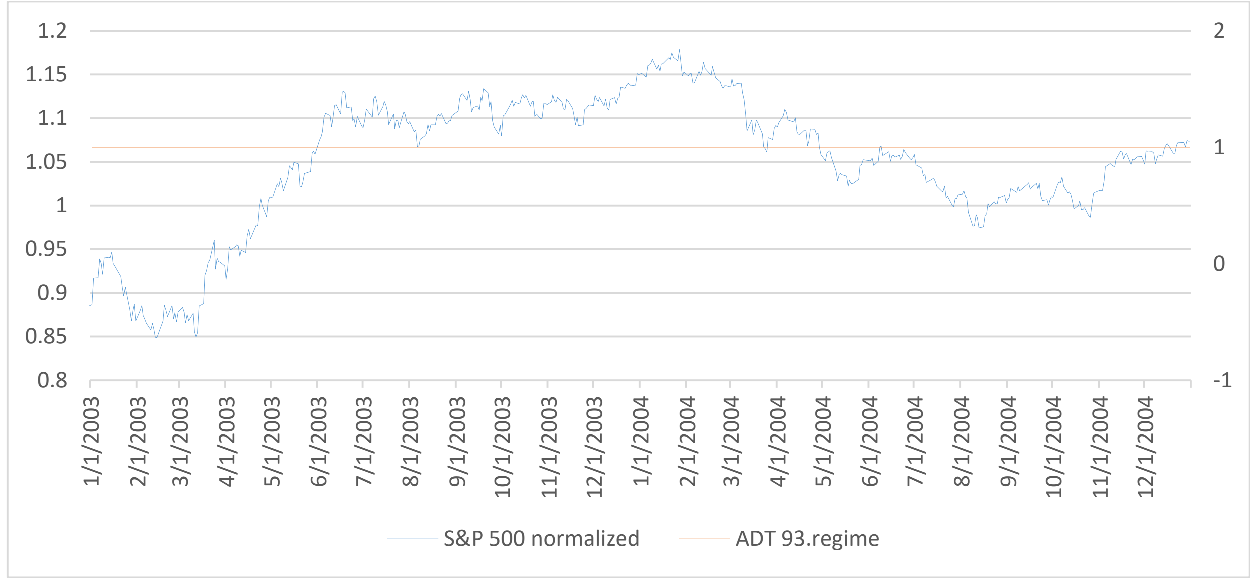



*Figure H14.* Regime predicted by best performing approach for 2003-2004 period with transaction costs, plotted against normalized target series.

```
main->
(or
  (or
    (if
      (> 1.023557394642299 Offsetvalue 0)
      (> 1.5812329159226999 38)
      (< Offsetvalue 0 8))
    (and
      (or false false)
      (not false)))
  (not
    (>
      (periodMinimum SP500.m250.1 5)
      (movingAverage 204))))
adf0->
(and
  (not
    (>
      (- arg1 arg1) 0))
  (if
    (if arg0
      (> arg1 arg1)
      (< arg1 arg1))
    (>
      (% arg1 arg1)
      (PeriodMaximum SP500.m250.1 arg1))
    (>
      (* arg1 arg1)
      (PeriodMaximum SP500.m250.1 arg1))))
adf1->
(*
  (%
    (Norm arg1
```

```
      (offsetValue SP500.m250.1 arg1)) arg1)
  (-
    (PeriodMaximum SP500.m250.1
      (movingAverage arg1))
    (* arg1 arg1)))
```

*Figure H15.* Best performing ADF program for 2003-2004 period with transaction costs.

```
main->
(<
  (offsetValue SP500.m250.1
    (PeriodMaximum SP500.m250.1
      (periodMinimum SP500.m250.1 113)))
  (%
    (-
      (Norm 0.4620173131659049 1.8681790215436784)
      (movingAverage 1.9695191046103773))
    (Norm
      (offsetValue SP500.m250.1 91)
      (PeriodMaximum SP500.m250.1 Offsetvalue 0))))
adf0->
(>
  (movingAverage
    (offsetValue SP500.m250.1
      (movingAverage arg1)))
  (offsetValue SP500.m250.1
    (movingAverage
      (*
        (% arg1 arg1) arg1))))
adf1->
(%
  (%
    (+
      (+ arg1 arg1)
      (Norm arg1 arg1)) arg1)
  (-
    (PeriodMaximum SP500.m250.1 arg1)
    (+
      (PeriodMaximum SP500.m250.1 arg1)
      (PeriodMaximum SP500.m250.1 arg1))))
```

*Figure H16.* Best performing ADT program for 2003-2004 period with transaction costs.

```
main->
(BinaryNumber
  (if
    (and
      (and
        (<
          (+ 69 Offsetvalue 0)
          (PeriodMaximum SP500.m250.1 Offsetvalue 0))
        (not
          (< Offsetvalue 0 Offsetvalue 0)))
      (or
        (if
          (and true false)
          (if true true true)
          (< Offsetvalue 0 121))
```

```
        (and
          (or false false)
          (and false true)))) false true))
adf0->
(<
  (PeriodMaximum SP500.m250.1
    (periodMinimum SP500.m250.1
      (periodMinimum SP500.m250.1 arg1))) arg1)
adf1->
(*
  (% arg1 arg1)
  (%
    (+
      (movingAverage arg1)
      (- 0 arg1))
    (periodMinimum SP500.m250.1
      (-
        (periodMinimum SP500.m250.1
          (*
            (% arg1 arg1)
            (periodMinimum SP500.m250.1 arg1)))
        (periodMinimum SP500.m250.1
          (movingAverage
            (- arg1 arg1)))))))
```

*Figure H17.* Regime program for best performing ADT individual for 2003-2004 period with transaction costs.

### *1999-2004 with transaction costs.*

Only DyFor GP and ADT were run on the full series range in a single test. As these both dynamically evolve prediction programs over time, the evolved code is not included in this appendix. The full data details are available in the online data repository.

Table H4. Best Return, 1999-2004, With Transaction Costs

| Best Approach | ADT |
|---|---|
| Best Return | 0.4592 |
| Buy and Hold Return | -0.0189 |

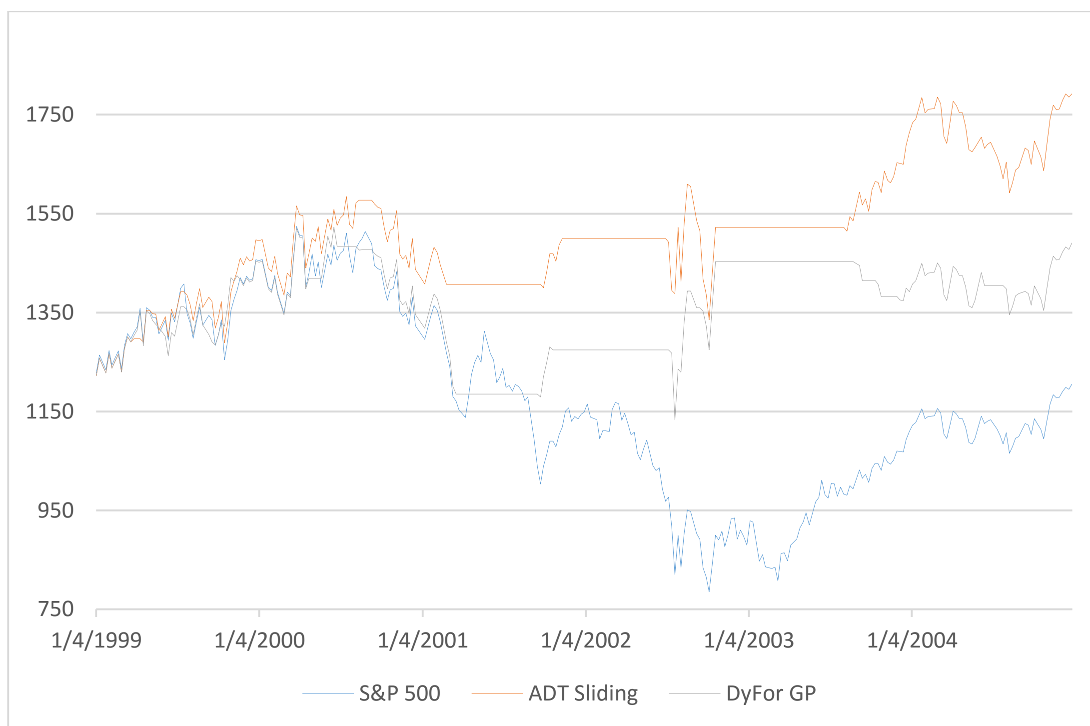



*Figure H18.* Best returns for 1999-2014 period with transaction costs.

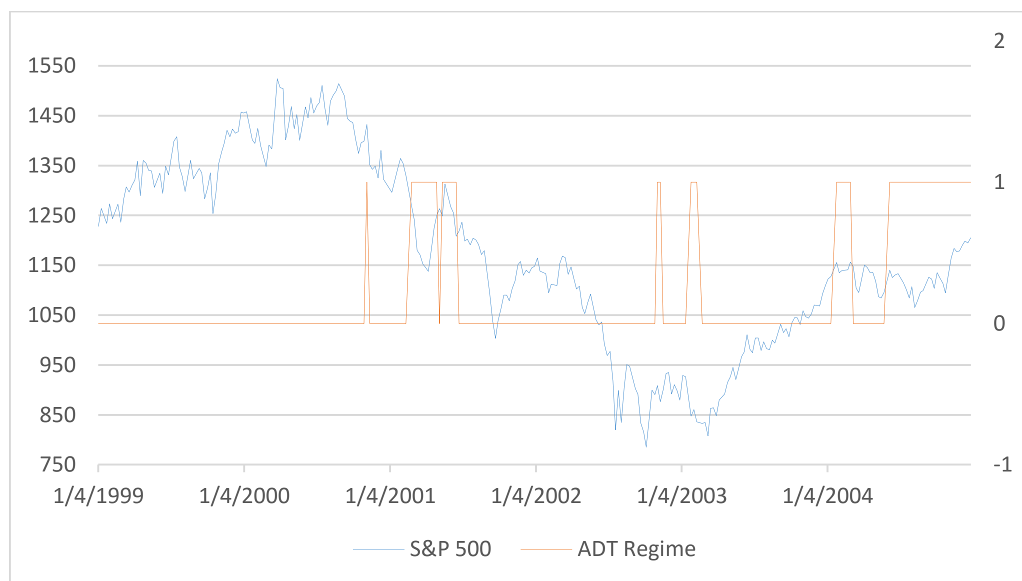



*Figure H19.* Regime predicted by best performing ADT individual for 1999-2004 period with transaction costs.

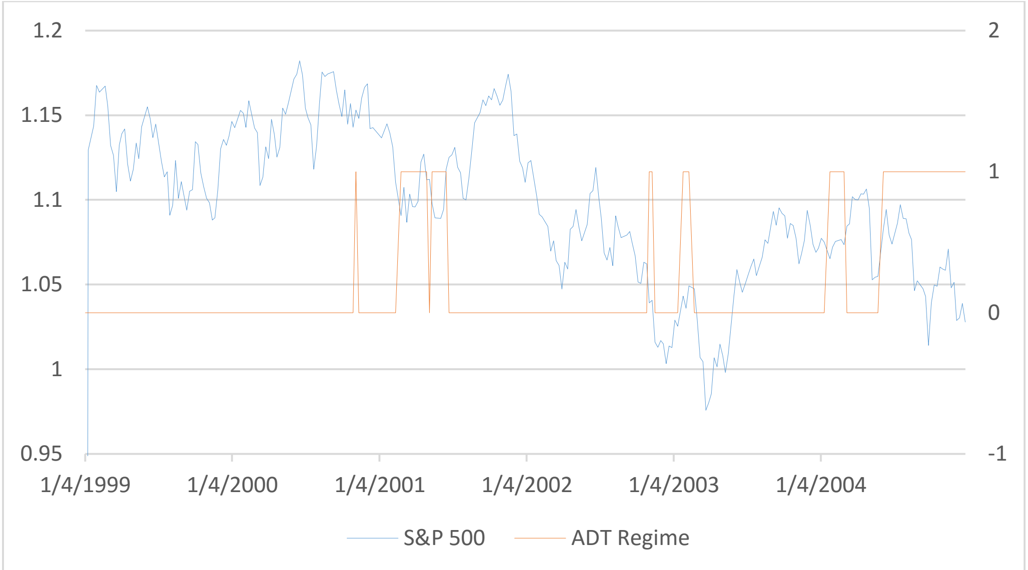



*Figure H20.* Regime predicted by best performing ADT individual for 1999-2004 period with transaction costs, plotted against normalized series.

**No transaction costs.**

***1999-2000 without transaction costs.***

Table H5. Best Return, 1999-2000, Without Transaction Costs

| | |
|---|---|
| Best Approach | ADF |
| Best Return | 0.5112 |
| ADT Return | 0.4796 |
| Buy and Hold Return | 0.0751 |

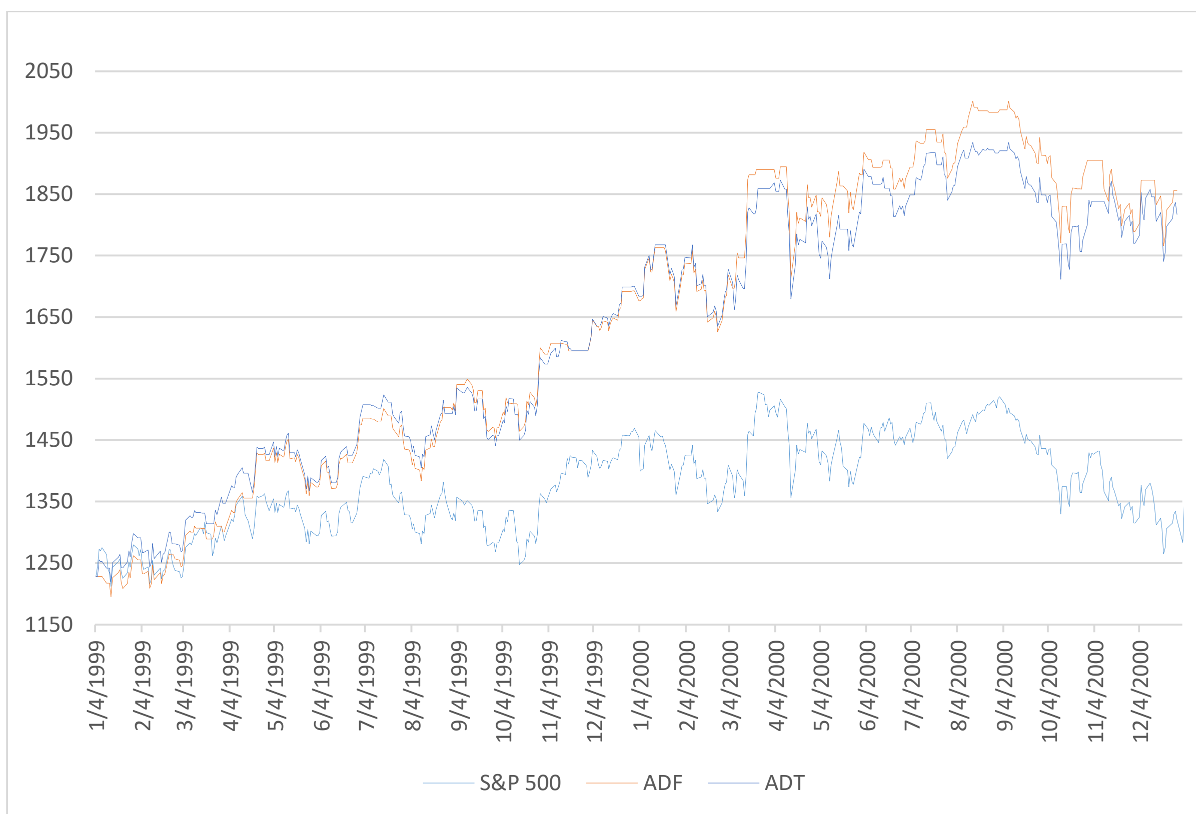



*Figure H21*. Best performing approach for 1999-2000 period without transaction costs. ADT is included for comparison.

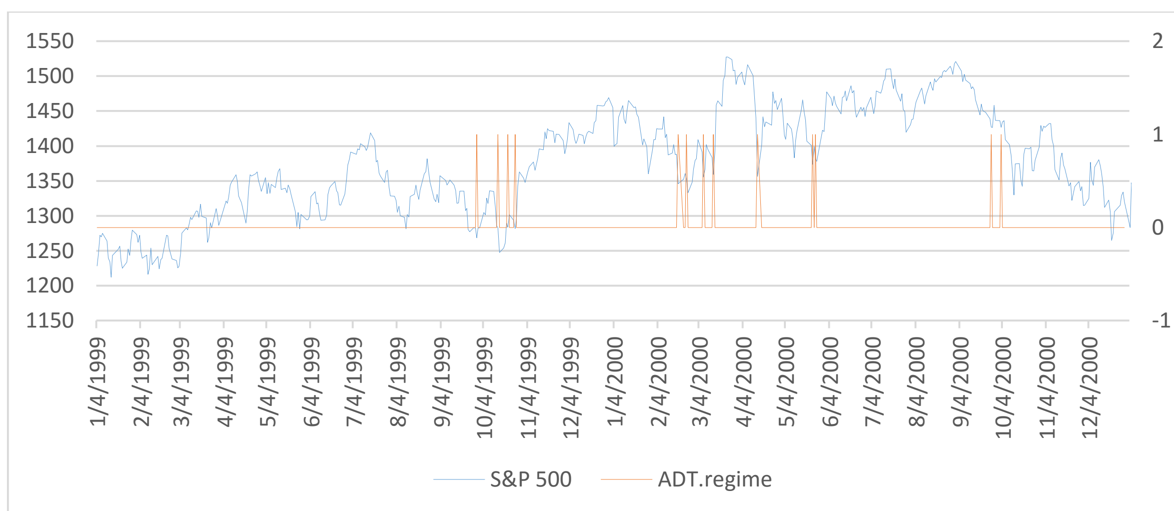



*Figure H22*. Regime predicted by best performing ADT individual for 1999-2000 period without transaction costs. The S&P 500 index is plotted on the primary axis.

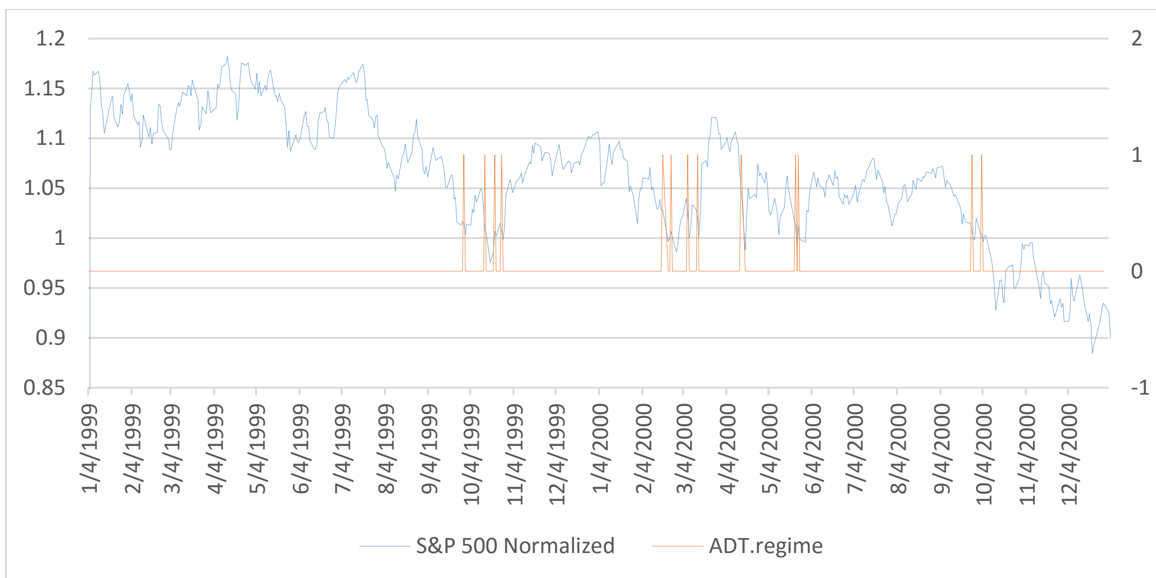



Figure H23. Predicted regime of best performing ADT individual for 1999-2000 period without transaction costs. The normalized S&P 500 index is plotted on the primary axis.

```
main->
(or
  (not
    (if
      (if true false false) true
      (<
        (movingAverage
          (+
            (* Offsetvalue 0 17)
            (% Offsetvalue 0 205)))
        (Norm
          (+
            (periodMinimum SP500.m250.1 Offsetvalue 0)
            (offsetValue SP500.m250.1 Offsetvalue 0))
          (movingAverage
            (* Offsetvalue 0 Offsetvalue 0))))))
  (if
    (<
      (movingAverage
        (offsetValue SP500.m250.1 Offsetvalue 0)) Offsetvalue 0)
    (>
      (adf1
        (or
          (not
            (if
              (not false)
              (< 0.9887563673051396 0.9368307229011728)
              (> 0.4099380718399315 148)))
          (not
            (not
              (not false)))) Offsetvalue 0)
      (movingAverage
        (-
```

```
          (-
            (movingAverage 94)
            (+ 187 0.8382032016270007))
          (periodMinimum SP500.m250.1
            (offsetValue SP500.m250.1
              (Norm
                (movingAverage Offsetvalue 0)
                (* 212 0.7017350783758893))))))) false))
adf0->
(%
  (periodMinimum SP500.m250.1
    (PeriodMaximum SP500.m250.1 arg1))
  (-
    (- arg1
      (+ arg1 arg1)) arg1))
adf1->
(Norm
  (%
    (+
      (*
        (offsetValue SP500.m250.1
          (+ arg1 arg1))
        (movingAverage
          (+ arg1 arg1)))
      (%
        (Norm
          (offsetValue SP500.m250.1 arg1)
          (* arg1 arg1))
        (-
          (% arg1 arg1)
          (movingAverage arg1))))
    (-
      (% arg1 arg1)
      (periodMinimum SP500.m250.1
        (PeriodMaximum SP500.m250.1
          (%
            (*
              (periodMinimum SP500.m250.1
                (offsetValue SP500.m250.1 arg1))
              (-
                (% arg1 arg1)
                (periodMinimum SP500.m250.1 arg1))) arg1)))))
  (PeriodMaximum SP500.m250.1
    (periodMinimum SP500.m250.1
      (Norm
        (offsetValue SP500.m250.1 arg1) arg1))))
```

*Figure H24.* Best Performing ADF Program for 1999-2000 period without transaction costs.

```
main->
(not
  (if
    (if
      (if
        (if
          (if
            (and
              (and
                (not true)
                (and true false))
```

```
              (not
                (not false)))
            (<
              (periodMinimum SP500.m250.1
                (+ 0.04534347367068148 0.6752217626562438)) 1.6581434065095961)
            (not
              (<
                (offsetValue SP500.m250.1 Offsetvalue 0)
                (periodMinimum SP500.m250.1 0.7496229268283094)))) true false)
        (if true true true)
        (< 0.6476672445548195 0.5586636132707414))
      (if
        (not
          (>
            (movingAverage
              (Norm
                (movingAverage Offsetvalue 0)
                (Norm 211 193)))
            (+
              (offsetValue SP500.m250.1
                (offsetValue SP500.m250.1 0.6736792933277249))
              (Norm
                (periodMinimum SP500.m250.1 1.4428131113809106)
                (PeriodMaximum SP500.m250.1 0.4054839088970026)))))
        (if false false false)
        (and true true))
      (not
        (if false true false)))
    (<
      (+
        (+
          (+ 116 56)
          (periodMinimum SP500.m250.1 Offsetvalue 0))
        (offsetValue SP500.m250.1
          (movingAverage 117)))
      (periodMinimum SP500.m250.1
        (periodMinimum SP500.m250.1
          (Norm Offsetvalue 0 1.9788305923688534))))
    (>
      (offsetValue SP500.m250.1
        (+ Offsetvalue 0 1.7509562996891792))
      (+
        (movingAverage 30)
        (Norm
          (periodMinimum SP500.m250.1
            (offsetValue SP500.m250.1 0.5586636132707414)) Offsetvalue 0)))))
adf0->
(*
  (PeriodMaximum SP500.m250.1
    (+ arg1
      (movingAverage arg1)))
  (*
    (offsetValue SP500.m250.1 arg1)
    (- arg1 arg1)))
adf1->
(or
  (<
    (offsetValue SP500.m250.1
      (offsetValue SP500.m250.1
        (periodMinimum SP500.m250.1 arg1)))
    (+
```

```
      (*
        (periodMinimum SP500.m250.1 arg1)
        (+ arg1 arg1))
      (PeriodMaximum SP500.m250.1
        (* arg1 arg1))))
  (if
    (or
      (< arg1 arg1)
      (or false arg0))
    (not
      (or
        (or
          (or arg0 arg0)
          (and true arg0))
        (>
          (PeriodMaximum SP500.m250.1 arg1)
          (periodMinimum SP500.m250.1 arg1))))
    (>
      (* arg1 arg1)
      (Norm arg1 arg1))))
```

*Figure H25*. Best performing ADT program for 1999-2000 period without transaction costs.

```
main->
(BinaryNumber
  (<
    (movingAverage Offsetvalue 0)
    (movingAverage
      (PeriodMaximum SP500.m250.1
        (PeriodMaximum SP500.m250.1
          (-
            (periodMinimum SP500.m250.1
              (PeriodMaximum SP500.m250.1 Offsetvalue 0))
            (Norm
              (% 1.9155758128983693 1.6830512465738168)
              (+ 238 145))))))))
adf0->
(*
  (Norm
    (movingAverage
      (Norm arg1 arg1)) arg1)
  (+ arg1
    (* arg1
      (PeriodMaximum SP500.m250.1 arg1))))
adf1->
(%
  (movingAverage arg1)
  (-
    (+
      (+ arg1 arg1)
      (* arg1
        (PeriodMaximum SP500.m250.1
          (Norm
            (- arg1 arg1)
            (+ arg1 arg1))))) arg1))
```

*Figure H26*. Best performing ADT regime program for 1999-2000 period without transaction costs.

***2001-2002 without transaction costs.***

Table H6. Best Return, 2001-2002, Without Transaction Costs

| Best Approach | ADF |
|---|---|
| Best Return | 0.0196 |
| ADT Return | -0.1245 |
| Buy and Hold Return | -0.3144 |

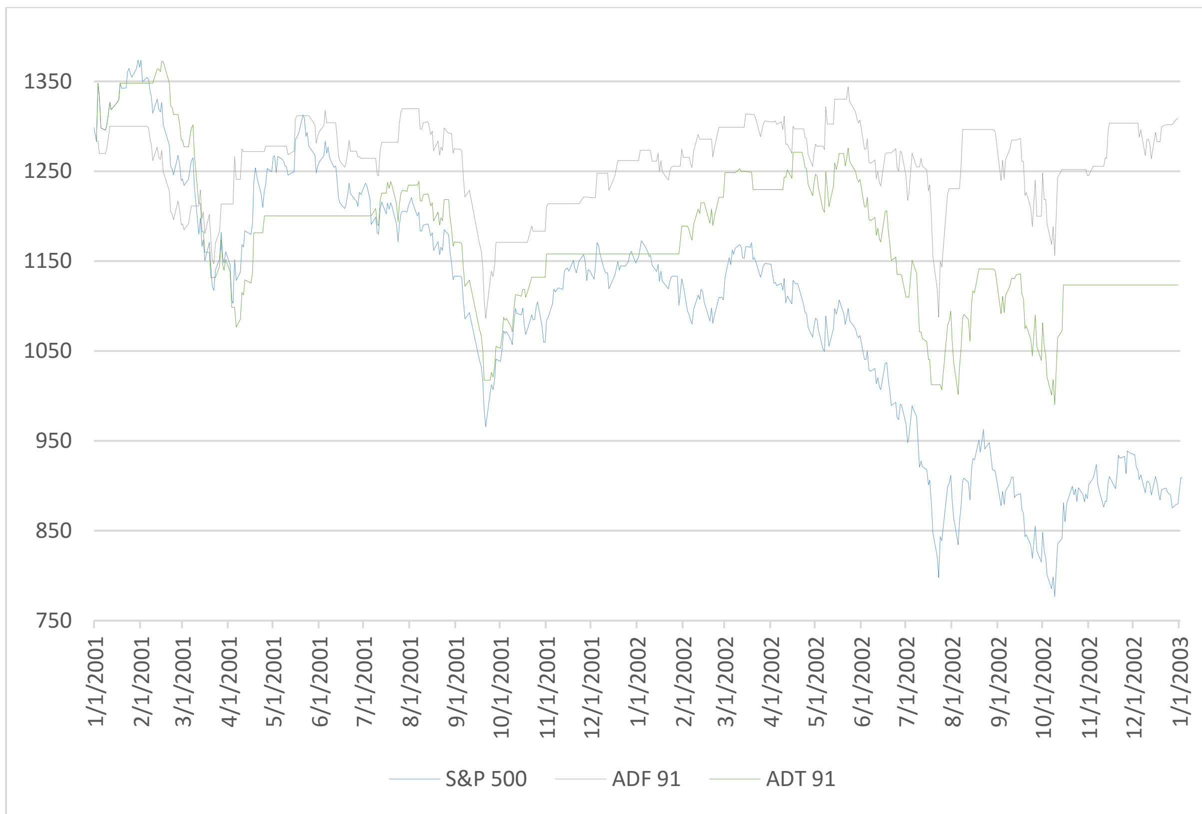



*Figure H27.* Best performing approach for 2001-2002 period without transaction costs.

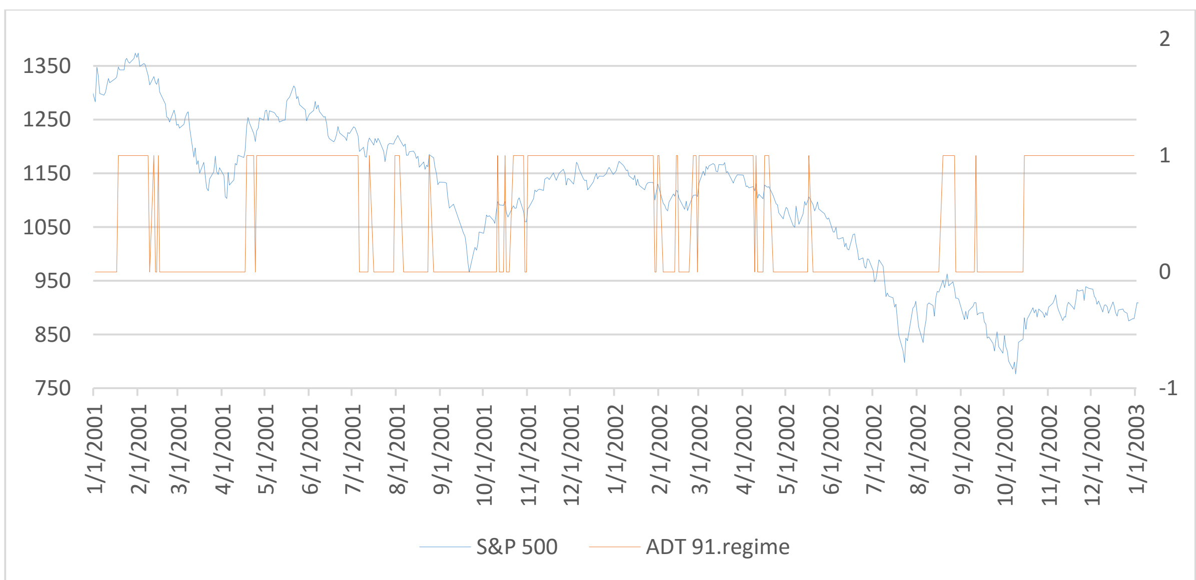



*Figure H28.* Regime predicted by best performing approach for 2001-2002 period without transaction costs.

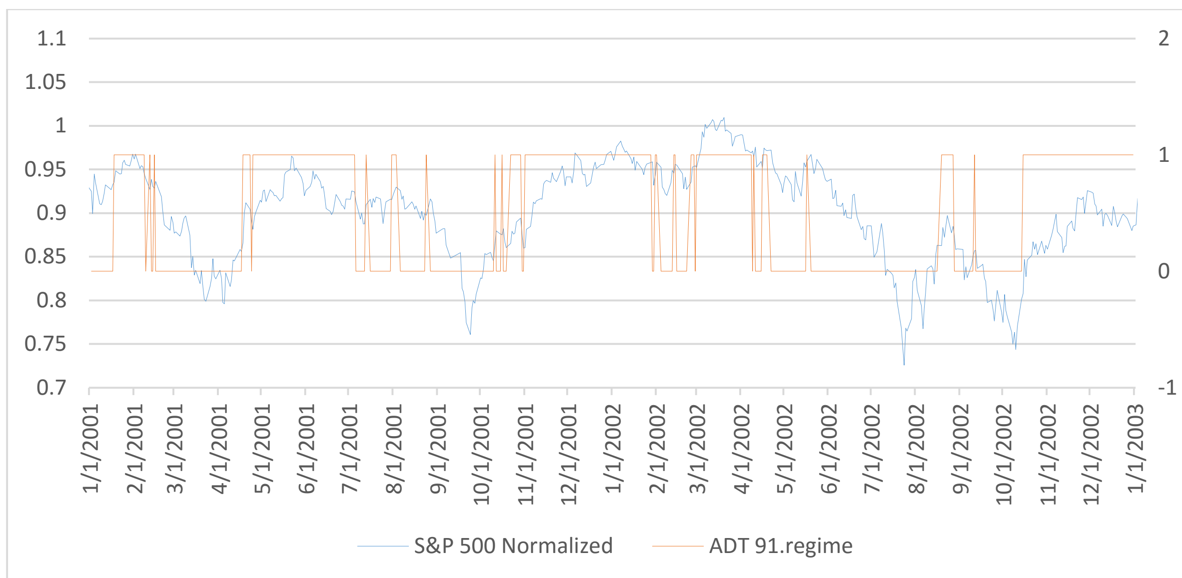



*Figure H29*. Regime predicted by best performing approach for 2001-2002 period without transaction costs, plotted against normalized target series.

```
main->
 (>
   (movingAverage
     (%
       (%
         (PeriodMaximum SP500.m250.1 246)
         (% 0.0617523580168530 8 0.5120859217289804))
       (periodMinimum SP500.m250.1
         (PeriodMaximum SP500.m250.1 1.1435868729905192))))
   (offsetValue SP500.m250.1 1.1435868729905192))
adf0->
 (-
   (periodMinimum SP500.m250.1
     (-
       (% arg1 arg1)
       (offsetValue SP500.m250.1 arg1))) arg1)
adf1->
 (+
   (periodMinimum SP500.m250.1
     (- arg1
       (offsetValue SP500.m250.1 arg1)))
   (Norm arg1
     (-
       (PeriodMaximum SP500.m250.1
         (- arg1 arg1)) arg1)))
```

*Figure H30.* Best performing ADF program for 2001-2002 period without transaction costs.

```
main->
 (and
   (<
     (*
       (- 59 233)
       (*
```

```
        (%
          (-
            (periodMinimum SP500.m250.1
              (- 0.56246175785108 178))
            (%
              (- 66 18)
              (movingAverage Offsetvalue 0)))
          (-
            (periodMinimum SP500.m250.1
              (* Offsetvalue 0 192))
            (periodMinimum SP500.m250.1 Offsetvalue 0))) 53))
    (PeriodMaximum SP500.m250.1
      (PeriodMaximum SP500.m250.1 239)))
  (or
    (and false
      (if true false true))
    (adf0
      (< 72 1.0370796085258633)
      (-
        (Norm
          (offsetValue SP500.m250.1
            (-
              (- 59 Offsetvalue 0)
              (- 0.56246175785108
                (- 59 Offsetvalue 0))))
          (Norm 0.6167750572429982
            (offsetValue SP500.m250.1
              (-
                (offsetValue SP500.m250.1 Offsetvalue 0) 25))))
        (Norm
          (offsetValue SP500.m250.1
            (-
              (- 59 Offsetvalue 0)
              (offsetValue SP500.m250.1 0.1137333582369704)))
          (Norm
            (*
              (periodMinimum SP500.m250.1 139)
              (PeriodMaximum SP500.m250.1 247))
            (offsetValue SP500.m250.1 Offsetvalue 0)))))))
adf0->
(and
  (or
    (or
      (< arg1 arg1)
      (or arg0 arg0))
    (>
      (* arg1 arg1)
      (Norm arg1 arg1)))
  (or
    (not
      (and arg0 arg0))
    (if
      (and
        (not
          (>
            (+ arg1 arg1)
            (+ arg1 1))) arg0)
      (or arg0 false)
      (not
        (and arg0 arg0)))))
adf1->
```

```
(+
  (+
    (offsetValue SP500.m250.1
      (*
        (movingAverage
          (* arg1 arg1))
        (+
          (movingAverage arg1)
          (% arg1 arg1))))
    (%
      (- 1 arg1)
      (Norm arg1 arg1)))
  (%
    (offsetValue SP500.m250.1
      (PeriodMaximum SP500.m250.1 arg1))
    (Norm
      (PeriodMaximum SP500.m250.1
        (%
          (offsetValue SP500.m250.1
            (PeriodMaximum SP500.m250.1
              (periodMinimum SP500.m250.1 arg1)))
          (offsetValue SP500.m250.1
            (*
              (movingAverage arg1)
              (+ arg1 arg1)))))
      (movingAverage arg1))))
```

*Figure H31.* Best performing ADT program for 2001-2002 period without transaction costs.

```
main->
(BinaryNumber
  (<
    (movingAverage
      (%
        (Norm 62
          (PeriodMaximum SP500.m250.1 1.9695511304638724))
        (offsetValue SP500.m250.1
          (periodMinimum SP500.m250.1 0.9792433541927676))))
    (periodMinimum SP500.m250.1
      (Norm
        (Norm
          (PeriodMaximum SP500.m250.1 1.9695511304638724)
          (movingAverage
            (periodMinimum SP500.m250.1 Offsetvalue 0)))
        (offsetValue SP500.m250.1
          (Norm 31
            (offsetValue SP500.m250.1
              (periodMinimum SP500.m250.1 0.9792433541927676))))))))
adf0->
(and
  (not
    (or arg0
      (> arg1 arg1)))
  (and
    (and
      (< arg1 arg1)
      (not arg0)) arg0))
adf1->
(Norm
  (Norm
    (Norm
      (Norm
        (% arg1 arg1)
        (* arg1 arg1)) arg1)
    (- arg1
      (offsetValue SP500.m250.1 arg1)))
  (-
    (Norm
      (- arg1 arg1)
      (movingAverage arg1))
    (PeriodMaximum SP500.m250.1
      (%
        (%
          (offsetValue SP500.m250.1
            (+ arg1 arg1))
          (+
            (+ arg1 arg1)
            (periodMinimum SP500.m250.1 arg1)))
        (movingAverage
          (+
            (% arg1 arg1)
            (periodMinimum SP500.m250.1 arg1)))))))
```

*Figure H32.* Regime predicted by best performing ADT program for 2001-2002 period without transaction costs.

***2003-2004 without transaction costs.***

Table H7. Best Return, 2003-2004, Without Transaction Costs

| Best Approach | ADT |
|---|---|
| Best Return | 0.6443 |
| Buy and Hold Return | 0.3332 |

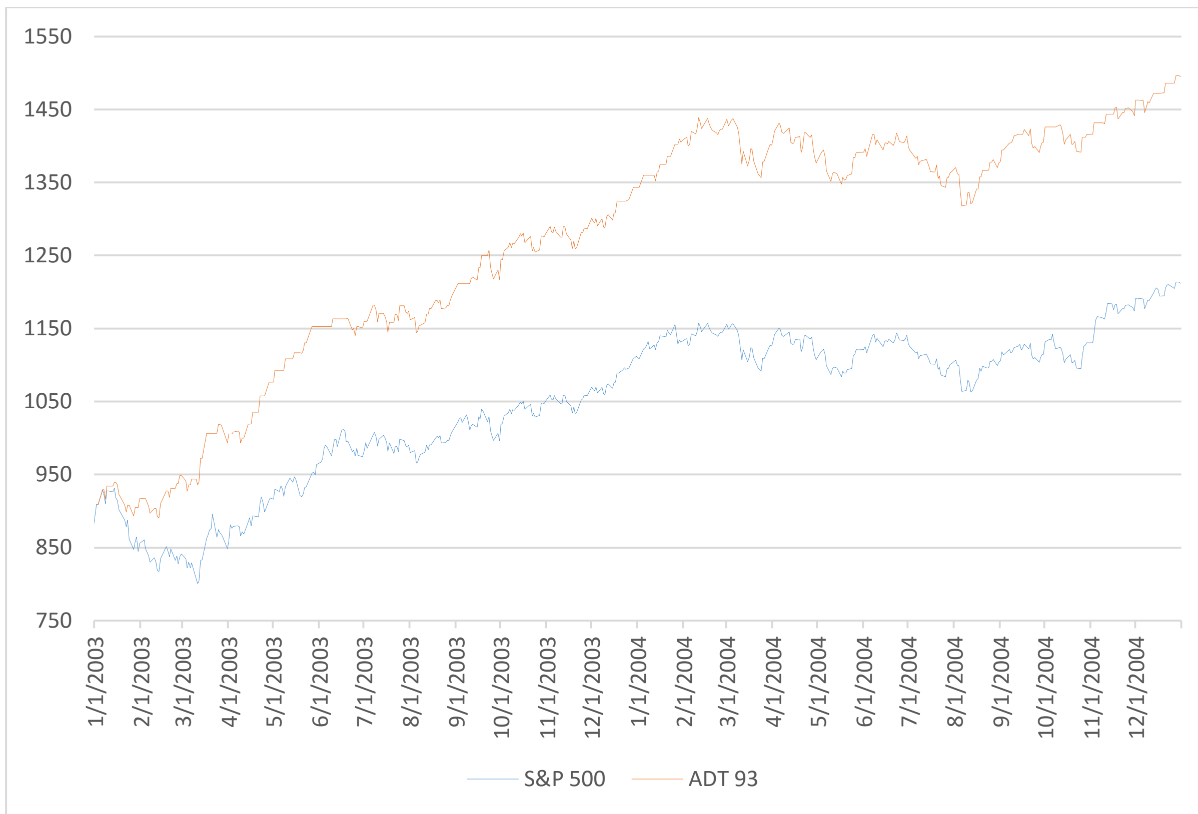



*Figure H33*. Best performing approach for 2003-2004 period without transaction costs.

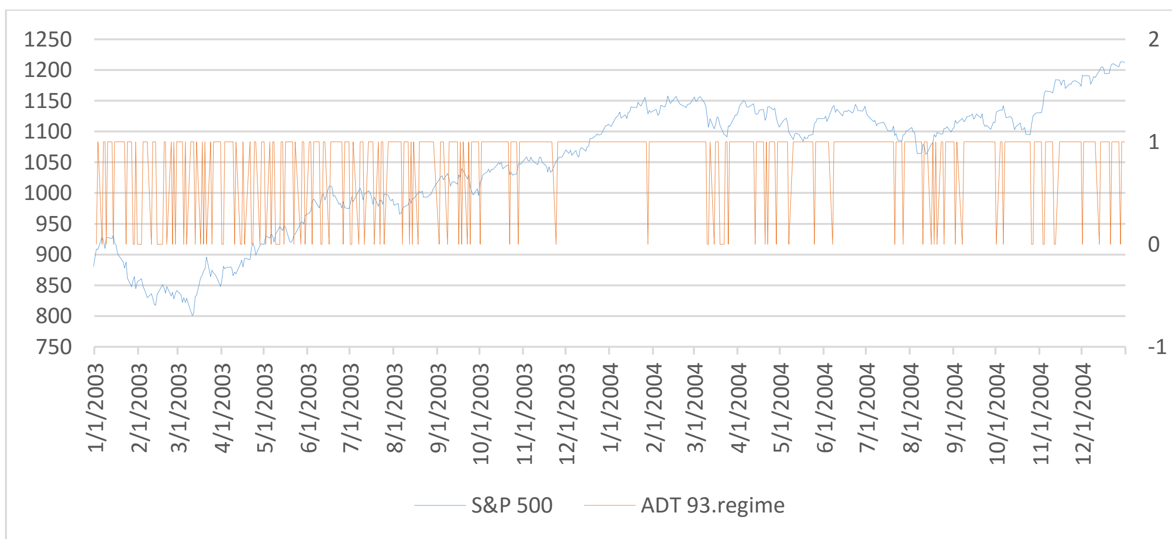



*Figure H34.* Regime predicted by best performing approach for 2003-2004 period without transaction costs.

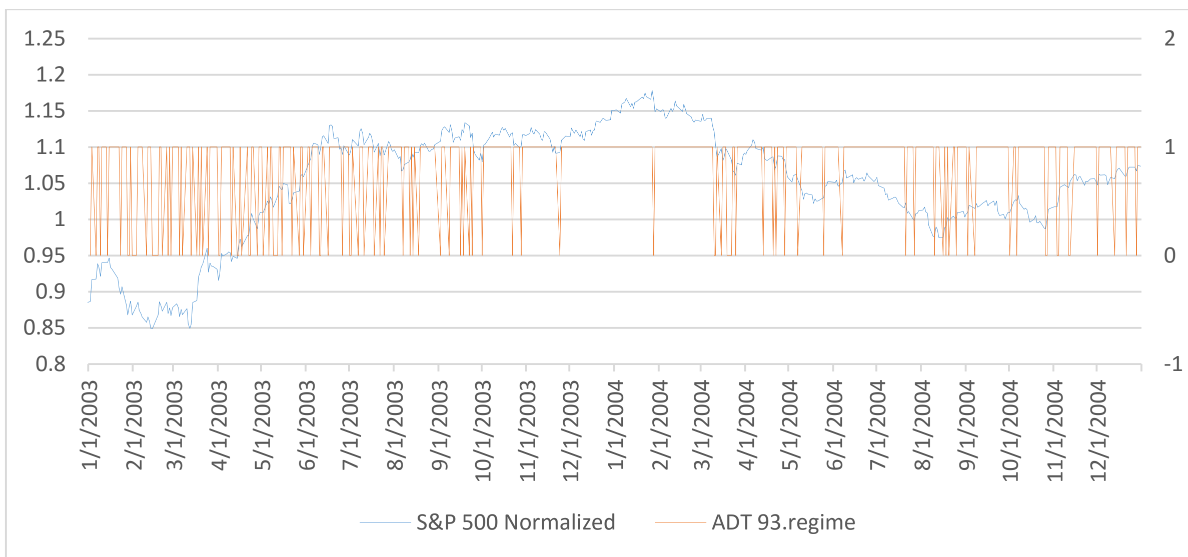



*Figure H35.* Regime predicted by best performing approach for 2003-2004 period without transaction costs, plotted against normalized target series.

```
main->
(and
  (and
    (or
      (> 1.0650254196845876 3)
      (> 199 Offsetvalue 0))
    (>
      (PeriodMaximum SP500.m250.1 42)
      (PeriodMaximum SP500.m250.1 Offsetvalue 0)))
  (or
    (adf0 true
      (movingAverage Offsetvalue 0))
    (not true)))
adf0->
(<
  (offsetValue SP500.m250.1
    (-
      (PeriodMaximum SP500.m250.1 arg1)
      (offsetValue SP500.m250.1
        (+ arg1 1))))
  (offsetValue SP500.m250.1
    (% arg1 arg1)))
adf1->
(%
  (%
    (+
      (Norm arg1 arg1)
      (Norm 0 arg1))
    (offsetValue SP500.m250.1
      (+ arg1 arg1)))
  (PeriodMaximum SP500.m250.1
    (PeriodMaximum SP500.m250.1
      (offsetValue SP500.m250.1
        (-
          (PeriodMaximum SP500.m250.1 arg1)
          (offsetValue SP500.m250.1 arg1))))))
```

*Figure H36.* . Best performing ADT program for 2003-2004 period without transaction costs.

```
main->
(BinaryNumber
  (if
    (if
      (if
        (< 0.5752406769227789
          (-
            (PeriodMaximum SP500.m250.1
              (offsetValue SP500.m250.1
                (offsetValue SP500.m250.1 0.6578518733450041)))
            (*
              (*
                (* Offsetvalue 0 147)
                (periodMinimum SP500.m250.1 Offsetvalue 0))
              (PeriodMaximum SP500.m250.1
                (PeriodMaximum SP500.m250.1 1.768925267784819)))))
        (or true false)
        (> 0.23028702202513518 Offsetvalue 0))
      (>
        (offsetValue SP500.m250.1 Offsetvalue 0)
        (PeriodMaximum SP500.m250.1 15))
      (not
        (or true false)))
    (or
      (<
        (adf1 false 208)
        (offsetValue SP500.m250.1 221))
      (not
        (or false
          (>
            (movingAverage
              (periodMinimum SP500.m250.1
                (% 1.973853119591221 0)))
            (Norm
              (periodMinimum SP500.m250.1
                (PeriodMaximum SP500.m250.1 51))
              (Norm
                (periodMinimum SP500.m250.1 0.8657650017727785)
                (Norm Offsetvalue 0 104)))))))
    (>
      (+
        (adf0 false 165)
        (periodMinimum SP500.m250.1 1.349232507728061))
      (movingAverage
        (periodMinimum SP500.m250.1 99)))))
adf0->
(%
  (movingAverage
    (%
      (- 0 arg1)
      (* arg1 arg1)))
  (+
    (*
      (offsetValue SP500.m250.1 0)
      (movingAverage arg1))
    (Norm arg1
      (- arg1 arg1))))
```

```
adf1->
(-
  (offsetValue SP500.m250.1
    (*
      (+ arg1
        (%
          (offsetValue SP500.m250.1 arg1)
          (+ arg1 arg1)))
      (movingAverage arg1)))
  (offsetValue SP500.m250.1
    (%
      (offsetValue SP500.m250.1 arg1)
      (+ arg1 arg1))))
```

*Figure H37*. Regime program for best performing ADT individual for 2003-2004 period without transaction costs.

***1999-2004 without transaction costs.***

Table H8. Best Return, 1999-2004, Without Transaction Costs

| | |
|---|---|
| Best Approach | ADT |
| Best Return | 0.6959 |
| Buy and Hold Return | -0.0189 |

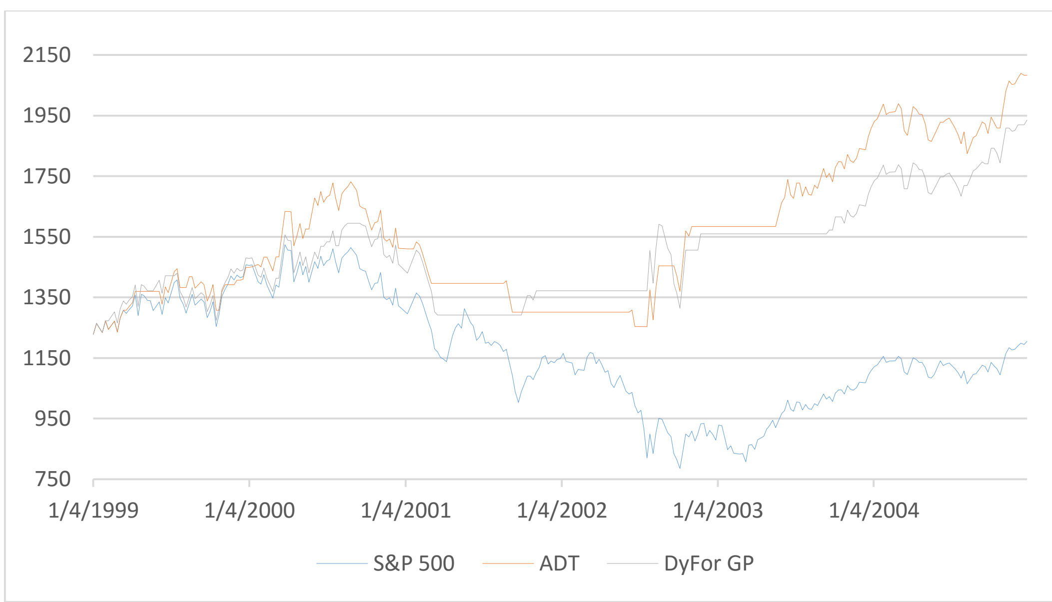



*Figure H38.* Best returns for 1999-2014 period without transaction costs.

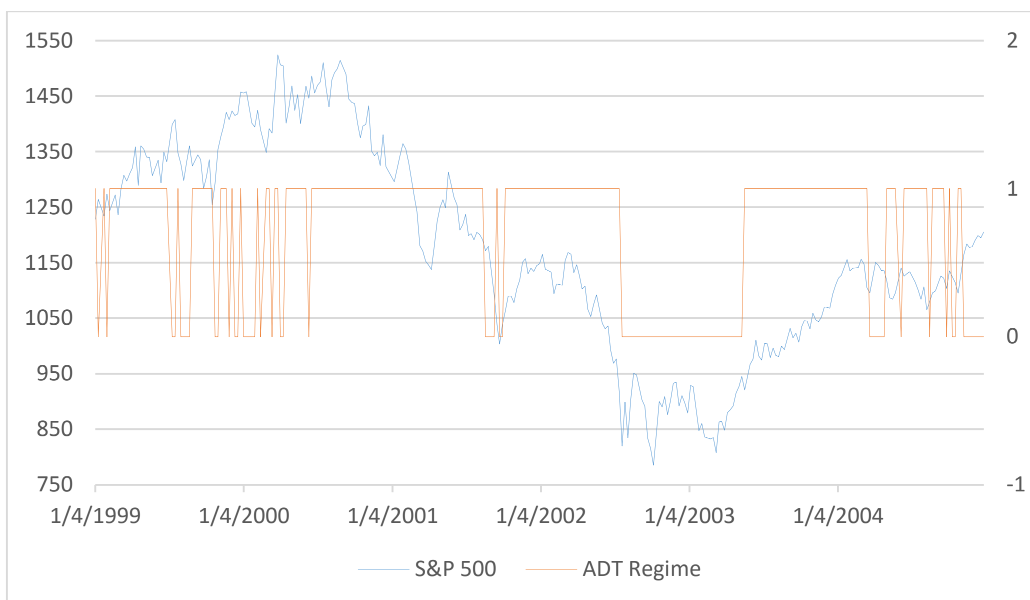



*Figure H39.* Regime predicted by best performing ADT individual for 1999-2004 period without transaction costs.

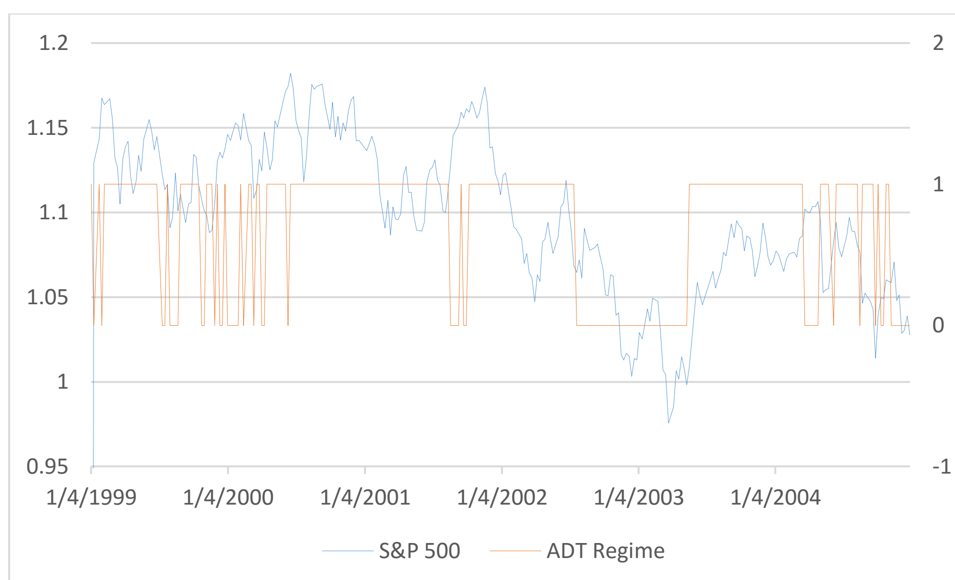



*Figure H40.* Regime predicted by best performing ADT individual for 1999-2004 period without transaction costs, plotted against normalized series.

# Appendix I

## Raw Data Files

Full data sets and parameter files from the experiments described in this report are available at http://www.infoblazer.com/geneticprogramming.

# References

Allen, F., & Karjalainen, R. (1999). Using genetic algorithms to find technical trading rules. *Journal of Financial Economics*, *51*(2), 245–271.

Amazon Web Services, I. (2014). Amazon EC2. Retrieved from http://aws.amazon.com/ec2/

American Association of Individual Investors. (2014). Sentiment Survey. Retrieved December 10, 2014, from http://www.aaii.com/sentimentsurvey

Angeline, P. (1993). *Evolutionary algorithms and emergent intelligence*. The Ohio State University.

Angeline, P. (1994). Genetic Programming and Emergent Intelligence. *Advances in Genetic Programming*, *1*, 75–98.

Angeline, P. J., & Pollack, J. (1993). Evolutionary module acquisition. In *The Second Annual Conference on Evolutionary Programming*. La Jolla, California.

Angeline, P., & Pollack, J. (1992). The Evolutionary Induction of Subroutines. In *Proceedings of the fourteenth annual conference of the cognitive science society* (pp. 236–241).

Atsalakis, G. S., & Valavanis, K. P. (2009). Surveying stock market forecasting techniques – Part II: Soft computing methods. *Expert Systems with Applications*, *36*(3), 5932–5941. doi:10.1016/j.eswa.2008.07.006

Banzhaf, W., Nordin, P., Keller, R. E., & Francone, F. D. (1998). *Genetic programming: an introduction*. San Francisco: Morgan Kaufmann.

Bleuler, S., Brack, M., Thiele, L., & Zitzler, E. (2001). Multiobjective Genetic Programming : Reducing Bloat Using SPEA2. In *Evolutionary Computation, 2001.* (Vol. 1, pp. 536–543). Seoul: Ieee. doi:10.1109/CEC.2001.934438

Brooks, C. (2014). *Introductory econometrics for finance [Kindle version]* (Third Edit.). Cambridge: Cambridge University Press.

Canelas, A., Neves, R., & Horta, N. (2012). A new SAX-GA methodology applied to investment strategies optimization. In *Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference - GECCO '12* (pp. 1055–1062). New York, New York, USA: ACM Press. doi:10.1145/2330163.2330310

Chen, S. H., Kuo, T. W., & Hoi, K. M. (2008). Genetic Programming and Financial Trading:How Much About "What We Know." In *Handbook of financial engineering* (pp. 99–154). Springer US. doi:10.1007/978-0-387-76682-9

Chen, S., & Yeh, C. (1997). Toward a computable approach to the efficient market hypothesis: an application of genetic programming. *Journal of Economic Dynamics and Control*, *21*, 1043–1063.

Chicago Board Options Exchange. (2014). CBOE - CBOE Volatility Index (VIX) Options and

Futures Micro Site. Retrieved December 10, 2014, from http://www.cboe.com/micro/VIX/vixintro.aspx

Dacco, R., & Satchell, S. (1999). Why do Regime-switching Models Forecast so Badly? *Journal of Forecasting*, *18*(1), 1–16.

Fama, E. F. (1965). The behavior of stock-market prices. *Journal of Business*, *38*(1), 34–105.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). *Design Patterns: Elements of Reusable Object-Oriented Software*. Reading: Addison-Wesley.

Genetic Programming: A tutorial with the software simple GP. (2002).

Hamilton, J. (1989). A new approach to the economic analysis of nonstationary time series and the business cycle. *Econometrica: Journal of the Econometric Society*, *57*(2), 357–384.

Hamilton, J. (2008). Regime-switching models. *The New Palgrave Dictionary of Economics*, 1–15.

Hellstrom, T., & Holmstrom, K. (1999). Parameter tuning in trading algorithms using ASTA. *Computational Finance*, 1–15.

Hénon, M. (1976). A two-dimensional mapping with a strange attractor. *Communications in Mathematical Physics*, *77*, 69–77.

Hickey, R. (2014). Clojure Web Site. Retrieved from http://clojure.org

Johnson, R. (1997). Frameworks=(components+ patterns). *Communications of the ACM*, *40*(10), 39–42.

Jones, C. M. (2002). *A century of stock market liquidity and trading costs*. *Columbia University working paper*. doi:10.2139/ssrn.313681

Jong, E. de, Watson, R., & Pollack, J. (2001). Reducing bloat and promoting diversity using multi-objective methods. In *Genetic and Evolutionary Computation Conference (GECCO-2001)* (pp. 11 –18). San Francisco.

Kaboudan, M. (2000). Genetic programming prediction of stock prices. *Computational Economics*, *16*(1988), 207–236. doi:10.1023/A:1008768404046

Kaboudan, M. A. (1998). A GP Approach to Distinguish Chaotic from Noisy Signals. In *Genetic Programming 1998: Proceedings of the Third Annual Conference* (pp. 187–191).

Kampouridis, M., & Tsang, E. (2010). EDDIE for Investment Opportunities Forecasting: Extending the Search Space of the GP. In *2010 IEEE Conference on Evolutionary Computation (CEC)* (pp. 1–8). IEEE.

Kampouridis, M., & Tsang, E. (2012). Investment Opportunities Forecasting: Extending the Grammar of a GP-based Tool. *International Journal of Computational Intelligence Systems*, *5*, 530–541. doi:10.1080/18756891.2012.696918

Keane, M. A., Streeter, M. J., Mydlowec, W., Lanza, G., & Yu, J. (2006). *Genetic programming IV: Routine human-competitive machine intelligence (Vol. 5)*. Springer.

Kinnear Jr, K. E. (1994). Alternatives in Automatic Function Definition: A Comparison of Performance. In *Advances in genetic programming* (pp. 119–141).

Koza, J. ., Keane, M. A., Streeter, M. J., Mydlowec, W., Yu, J., & Lanza, G. (2006). *Genetic programming IV: Routine human-competitive machine intelligence (Vol. 5)*. Springer.

Koza, J. R. (1990). *Genetic Programming : A Paradigm for Genetically Breeding Populations of Computer Programs to Solve Problems*. Stanford University, Department of Computer Science.

Koza, J. R. (1992). *Genetic programming: on the programming of computers by means of natural selection (Vol. 1)*. MIT press.

Koza, J. R. (1994). *Genetic programming II: automatic discovery of reusable programs*. MIT press.

Koza, J., Streeter, M., & Keane, M. (2008). Routine high-return human-competitive automated problem-solving by means of genetic programming. *Information Sciences*, *178*(23), 4434–4452. doi:10.1016/j.ins.2008.07.028

Li, J. (2000). *FGP: A genetic programming based financial forecasting tool*. University of Essex.

Li, J., & Tsang, E. (1999). Investment decision making using FGP: A case study. In *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on (Vol. 2)* (Vol. 5). IEEE.

Mackey, M. C., & Glass, L. (1977). Oscillation and Chaos in Physiological Control Systems. *Science*, *197*(4300), 287–289.

Mahfoud, S., & Mani, G. (1996). Financial forecasting using genetic algorithms. *Applied Artificial Intelligence*, *10*(6), 543–566. doi:10.1080/088395196118425

Montana, D. J. (1995). Strongly Typed Genetic Programming. *Evolutionary Computation*, *3*(2), 199–230. doi:10.1162/evco.1995.3.2.199

Mulloy, B., Riolo, R., & Savit, R. (1996). Dynamics of genetic programming and chaotic time series prediction. In *Proceedings of the first annual conference on genetic programming* (pp. 166–174). MIT Press.

O'Neill, M., Vanneschi, L., Gustafson, S., & Banzhaf, W. (2010). Open issues in genetic programming. *Genetic Programming and Evolvable Machines*, *11*(3-4), 339–363. doi:10.1007/s10710-010-9113-2

Pivotal Software. (2016). Spring Boot. Retrieved from http://projects.spring.io/spring-boot/

Poli, R., Vanneschi, L., Langdon, W. B., & McPhee, N. F. (2010). Theoretical results in genetic programming: the next ten years? *Genetic Programming and Evolvable Machines*, *11*(3-4), 285–320. doi:10.1007/s10710-010-9110-5

Quandl. (2016). Quandl. Retrieved from https://www.quandl.com

Robinhood Financial, L. (2016). Robinhood - Free Stock Trading. Retrieved January 1, 2016, from https://www.robinhood.com/

S&P Dow Jones Indices LLC. (2016). S&P 500® - S&P Dow Jones Indices. Retrieved from http://us.spindices.com/indices/equity/sp-500

Schmidt, D. C., Fayad, M., & Johnson, R. E. (1996). Software patterns. *Communications of the ACM*, *39*(10), 37–39. doi:10.1145/236156.236164

Securities and Exchange Commission. (2001). Order Directing the Exchanges and NASD to Submit a Decimalization Implementation Plan. Release No. 34-42360/January 28, 2000.

Sincere, M. (2011). *All About Market Indicators*. McGraw-Hill.

Spector, L. (1995). Evolving Control Structures with Automatically De ned Macros, 99–105.

Srinivas, M., & Patnaik, L. M. (1994). Genetic algorithms: a survey. *Computer*, *27*(6), 17–26. doi:10.1109/2.294849

Tsang, E. P. K., & Butler, J. M. (1998). EDDIE Beats the Bookies, (April).

Turing, a. M. (1950). Computing Machinery and Intelligence. *Mind*, *LIX*(236), 433–460. doi:10.1093/mind/LIX.236.433

Verma, A., Llorà, X., Goldberg, D. E., & Campbell, R. H. (2009). Scaling Genetic Algorithms Using MapReduce. In *Ninth International Conference on Intelligent Systems Design and Applications, 2009. ISDA'09.* (pp. 13–18). IEEE. doi:10.1109/ISDA.2009.181

Wagner, N. (2005). *Time Series Forecasting for Non-static Environments: the DyFor Genetic Program Model*. the University of North Carolina at Charlotte.

Wagner, N., Khouja, M., Michalewicz, Z., & McGregor, R. R. (2008). Forecasting economic time series with the DyFor genetic program model. *Applied Financial Economics*, *18*(5), 357–378. doi:10.1080/09603100600949200

Wagner, N., & Michalewicz, Z. (2008). An Analysis of Adaptive Windowing for Time Series Forecasting in Dynamic Environments: Further Tests of The DyFor GP Model. In *Proceedings of the 10th annual conference on Genetic and evolutionary computation* (pp. 1657–1664). Atlanta: ACM.

Wagner, N., Michalewicz, Z., Khouja, M., & McGregor, R. R. (2007). Time Series Forecasting for Dynamic Environments: The DyFor Genetic Program Model. *IEEE Transactions on Evolutionary Computation*, *11*(4), 433–452. doi:10.1109/TEVC.2006.882430

Weizenbaum, J. (1966). ELIZA---a computer program for the study of natural language communication between man and machine. *Communications of the ACM*, *9*(1), 36–45. doi:10.1145/365153.365168

Woodward, J. R. (2003). Modularity in Genetic Programming. In C. Ryan, T. Soule, M. Keijzer, E. Tsang, R. Poli, & E. Costa (Eds.), *Genetic Programming* (pp. 254–263). Berlin Heidelberg: Springer. doi:http://dx.doi.org/10.1007/3-540-36599-0_23

Yu, T., Chen, S.-H., & Kuo, T.-W. (2005). Discovering financial technical trading rules using genetic programming with lambda abstraction. In *Genetic programming theory and practice II* (pp. 11–30). Springer US.

Yu, T., & Clack, C. (1998). Recursion, lambda-abstractions and genetic programming. *Cognitive Science Research Papers-University Of Birmingham*, 26–30.

Zhang, G. P. (2003). Time series forecasting using a hybrid ARIMA and neural network model. *Neurocomputing*, *50*, 159–175. doi:10.1016/S0925-2312(01)00702-0

Zhang, G., Patuwo, B. E., & Hu, M. (1998). Forecasting with artificial neural networks:: The state of the art. *International Journal of Forecasting*, *14*, 35–62.