Genetic Programming Hyper-heuristics for Job Shop Scheduling

by

Rachel Joanne Hunt

A thesis
submitted to the Victoria University of Wellington
in fulfilment of the
requirements for the degree of
Doctor of Philosophy
in Operations Research.

Victoria University of Wellington
2016

Abstract

Scheduling problems arise whenever there is a choice of order in which a number of tasks should be performed; they arise commonly, daily and everywhere. A job shop is a common manufacturing environment in which a schedule for processing a set of jobs through a set of machines needs to be constructed. Job shop scheduling (JSS) has been called a fascinating challenge as it is computationally hard and prevalent in the real-world. Developing more effective ways of scheduling jobs could increase profitability through increasing throughput and decreasing costs. Dispatching rules (DRs) are one of the most popular scheduling heuristics. DRs are easy to implement, have low computational cost, and cope well with the dynamic nature of real-world manufacturing environments. However, the manual development of DRs is time consuming and requires expert knowledge of the scheduling environment. Genetic programming (GP) is an evolutionary computation method which is ideal for automatically discovering DRs. This is a hyper-heuristic approach, as GP is searching the search space of heuristic (DR) solutions rather than constructing a schedule directly.

The overall goal of this thesis is to develop GP based hyper-heuristics for the efficient evolution (automatic generation) of robust, reusable and effective scheduling heuristics for JSS environments, with greater interpretability.

Firstly, this thesis investigates using GP to evolve optimal DRs for the static two-machine JSS problem with makespan objective function. The results show that some evolved DRs were equivalent to an optimal scheduling algorithm. This validates both the GP based hyper-heuristic approach for generating DRs for JSS and the representation used.

Secondly, this thesis investigates developing "less-myopic" DRs through the use of wider-looking terminals and local search to provide additional fitness information. The results show that incorporating features of the state of the wider shop improves the mean performance of the best evolved DRs, and that the inclusion of local search in evaluation evolves DRs which make better decisions over the local time horizon, and attain lower total weighted tardiness.

Thirdly, this thesis proposes using strongly typed GP (STGP) to address the challenging issue of interpretability of DRs evolved by GP. Several grammars are investigated and the results show that the DRs evolved in the semantically constrained search space of STGP do not have (on average) performance that is as good as unconstrained. However, the interpretability of evolved rules is substantially improved.

Fourthly, this thesis investigates using multiobjective GP to encourage evolution of DRs which are more readily interpretable by human operators. This approach evolves DRs with similar performance but smaller size. Fragment analysis identifies popular combinations of terminals which are then used as high level terminals; the inclusion of these terminals improved the mean performance of the best evolved DRs.

Through this thesis the following major contributions have been made: (1) the first use of GP to evolve optimal DRs for the static two-machine job shop with makespan objective function; (2) an approach to developing less-myopic DRs through the inclusion of wider looking terminals and the use of local search to provide additional fitness information over an extended decision horizon; (3) the first use of STGP for the automatic discovery of DRs with better interpretability and semantic validity for increased trust; and (4) the first multiobjective GP approach that considers multiple objectives investigating the trade-off between scheduling behaviour and interpretability. This is also the first work that uses analysis of evolved GP individuals to perform feature selection and construction for JSS.

List of Publications

Hunt, R., Johnston, M. and Zhang, M. "Evolving machine-specific dispatching rules for a two-machine job shop using genetic programming". In *Proceedings of the IEEE Congress on Evolutionary Computation* (2014), pp. 618–625.

Hunt, R., Johnston, M. and Zhang, M. "Evolving "less-myopic" scheduling rules for dynamic job shop scheduling using genetic programming". In *Proceedings of the Genetic and Evolutionary Computation Conference* (2014), pp. 927–934. (Best paper nomination - GP Track).

Hunt, R., Johnston, M. and Zhang, M. "Using local search to evaluate dispatching rules in dynamic job shop scheduling". In *Proceedings of Evolutionary Computation in Combinatorial Optimization* (2015), pp. 222–233.

Hunt, R., Johnston, M. and Zhang, M. "Evolving dispatching rules with greater understandability for dynamic job shop scheduling". Submitted to *Evolutionary Computation Journal (MIT Press)* (2015), 36pp. (under second review).

Hunt, R., Johnston, M. and Zhang, M. "Multiobjective genetic programming for feature selection and dispatching rule generation in job shop scheduling". Submitted to *IEEE Transactions on Cybernetics.* (2015), 14pp. (under review).

Acknowledgements

I would like to thank my supervisors, Prof Mengjie Zhang and Dr Mark Johnston, for their guidance, support and encouragement throughout the course of my PhD study. Thank you for the hours of reading of my drafts of papers and of this thesis, and for the carefully thought out red pen which they have had on their return. Thank you for challenging me to dig deeper and question why. I am especially grateful to Dr Mark Johnston for continuing to supervise the last half of this thesis from his new position in the UK.

I am grateful for the financial support of the Victoria Doctoral Scholarship and the Victoria Doctoral Submission Scholarship. I have enjoyed being part of the Evolutionary Computation Research Group (ECRG) and Evolutionary Computation for Scheduling and Combinatorial Optimisation subgroup.

I am incredibly blessed to have such a caring and supportive family. Thank you to my amazing husband Simon for listening to my frustrations and lovingly motivating me to keep going. Thank you to my fantastic parents, Andrew and Janice, for the coffee dates when they were needed, for listening to my research ideas, and for celebrating in my successes along the way. Thank you to my great friend Claire, for her prayerful support and encouragement.

Contents

List of Publications **iii**

Acknowledgements **v**

List of Tables **xiii**

List of Figures **xxi**

List of Notations **xxix**

List of Abbreviations **xxxi**

Glossary of Terms **xxxiii**

1 Introduction **1**

1.1 Problem Statement 1

1.2 Motivations 3

1.2.1 Challenges of Scheduling 3

1.2.2 Why Genetic Programming? 5

1.2.3 Limitations of Existing Work 7

1.3 Research Goals 9

1.4 Major Contributions 16

1.5 Organisation of Thesis 19

2 Literature Review 23
2.1 Introduction . . . 23
2.1.1 Scheduling . . . 24
2.1.2 Heuristics, Meta-heuristics and Hyper-heuristics . . . 25
2.1.3 Evolutionary Computation . . . 27
2.2 Job Shop Scheduling . . . 29
2.2.1 The Job Shop Model . . . 31
2.2.2 Scheduling Algorithms for Static JSS . . . 34
2.2.3 Scheduling Methods in Dynamic Environments . . . 40
2.3 Genetic Programming . . . 43
2.3.1 Representation . . . 44
2.3.2 Population Initialisation . . . 45
2.3.3 Evaluation . . . 46
2.3.4 Selection . . . 47
2.3.5 Genetic Operators . . . 47
2.3.6 Basic GP Algorithm . . . 49
2.3.7 Multiobjective Genetic Programming . . . 51
2.3.8 Semantic Constraint in GP . . . 53
2.4 Local Search . . . 55
2.5 Feature Manipulation . . . 56
2.5.1 GP based Feature Manipulation . . . 56
2.5.2 Feature Manipulation for Scheduling . . . 57
2.6 GP for Generating Heuristics for JSS . . . 58
2.6.1 Genetic Programming for Automatic Generation of DRs . . . 59
2.6.2 Other Evolutionary Computation based Methods for Scheduling . . . 69
2.7 Chapter Summary . . . 70

3 Dispatching Rules for Two-Machine Job Shop Scheduling 73
3.1 Introduction . . . 73
3.1.1 Chapter Organisation . . . 74

3.2 Optimal DR Representation for Static JSS 74
3.3 Can GP Evolve Optimal DRs for Static JSS? 80
3.3.1 GP System for Static Two-Machine JSS 80
3.3.2 Initial Experimental Results on Static Problems 83
3.3.3 Improvements through Parameter Settings 84
3.3.4 Improvements in Representation 85
3.4 Machine Specific DRs for Dynamic JSS 87
3.4.1 Representation of GP Individuals for Machine Specific DRs 88
3.4.2 GP System for Dynamic JSS 88
3.4.3 Dynamic Two-Machine JSS Experimental Results . . 94
3.4.4 Analysis of Evolved Programs 98
3.5 Chapter Summary 101

4 Automatic Discovery of "Less-Myopic" Dispatching Rules 105
4.1 Introduction 105
4.1.1 Chapter Goals 106
4.1.2 Chapter Organisation 107
4.2 Less-Myopic Feature Sets 107
4.2.1 Terminal and Function Sets 107
4.2.2 GP System Setup 110
4.2.3 Experimental Results 112
4.2.4 Analysis of Evolved DRs 118
4.2.5 Feature Set Conclusions 119
4.3 Using Local Search to Evaluate DRs 122
4.3.1 Local Search based Evaluation 124
4.3.2 Initial Experiment 130
4.3.3 Extended Experiments 135
4.3.4 Experimental Results 138
4.3.5 Local Search Conclusions 147
4.4 Local Search to Investigate Tie Breaking 150
4.4.1 Local Search Fitness for Tie Breaking 151

4.4.2 GP System Setup 152
4.4.3 Results 154
4.5 Chapter Summary 154

5 Evolving Dispatching Rules with Greater Understandability 159
5.1 Introduction 159
5.1.1 Chapter Goals 161
5.1.2 Chapter Organisation 162
5.2 Method and Experimental Design 163
5.2.1 GP System for Automatic Generation of DRs 163
5.2.2 Job Shop Scenarios 163
5.2.3 Function and Terminal Sets 166
5.2.4 GP Parameter Settings 167
5.3 Experimental Results 169
5.3.1 Stage One: Arithmetic GP and STGP 170
5.3.2 Stage Two: Inclusion of If, Max and Min 179
5.3.3 Stage Three: Specialised Conditionals 187
5.4 Analysis of Evolved Rules 192
5.4.1 Fragment Analysis 195
5.4.2 Best Evolved DRs 199
5.5 Further Discussions 209
5.5.1 DR Size 211
5.5.2 Comparison of Performance in Testing vs Extreme Testing 214
5.5.3 Training and Testing Times 216
5.5.4 Exploration of Performance Decrease in STGP 216
5.6 Chapter Summary 218

6 Multiobjective Genetic Programming for Feature Selection and Dispatching Rule Generation in Job Shop Scheduling 223
6.1 Introduction 223
6.1.1 Chapter Goals 225

6.1.2 Chapter Organisation . . . 226
6.2 MOGP for Evolving Interpretable DRs . . . 227
6.2.1 Representation of Semantically Constrained DRs . . . 227
6.2.2 Multiobjective Fitness . . . 229
6.2.3 Job Shop Simulation Model . . . 229
6.2.4 Benchmark Methods for Comparison . . . 232
6.3 Stage One: Initial Investigation . . . 233
6.4 Stage Two: Higher-level Constructed Terminals . . . 236
6.5 Stage Three: Specialised Conditional Operators . . . 239
6.6 Stage Four: Feature Reduction . . . 244
6.7 Further Analysis . . . 247
6.7.1 Comparison of Results by Stages . . . 247
6.7.2 Extremes . . . 252
6.7.3 Aggregate Pareto Fronts . . . 255
6.7.4 Trade-Offs between Pairs of Objectives . . . 257
6.7.5 Attribute Sets . . . 262
6.8 Chapter Summary . . . 264

7 Conclusions 267
7.1 Achieved Objectives . . . 267
7.2 Main Conclusions . . . 270
7.2.1 Optimal Dispatching Rules . . . 270
7.2.2 Less-Myopic Dispatching Rules . . . 272
7.2.3 Interpretability of Evolved Dispatching Rules . . . 274
7.2.4 Multiobjective GP for Interpretable DRs . . . 275
7.2.5 Feature Manipulation in JSS using GP . . . 277
7.3 Future Work . . . 278

Bibliography 283

List of Tables

1.1 Mapping of contributions (Contrib) to research objectives (RO). 16

2.1 Processing times and paths of 20 jobs for $J2\,|\,|\,C_{max}$. 37

3.1 GP terminal set for static two-machine JSS problem with makespan objective GP system. 76

3.2 Evaluation of priorities of jobs J1 to J20 using the dispatching rule in Figure 3.1. T indicates `(- PR (- RT PR))` is greater than 0 (i.e. `(- PR (- RT PR))`>0 is true) and F indicates that `(- PR (- RT PR))` is less than or equal to 0. 78

3.3 Configurations of proportions of job types for static two-machine training problem instances. 83

3.4 Configurations of proportions of job types for training scenarios (TC1–TC4) and testing scenarios (C1–C14) for dynamic two-machine job shop problem instances. 92

3.5 GP terminal set for dynamic two-machine GP system. 93

3.6 Approximate order of performance of methods on each scenario of dynamic two-machine job shop. 97

3.7 Comparison of mean performance, mean$\pm$standard deviation of training/evolution and testing times of the four GP methods over 40 runs and three benchmark dispatching rules for dynamic two-machine job shop. 98

4.1 Terminal set used in GP system for dynamic ten-machine job shop. 108

4.2 Configurations for training and testing scenarios (mean of processing time distribution, expected utilisation and due date tightness) for dynamic ten-machine job shop problem instances. 111

4.3 Mean and standard deviation of total weighted tardiness for ten-machine job shop. 116

4.4 Mean of average and mean of standard deviation of queue length for dynamic ten-machine job shop. 117

4.5 Mean and standard deviation of training and testing times for dynamic ten-machine job shop. 118

4.6 Properties of queued jobs at machine M3 in the example of Figure 4.6. 128

4.7 Configuration of training, testing and extreme testing scenarios for local search based evaluation experiments (mean of processing time distribution, expected utilisation, due date tightness and operation setting). 133

4.8 Configuration of training, testing and extreme testing scenarios (processing time distribution, expected utilisation, due date tightness and operation setting) for extended local search based fitness experiments. Here “unif” is the discrete uniform(1,49) distribution, “geom” is the Geometric(0.04) distribution, F indicates “full” operations and V indicates “variable” operations. 137

4.9 Configuration of training and testing scenarios (mean of processing time distribution, expected utilisation, due date tightness parameter and operation setting) for local search based tie breaking. 153

5.1 Configuration of training, testing and extreme testing scenarios (processing time distribution, expected utilisation, due date tightness and operation setting). Here “unif” is the discrete uniform(1,49) distribution, “geom” is the Geometric(0.04) distribution, F indicates “full” operations and V indicates “variable” operations. 164

5.2 Allowable interactions for each operator in $\{*, \%, +, -, \max, \min\}$ for Grammar 1. Where interaction is allowed the type of the result is given, and the cell is coloured black if the interaction is not allowed. Terminals are split into four different type categories: C represents counts, X represents job weight (and other ratios), D represents time durations and T represents absolute (clock) times. 168

5.3 Grammar 1 for STGP. This is the grammar based representation of the allowable interactions for the arithmetic operators and four basic types presented in Table 5.2. The four types representing the initial terminal set, X, C, D and T are used. Any interaction which returns type I is not allowable in this basic grammar. 171

5.4 Allowable interactions for each operator in $\{*, \%, +, -, \max, \min\}$ for Grammar 1A. Where interaction is allowed the type of the result is given, and the cell is coloured black if the interaction is not allowed. Terminals are split into five different type categories: C, X, D, T, and new type I which represents inverse time durations. 173

5.5 Mean and standard deviation of total weighted tardiness on test problem instances of Stage 1 (lower values are better). Values that are statistically significantly better than all other GP and STGP methods across the same row across Tables 5.5, 5.9 and 5.11 are in **bold**. Where there are two values in a row that are bold, they are not statistically significantly different from each other, but are statistically significantly better than all remaining methods. 177

5.6 Mean and standard deviation of total weighted tardiness on extreme test problem instances of Stage 1 (lower values are better). Values that are statistically significantly better than all others across the same row across Tables 5.6, 5.10 and 5.12 are in **bold**. Where there are two values in a row that are bold, they are not statistically significantly different from each other, but are statistically significantly better than all remaining methods. 178

5.7 Allowable interactions for each operator in $\{\times, \%, +, -, \max, \min\}$ for Grammar 1S. Where interaction is allowed the type of the result is given, and the cell is coloured black if the interaction is not allowed. Terminals are split into six different type categories: C, X, D, T, I, and new type S which represents squared time durations. 180

5.8 Allowable interactions for each operator in $\{\times, \%, +, -, \max, \min\}$ for Grammar 1F. Where interaction is allowed the type of the result is given, and the cell is coloured black if the interaction is not allowed. Terminals are split into seven different type categories: C, X, D, T, I, S and new type F which represents inverse-square-durations. 181

5.9 Mean and standard deviation of total weighted tardiness on test problem instances of Stage 2 (lower values are better). Values that are statistically significantly better than all others across the same row across Tables 5.5, 5.9 and 5.11 are in **bold**. Where there are two values in a row that are bold, they are not statistically significantly different from each other, but are statistically significantly better than all remaining methods. 185

5.10 Mean and standard deviation of total weighted tardiness on extreme test problem instances of Stage 2 (lower values are better). Values that are statistically significantly better than all others across the same row across Tables 5.6, 5.10 and 5.12 are in **bold**. Where there are two values in a row that are bold, they are not statistically significantly different from each other, but are statistically significantly better than all remaining methods. 186

5.11 Mean and standard deviation of total weighted tardiness on test problem instances of Stage 3 (lower values are better). Values that are statistically significantly better than all others across the same row across Tables 5.5, 5.9 and 5.11 are in **bold**. Where there are two values in a row that are bold, they are not statistically significantly different from each other, but are statistically significantly better than all remaining methods. 190

5.12 Mean and standard deviation of total weighted tardiness on extreme test problem instances of Stage 3 (lower values are better). Values that are statistically significantly better than all others across the same row across Tables 5.6, 5.10 and 5.12 are in **bold**. Where there are two values in a row that are bold, they are not statistically significantly different from each other, but are statistically significantly better than all remaining methods. 191

5.13 Number of occurrences of functions and terminals in rules and number of rules functions and terminals occur in for Stage 1 methods. 193

5.14 Number of occurrences of functions and terminals in rules and number of rules functions and terminals occur in for Stage 2 methods. 194

5.15 Number of occurrences of functions and terminals in rules and number of rules functions and terminals occur in for Stage 3 methods. 196

5.16 Number of occurrences of fragments of depth two for Stage 1 methods. We have combined (op A B) and (op B A) and present the top 10 most common fragments for each method. 197

5.17 Number of occurrences of fragments of depth two for Stage 2 methods. We have combined (op A B) and (op B A) and present the top 10 most common fragments for each method. 198

5.18 Number of occurrences of fragments of depth two for Stage 3 methods. We have combined (op A B) and (op B A) and present the top 10 most common fragments for each method. 200

5.19 Mean and standard deviation of evolution time and test time for GP and STGP. 217

6.1 Terminal set used in the GP system with type. 228

6.2 Stage 1. Top six most frequently appearing fragments in Pareto-fronts of MOGP$_{1,N}$ and MOGP$_{1,N}$, and top six most frequently appearing fragments in best-of-run DRs of SOGP$_1$ at Stage 1. The number of times the fragment appears and the number of rules the fragment appears in are given for each. 237
6.3 Higher-level constructed terminals introduced at Stage 2. . . 238
6.4 Occurrence of each function as a percentage of all function occurrences in DRs evolved by each method, and occurrence of each terminal as a percentage of all terminal occurrences in DRs evolved by each method. 246
6.5 Table of mean $\pm$ standard deviation of evolution times in minutes for each method. 252

List of Figures

1.1 Overall structure of the thesis contribution chapters. 19

2.1 Gantt Chart showing optimal processing schedules at machines A and B for the example static two-machine job shop problem instance in Table 2.1. 37

2.2 GP syntax tree of $((2 * (x + 1)) - y)$. 45

2.3 GP tree initialisation methods, under the ramped-half-and-half method trees of these shapes are able to be generated for any maximum depth greater than three. 46

2.4 GP subtree mutation. 47

2.5 GP subtree crossover. 49

3.1 An optimal dispatching rule for the static two-machine job shop with makespan objective, represented using tree-based GP. 77

3.2 An optimal dispatching rule for the static two-machine job shop, using the `NPR` terminal. 85

3.3 An optimal dispatching rule for the static two-machine job shop evolved by GP. 86

3.4 Representations of tree-based GP dispatching rules for the dynamic two-machine GP system. 89

3.5 Boxplots showing results (lower TWT is better) of methods R1S, R1C, R2S and R2C across the 14 configurations for dynamic two-machine job shop (vertical scales differ between subfigures). 95
3.6 Best evolved R1C individuals for dynamic two-machine job shop. 99
3.7 Best evolved R2C individual for dynamic two-machine job shop. 100

4.1 Violin plots presenting results (TWT) of test scenarios for NGP-4op, LMGP-4op, NGP-8op and LMGP-8op on ten-machine dynamic job shop. Note that the vertical axis scales are different in each subfigure. 114
4.2 One of the best performing LMGP-4op DRs on the test scenarios. 118
4.3 One of the best performing LMGP-8op DRs on the test scenarios. 120
4.4 One of the best performing NGP-8op DRs on the test scenarios. 121
4.5 Diagram of evaluation process when a machine becomes available to process a job. 125
4.6 Example showing a full ten-machine job shop at a particular scheduling point (time= 0). 127
4.7 Example showing next machines visited by jobs at machine M3 under different ordering of jobs at machine M3. 128
4.8 Aggregate non-dominated front on problem instance from T1. 134
4.9 Aggregate non-dominated front on problem instance from XT1. 134
4.10 Aggregate non-dominated fronts from MOMF, MOSW and MOTR, and best-of-run individuals from BMMF, BMSW and BMTR on scenario T10. 139

4.11 Aggregate non-dominated fronts from MOMF, MOSW and MOTR, and best-of-run individuals from BMMF, BMSW and BMTR on scenario XT12. 141
4.12 Normalised TWT and penalty values across all test and extreme test instances. 142
4.13 Violin plot of mean queue length over all dispatch points, mean queue length when local search is used and *positive penalty* is assigned, and mean queue length when local search is used and *no penalty* is assigned with BMMF on test instances. 144
4.14 Violin plot of mean queue length over all dispatch points, mean queue length when local search is used and *positive penalty* is assigned, and mean queue length when local search is used and *no penalty* is assigned with BMMF on extreme test instances. 145
4.15 Mean penalty per queued job for all queue lengths on scenario T10. 146
4.16 Mean penalty per queued job for all queue lengths on scenario XT12. 148
4.17 Evolved DR with best overall performance, evolved by MOSW. 149
4.18 Plots of Mean Penalty for each Tie frequency across test and extreme test instances. 155
4.19 Aggregate non-dominated fronts from TB-MO, and best-of-run individuals from TB-SO and Benchmark GP on scenario T1. 156
4.20 Aggregate non-dominated fronts from TB-MO, and best-of-run individuals from TB-SO and Benchmark GP on scenario T2. 156
4.21 Aggregate non-dominated fronts from TB-MO, and best-of-run individuals from TB-SO and Benchmark GP on scenario XT1. 157

4.22 Aggregate non-dominated fronts from TB-MO, and best-of-run individuals from TB-SO and Benchmark GP on scenario XT2. 157

5.1 Violin plots presenting results (TWT in 1000s) of test scenarios for Stage 1 GP and STGP methods. Note that the vertical axis scales are different in each subfigure. These figures shows the distribution of TWT values attained from the 50 independent GP runs of each method. 175

5.2 Violin plots presenting results (TWT in 1000s) of extreme test scenarios for Stage 1 GP and STGP methods. Note that the vertical axis scales are different in each subfigure. These figures shows the distribution of TWT values attained from the 50 independent GP runs of each method. 176

5.3 Violin plots presenting results (TWT in 1000s) of test scenarios for Stage 2 GP and STGP methods. Note that the vertical axis scales are different in each subfigure. These figures shows the distribution of TWT values attained from the 50 independent GP runs of each method. 182

5.4 Violin plots presenting results (TWT in 1000s) of extreme test scenarios for Stage 2 GP and STGP methods. Note that the vertical axis scales are different in each subfigure. These figures shows the distribution of TWT values attained from the 50 independent GP runs of each method. 183

5.5 Violin plots presenting results (TWT in 1000s) of test scenarios for Stage 3 GP and STGP methods. Note that the vertical axis scales are different in each subfigure. These figures shows the distribution of TWT values attained from the 50 independent GP runs of each method. 188

5.6 Violin plots presenting results (TWT in 1000s) of extreme test scenarios for Stage 3 GP and STGP methods. Note that the vertical axis scales are different in each subfigure. These figures shows the distribution of TWT values attained from the 50 independent GP runs of each method. 189

5.7 An evolved rule from *GP1* which outperformed ATC on 34/48 test instances and WCOVERT on 36/48 test instances. 201

5.8 An evolved rule from *ST1* which outperformed ATC on 33/48 test instances and WCOVERT on 39/48 test instances. 201

5.9 An evolved rule from *ST1A* which outperformed ATC on 25/48 test instances and WCOVERT on 32/48 test instances. 202

5.10 An evolved rule from *ST1B* which outperformed ATC on 34/48 test instances and WCOVERT on 38/48 test instances. 202

5.11 An evolved rule from *ST1S* which outperformed ATC on 26/48 test instances and WCOVERT on 34/48 test instances. 203

5.12 An evolved rule from *ST1F* which outperformed ATC on 25/48 test instances and WCOVERT on 37/48 test instances. 203

5.13 An evolved rule from *GP2*, and in simplified form below, which outperformed ATC on 44/48 test instances and WCOVERT on 47/48 test instances. 204

5.14 An evolved rule from *ST2* which outperformed ATC on 35/48 test instances and WCOVERT on 43/48 test instances. 205

5.15 An evolved rule from *ST2A* which outperformed ATC on 39/48 test instances and WCOVERT on 43/48 test instances. 206

5.16 An evolved rule from *ST2S* which outperformed ATC on 23/48 test instances and WCOVERT on 33/48 test instances. 207

5.17 An evolved rule from *ST2F* which outperformed ATC on 29/48 test instances and WCOVERT on 38/48 test instances. 207

5.18 An evolved rule from *GP3* which outperformed ATC on 22/48 test instances and WCOVERT on 34/48 test instances. 208

5.19 An evolved rule from *ST3* which outperformed ATC on 36/48 test instances and WCOVERT on 38/48 test instances. 208

5.20 An evolved rule from *ST3A* which outperformed ATC on 31/48 test instances and WCOVERT on 43/48 test instances. 210

5.21 An evolved rule from *ST3S* which outperformed ATC on 33/48 test instances and WCOVERT on 42/48 test instances. 211

5.22 An evolved rule from *ST3F* which outperformed ATC on 18/48 test instances and WCOVERT on 28/48 test instances. 212

5.23 Number of instances DRs in Figures 5.7 to 5.22 that outperform WCOVERT (left) and ATC (right) on 50 problem instances from each test and extreme test scenario. 214

5.24 Mean Z-score of DRs in Figures 5.7 to 5.22 across 50 problem instances from each test and extreme test scenario. . . . 215

5.25 Mean best-of-generation fitness (mean normalised TWT) across all methods through 50 generations of evolution. 218

6.1 Paired plot of Stage 1 results. This plot contains the Pareto fronts of 500 runs for all methods, 5188 DRs (2495 distinct DRs) from MOGP$_{1,N}$, 4640 DRs (1982 distinct DRs) from MOGP$_{1,S}$ and 500 DRs from SOGP$_1$. 234

6.2 Plot of normalised TWT over test instances vs extreme test instances. The bottom plot zooms in on the area of best test and extreme test instances. 235

6.3 Paired plot of Stage 2 results. This plot contains the Pareto fronts of 500 runs for all methods, 6299 DRs (2885 distinct DRs) from MOGP$_{2,N}$, 6585 DRs (2268 distinct DRs) from MOGP$_{2,S}$ and 500 DRs from SOGP$_2$. 240

6.4 Plot of normalised TWT over test instances vs extreme test instances. The bottom plot zooms in on the area of best test and extreme test instances. 241

6.5 Paired plot of Stage 3 results. This plot contains the Pareto fronts of 500 runs for all methods, 8535 DRs (3159 distinct DRs) from MOGP$_{3,N}$, 7376 DRs (2280 distinct DRs) from MOGP$_{3,S}$ and 500 DRs from SOGP$_3$. 243
6.6 Plots of normalised TWT over test instances vs extreme test instances for Stage 3. The bottom plot zooms in on the area of best test and extreme test instances. 244
6.7 Paired plot of Stage 4 results. This plot contains the Pareto fronts of 500 runs for all methods, 9610 DRs (3401 distinct DRs) from MOGP$_{4,N}$, 7894 DRs (2343 distinct DRs) from MOGP$_{4,S}$ and 500 DRs from SOGP$_4$. 248
6.8 Plot of normalised TWT over test instances vs extreme test instances for Stage 4. The bottom plot zooms in on the area of best test and extreme test instances. 249
6.9 Paired plots of results and plots of normalised TWT over test instances vs normalised TWT over extreme test instances using the NumNodes as the plotting symbol for MOGP with NSGA-II (a)–(d), MOGP with SPEA2 (e)–(h) and SOGP (i)–(l).250
6.10 DR with the lowest mean normalised TWT on test and on extreme test scenarios. 254
6.11 Two of the best DRs evolved by MOGP methods. 255
6.12 Rotated 3D plot of aggregate non-dominated Pareto fronts from each method. 256
6.13 Graph of mean$\pm$standard deviation of mean normalised TWT vs size of DR. The graph is cut off at 20 nodes for space, and methods are offset from each other for visibility. 258
6.14 Graph of mean$\pm$standard deviation of mean normalised TWT vs number of distinct terminal nodes. The methods are offset from each other for visibility. 260
6.15 Most frequently occurring attribute subsets of size two to nine. 263

List of Notation

N_j	Number of operations of job j
O_j	Set of operations of job j
w_j	Weight of job j
d_j	Due date of job j
r_j	Release date of job j into the shop
C_j	Completion time of job j
f_j	Flowtime of job j
T_j	Tardiness of job j
$\sigma_{j,i}$	ith operation of job j
$p(\sigma_{j,i})$	Processing time of operation $\sigma_{i,j}$
$m(\sigma_{j,i})$	Machine required to process $\sigma_{i,j}$

List of Abbreviations

ACO	Ant colony optimisation
ATC	Apparent tardiness cost (dispatching rule)
DR	Dispatching rule
DJSS	Dynamic job shop scheduling
EA	Evolutionary algorithm
EC	Evolutionary computation
EDD	Earliest due date (dispatching rule)
EMO	Evolutionary multiobjective optimisation
ES	Evolutionary strategy
EP	Evolutionary programming
FCFS	First-come-first-served (dispatching rule)
GA	Genetic algorithm
GP	Genetic programming
GPHH	Genetic programming based hyper-heuristic
HH	Hyper-heuristic
JSS	Job shop scheduling

MO Multiobjective

MOEA Multiobjective evolutionary algorithm

MOGP Multiobjective genetic programming

MOGPHH Multiobjective genetic programming based hyper-heuristic

MS Minimum slack (dispatching rule)

NSGA-II Non-dominated sorting genetic algorithm II

PSO Particle swarm optimisation

SI Swarm intelligence

SPEA2 Strength Pareto evolution algorithm 2

SPT Shortest processing time (dispatching rule)

S/RPT Slack per remaining processing time (dispatching rule)

STGP Strongly typed genetic programming

TWT Total weighted tardiness

WCOVERT Weighted cost over time (dispatching rule)

WSPT Weighted shortest processing time (dispatching rule)

Glossary of Terms

This glossary lists the definitions of common terms used through this thesis.

arrival rate The rate at which jobs arrive into a dynamic manufacturing system, often following a stochastic process.

balanced job shop A job shop where the average processing times of operations is the same on all machines.

crossover A genetic operator in which genes of two parent individuals are selected, manipulated and recombined to produce a new set of genes for the child individual.

dispatching rule A scheduling heuristic which assigns a priority value to each job waiting in a specific machine's queue, and selects the highest priority job to begin scheduling.

dynamic job shop scheduling A job shop scheduling problem where jobs arrive throughout time according to a stochastic process, and there is no knowledge about the job until its arrival into the shop.

elitism A genetic operator which selects only the fittest individuals of the parent population up to the prescribed proportion.

evaluation The process of measuring the fitness of individuals within a GP population during the evolutionary process.

evolutionary computation The field of study of population based optimisation methods which simulate evolution inspired by biology.

feature An attribute of the problem domain.

feature construction The process of combining existing features of a (machine learning) task to make new features.

feature manipulation The process of altering the input space of a (machine learning) task to improve the performance and (learning) quality.

feature selection A combinatorial optimisation problem which aims to find a subset of the original features which is as small as possible but still sufficiently describes the problem space.

fitness A measure of how well adapted an individual is to perform a specific task.

genetic operators Sources of variation in the evolutionary process: crossover, elitism and mutation.

genetic programming (GP) An evolutionary computation method.

grammar A context-free grammar is given by (S, N, Σ, P), where S is the start symbol, N is a set of non-terminals, Σ is the set of terminals, and P is the set of production rules of the grammar.

heuristic A "rule of thumb" which seeks a good solution at reasonable computational cost but does not guarantee optimality.

hyper-heuristic Search method for choosing or generating heuristics (or components of heuristics) to solve a range of optimisation problems.

initialisation The process of randomly generating the initial population of individuals in GP before the evolutionary process proceeds.

job A task to be scheduled, consisting of a sequence of a least one operation which must be processed on specific machines in the shop system.

job shop A manufacturing environment with a set of machines, and a set of jobs to be processed through the machines, where each job has a specified route through the machines.

job shop scheduling (JSS) The task of creating a schedule for processing jobs in a job shop manufacturing environment.

machine A resource which processes jobs in a manufacturing system.

makespan The maximum completion time (i.e. the completion time of the last job to finish processing).

meta-heuristic Problem independent strategies which guide the search process to find optimal, or near-optimal, solutions to optimisation problems.

mutation A genetic operator which takes an individual, chooses a mutation point at random, and the subtree routed at that point is replaced with a randomly generated subtree to create the child individual.

myopic Short-sighted.

non-domination A solution which is not dominated by any other solution across all objectives in a multiobjective optimisation problem.

objective function A measure of job delivery speed or customer satisfaction which is to be optimised.

operation A component part of a job, with a specified machine and processing time.

Pareto front The set of non-dominated solutions in a multiobjective optimisation problem.

queue The set of jobs waiting at a machine to be processed.

release time The time that a job enters the manufacturing system and is available to be scheduled.

representation The form in which a heuristic rule or genetic program is represented for evolution and computation purposes.

schedule An allocation of jobs to machines, including the start time of each operation within each job.

scheduling The process of constructing a schedule.

selection The process of determining which individuals are to be used for the genetic operators.

shop A manufacturing environment with a set of machines (resources) available to process a set of jobs (tasks).

static job shop scheduling All jobs are available and all their properties known at the start time of the manufacturing process.

tardiness How overdue a completed job is (tardiness is zero when the job is completed early).

unbalanced job shop A job shop where the average processing time of operations is not the same on all machines.

utilisation The proportion of time a machine is not idle.

weight The importance of a job, used to calculate its weighted tardiness.

Chapter 1

Introduction

1.1 Problem Statement

Job shop scheduling (JSS) is a common and difficult problem that has been described as a "fascinating challenge" [28] and has been widely studied in the academic literature over the past 60 years [48, 116]. A *job shop* is a manufacturing environment in which products (i.e. jobs) are produced by following a product-specific route through a finite set of machines. JSS problems are computationally challenging, falling in the class of *NP*-hard problems with only a few exceptions [12]. Further, there are many factories around the world (tens of thousands in the USA alone) that follow the job shop model, producing billions of dollars worth of products each year [62]. The development of more effective means of scheduling jobs is worthwhile as it has the potential to increase throughput, decrease costs, and increase profitability [62].

Job shop scheduling is a combinatorial optimisation problem consisting of a set of machines and a set of jobs which must be processed by the machines, subject to a set of constraints and with the aim of finding a schedule of jobs which optimises some measure of delivery speed or customer satisfaction [48].

The job shop scheduling problem has two main forms: the static job shop deals with the scheduling of an entirely known set of jobs; whereas in the dynamic job shop no job information is known until the job arrives into the shop system at an unknown time [118]. Due to the availability of knowledge in the static job shop problem, scheduling often uses traditional combinatorial approaches, such as branch-and-bound [72], total enumeration and Lagrangian relaxation [49]. However, in dynamic job shop scheduling problems there are no optimal solutions, as all the information is only known at the end of the scheduling period, therefore methods that construct complete schedules in advance are not able to be used. Heuristics (including dispatching rules) are commonly used scheduling methods in dynamic job shop environments [12, 116].

Dispatching rules (DRs), also known as priority rules or scheduling rules, are popular for job shop scheduling due to their ease of implementation, low time complexity (and hence low computational cost), and ability to cope with both static and dynamic environments [12]. Dispatching rules are mathematical functions of attributes of the job shop, machines and jobs currently queued at a machine awaiting dispatch. They assign a priority value to each job waiting in the machine queue, whenever a machine becomes available to process a job. The job that is assigned the highest priority value is selected to begin processing. This means scheduling decisions are made as they are to be implemented, which is why dispatching rules cope with dynamic environments so well. As dispatching rules are heuristics, they cannot be guaranteed to give optimal solutions [131].

The manual development of dispatching rules is a time consuming process that requires domain knowledge. Manually designed dispatching rules are frequently specific to a particular job shop scheduling instance, defined by the number of machines, objective function, utilisation, etc. An approach to overcoming this problem that has gained popularity over the past twenty years [62] is the use of evolutionary computation based techniques to *automatically* discover new dispatching rules. These approaches are often known as hyper-heuristics [29, 21], as they search the search space of potential heuristics (i.e. dispatching rules) rather than searching the space of solutions (i.e. schedules of jobs) directly. Genetic Programming (GP) [70] is an evolutionary computation technique which has been shown to be an ideal candidate for use as a hyper-heuristic for shop scheduling problems [21]. The main reason for this is the common representation of GP, tree-based GP [70], naturally lends itself to represent mathematical functions, and hence to represent dispatching rules. GP is also relatively easy to implement and has variable length encoding.

The overall goal of this thesis is to develop genetic programming based hyper-heuristics for the efficient evolution (automatic generation) of robust, reusable and effective scheduling heuristics for job shop scheduling environments, with greater interpretability.

1.2 Motivations

The purpose of this section is to provide the motivations for this thesis. Firstly, we introduce some of the challenges of scheduling. Secondly, we describe why GP is an ideal evolutionary computation method to use to develop scheduling heuristics. Finally, we discuss limitations of the existing research.

1.2.1 Challenges of Scheduling

Job shops are a common real-world scheduling environment [62] and the complexity of the scheduling task combined with real-world prevalence has led to scheduling in such environments being widely studied for many years [62]. Scheduling is a very challenging problem. Often the scale of the problem is very large, with many hundreds of tasks (jobs) needing to be scheduled across many tens or hundreds of resources (machines). Even for the smallest of scheduling problems, finding the best possible solution can quickly become near impossible. For a 'simple' static single machine problem with 10 jobs, there are $10! = 3,628,800$ possible sequences of jobs.

As the number of jobs and machines increases, the number of possible sequences increases and the scheduling task becomes increasingly difficult. The majority of scheduling problems are *NP*-hard [12].

This computational challenge, and the fact that *many* real-world manufacturing and service environments use/need scheduling, means that there is real benefit to be found from improving scheduling. Increasing throughput of production and decreasing costs from, e.g., idle machines, inventory costs from holding partially completed items, and late delivery costs, all lead to increasing profitability.

Job shop scheduling in real-world situations is certainly a multiobjective problem; objective functions that relate to both the customer and the business sides of manufacturing, as well as business values such as environmental impact through power use or emissions, may all be of interest to the scheduler. Sometimes these objectives may work together, e.g., minimising flowtime (the time jobs are in the shop system) is likely to lead to lower inventory costs. However, frequently the objectives of interest are in conflict, e.g., minimising the number of tardy jobs and minimising the flowtime.

When the multiobjective nature of the scheduling task itself is combined with the task of finding better means of scheduling, the task becomes even more difficult. An effective and efficient scheduling rule must not only be good in terms of the multiple objectives of the scheduling problem but also in terms of the multiple attributes of a good scheduler. Further, the majority of manually designed dispatching rules and algorithms are designed for specific job shop scheduling instances, and do not generalise well across different shop settings. Scheduling is a major challenge due to the dynamic nature of the task. In dynamic job shops not only are jobs arriving at unknown times, but the machines may break down unexpectedly as well.

1.2.2 Why Genetic Programming?

Genetic programming is an ideal evolutionary computation method to automatically generate dispatching rules because of the following reasons.

Representation. Tree-based genetic programming [70] is the most common form of genetic programming; programs are represented as trees, made up of function and terminal nodes. Dispatching rules are mathematical functions used to determine the priority of jobs awaiting processing, and choose which job must begin next. Such mathematical functions are naturally represented as trees with mathematical functions as the function nodes, and constants and variables of the job shop problem domain as the terminal nodes in GP.

Variable Length Encoding. One advantage of genetic programming over other evolutionary computation methods is that the size of the program trees evolved is able to vary [7], between a minimum depth, m_1, and maximum depth, m_2. This means genetic programming is covering a larger search space of potential functions than a fixed length method such as a genetic algorithm. This is suitable for JSS where variable lengths of rules are needed, particularly for dynamic JSS environments which are complex and therefore require more complex dispatching rules to schedule effectively.

Previous Research. There is a large body of research using genetic programming for the automated design of scheduling rules [50, 56, 57, 83, 95, 96, 98, 107], giving us confidence in the approach and setting successful precedents. These approaches are classified as hyper-heuristic approaches [29, 21], as GP is searching for a scheduling heuristic (the dispatching rule) rather than searching for a schedule of jobs directly. Dispatching rules (DRs) are able to be reused in new JSS problems, with the same and different parameters, unlike the development of a schedule which is specific to a given instance. This is an advantage of the hyper-heuristic approach. GP has been used to develop DRs in a range of scheduling environments, from the simpler static two-machine flow shop [42], through job shops with many machines [50], dynamic flexible JSS [126], to very complex manufacturing environments such as semi-conductor manufacturing [108]. These works consider differing simplifications of the JSS model; some focus on static problems [83] and others on dynamic problems [50]. Works focus on different objectives of delivery speed, including those based on makespan [58, 57], flowtime [50], tardiness [58], and combinations of these [126]. Further works also consider evolving due date assignment rules alongside DRs [94] and DRs for order acceptance and scheduling [106]. Rules evolved by GP are frequently reported to compete with rules from the literature [94, 57]. However, there are limitations to this body of existing research which are discussed below.

Ease of Implementation. It is *relatively* easy to implement a genetic programming based system to evolve dispatching rules, using one of the many existing GP software packages, such as ECJ [78]. These packages require extension to include a job shop discrete-event simulation model to evaluate how good a dispatching rule is. Suitable functions need to be identified as DRs are mathematical functions and there are a large number of such functions which can be explored. It is less easy to identify which variables relevant to the problem domain of job shop scheduling should be included. The attributes of machines (e.g. number of jobs in the queue or how long the machine has been idle), jobs (e.g. the processing time of the current operation or the due date) and of the shop as a whole (e.g. the current clock time of the simulation) are naturally represented as terminals.

1.2.3 Limitations of Existing Work

Genetic programming has been used for the automated design of dispatching rules and other forms of scheduling policy. However, there are limitations to the research that has been conducted.

Dispatching rules evolved by most approaches lack a global perspective as the features of the jobs, machines and the shop used are mainly limited to the current state of the current machine and its queue. Research has explored the use of *look-ahead* functions to try and improve the scheduling performance, as they have the ability to significantly improve schedules when used with a reasonable DR [43]. Most DRs construct *non-delay* schedules where a machine cannot be idle while there is an operation awaiting processing [110]. Some works [98, 50] incorporate a "look-ahead" in which a machine is notified that a job will be arriving once it is finished its current operation. When the jobs in the queue at the machine are evaluated, if the job which has not yet arrived has the highest priority then the machine is able to enforce idle time, and wait for the job to arrive. However, there are other ways in which dispatching rules can be encouraged to take into account the wider state of the shop. Ways in which the longer term consequences of a particular scheduling decision are yet to be investigated.

Another limitation of DRs evolved automatically is their interpretability. The interpretability of heuristics evolved by genetic programming, and black box optimisers, has been identified as a *crucial* aspect to gain the trust of operators or managers [16]. How well, or even whether, these people are able to understand and interpret *how* and *why* the DR works is therefore important if DRs evolved using evolutionary computation (EC) techniques are to be used in real-world situations. One particular limitation is that many of the comparisons made in evolved DRs cannot be readily interpreted in terms of units, i.e., semantically most DRs evolved using GP are incorrect [58] because standard tree-based GP allows the unrestricted composition of available functions and features, subject only to the tree-depth restriction. Knowing how well a given rule will generalise across different distributions of arrivals and processing times, and different scales of shops (number of machines and jobs), is also necessary if they are to be used in practice. Means of improving the interpretability previously employed include online rule simplification techniques [61, 66, 137], analysis and visualisation of priority indices [15], and identifying weaknesses in decision logic of a given dispatching rule [17]. The interpretability of DRs evolved by GP is a particularly interesting problem as it has been reported that there is evidence that complex scheduling environments require heuristics with a certain level of complexity, so often an easy-to-interpret representation can result in lower quality heuristics [15].

Job shop scheduling is inherently a multiobjective problem. Often the multiple objectives explored in research are different manufacturing objectives such as tardiness and flowtime, or utilisation levels of the machines in the shop. While the job shop scheduling task is recognised as a multiobjective problem, we believe that the effectiveness of a dispatching rule should also be multiobjective. A limitation of existing research into multiobjective genetic programming for the automated design of scheduling rules is that very frequently researchers have been interested *only* in the performance of the evolved rules in terms of a measures of manufacturing speed or efficiency. There are other factors which are important if rules designed without human expert domain knowledge are to be trusted by human operators involved in the manufacturing process, particularly, how are the evolved rules to be understood? There are many more potential objectives to be explored relating directly to the scheduling problem as well as to the evolved dispatching rule that can enable the discovery of dispatching rules which better account for the issues of myopia ("shortsightedness") and understandability.

Job shop scheduling problems have a large number of attributes which could potentially be used in the terminal set of a genetic programming approach, and hence could appear in a useful dispatching rule. However, not all of these attributes are important, and some may be redundant in the presence of other attributes. The disadvantage of using a large number of attributes is that the more attributes, the larger the search space that GP has to cover. These attributes are often known as 'features' within the machine learning and evolutionary computation communities. In this thesis we will use the terms 'feature' and 'attribute' interchangeably. Feature manipulation [91] is the process of altering the input space of a (machine learning) task to improve the performance and (learning) quality. In scheduling, the input space consists of the attributes that are chosen from the scheduling domain to be part of the terminal set, together with the function set. Feature selection is one form of feature manipulation, which aims to find a subset of the original features which is as small as possible but still sufficiently describes the problem space [67]. Feature construction [77, 92] is another form of feature manipulation; existing features are combined to make new features. Feature manipulation in genetic programming for scheduling has not previously been explored. GP performs feature selection *automatically* as the most effective GP trees survive to future generations, but seldom are the features appearing across best-of-run evolved dispatching rules examined and the information gleaned used. GP can also be considered to perform feature construction automatically, as there are 'fragments' or sub-branches of the GP trees which may appear frequently in multiple trees and these may be used as constructed features. Overall, the use of feature manipulation aims to reduce the size of the search space, improve the learning performance, and potentially make the output easier to interpret.

1.3 Research Goals

The overall goal of this thesis is to develop genetic programming based hyper-heuristics for the efficient evolution (automatic generation) of robust, reusable and effective scheduling heuristics for job shop scheduling environments, with greater interpretability. The focus of this research is to investigate how the use of genetic programming to discover new dispatching rules for job shop scheduling problems can be improved through studying the representations used, the incorporation of additional feedback on the performance of dispatching rules through other heuristic methods and the inclusion of additional objectives. We aim to use genetic programming to evolve new dispatching rules that are effective, incorporate attributes of the wider shop system and are interpretable to human operators. The evolved rules should be reusable in unseen job shop scheduling scenarios and competitive with existing dispatching rules from the literature. There is always some gap between the robustness and reusability of scheduling rules. Evolved rules not only need to be robust and reusable in terms of effective performance across different machine configurations, but we also should be able to understand the rules enough to trust the use of them across different configurations by increasing our understanding and interpretation of the way they work.

This thesis will address the following research objectives:

1. *Investigate whether GP can discover dispatching rules which are competitive with known optimal approaches on static job shop scheduling problems.*

   The static two-machine job shop scheduling environment is perhaps the simplest of all job shop environments. The makespan, C_{max}, is the maximum completion time of all jobs $(1, \ldots, N)$ in the shop, i.e., $C_{max} = \max\{C_1, \ldots, C_N\}$, where C_j is the completion time of job j. When the objective of interest is to minimise the makespan, Jackson's algorithm [55] gives the minimum makespan. It is not clear (initially) whether GP can evolve rules that perform as well as Jackson's algorithm, i.e., *always* dispatching jobs in a way that gives the optimal makespan. We aim to find a good representation within GP to evolve such rules. Success in this task can be considered as certification of the representation being used and of the search capability of GP. This will give confidence in the ability of the system to evolve heuristics for the more complicated job shop environments for which we do not have an optimal solution to compare against. If GP is not able to evolve a heuristic which can match the performance of Jackson's algorithm then the representation being used should be modified.

Specific sub-objectives to investigate are:

(a) Investigate *representation* of Jackson's algorithm as a dispatching rule and how a dispatching rule can be shown to be equivalent to Jackson's algorithm.

(b) Investigate whether GP can *discover* a dispatching rule which is equivalent to Jackson's algorithm.

(c) Investigate evolving two dispatching rules simultaneously, one for each machine in the dynamic two-machine job shop, and compare this to the standard approach of using one dispatching rule to make scheduling decisions at all machines in the shop.
While focusing on the simpler job shop environments, as part of the baseline of the thesis, we will also investigate the dynamic two-machine job shop. The typical approach to evolutionary scheduling takes the single best DR evolved by the system and applies this DR to determine which job should be processed next at *every* machine in the shop (the job shop model is simplified to use one DR). However there are other ways that a fixed amount of computational time could be used to develop rules for a system.

2. *Investigate the use of GP to develop "less myopic" scheduling rules.*

The majority of current research lies at two extremes: a global approach through optimisation methods where an entire schedule is constructed or the other extreme of very local approaches using dispatching rules. Dispatching rules are generally myopic, i.e., short sighted, and take into account the local and current conditions of the shop only [2, 42]. Dispatching rules generally examine the properties of the machine which is now available, including the queue of jobs awaiting processing at that machine at that point in time. The dispatching rule is used to make scheduling decisions at each machine in the job shop independently, so there is no consideration of the movement of jobs from one machine to another, or the impact a decision may have on the subsequent machines, whether positive or negative [108].

We will investigate two approaches to encourage dispatching rules to take into account the wider state of the shop and the consequences of scheduling decisions over an extended decision horizon. These aim to address the following sub-objectives:

(a) *Determine which possible attributes of jobs, machines and/or the shop system as a whole are useful to improve DR performance and how "less-myopic" features improve the scheduling performance.*
The attributes used in the terminal set have a large impact on the performance of the evolved scheduling rules, determining what information is available to the GP system, and how local/global the approach is. We will incorporate features from the wider shop system, to see if the rules evolved are less-myopic and give improved performance.

(b) *Investigate whether the inclusion of local search as an additional means of providing feedback on the fitness of a given dispatching rule throughout evolution can improve the performance and generalisation ability of evolved DRs.*
We want to find dispatching rules that make good decisions. The incorporation of local search allows us to examine the wider local effect of dispatching decisions over an extended decision horizon, and investigate whether this can encourage the evolution of dispatching rules that schedule jobs in an order which is better for more than just the first job dispatched, leading to

better global performance and better generalisation to unseen problem instances.

3. *Investigate whether a strongly typed GP (STGP) based hyper-heuristic approach can be used to encourage the evolution of interpretable dispatching rules.*

   The effectiveness and suitability of a dispatching rule evolved by genetic programming is normally measured only in terms of the scheduling objectives such as tardiness or flowtime, but for dispatching rules to be used in real-world scheduling environments they must be able to be trusted by human managers and operators involved in the scheduling process. This aspect of how well the rules are able to be understood, interpreted and trusted is often neglected in the existing literature. Further, rules evolved with only the standard GP constraints of closure are often semantically incorrect.

   One approach to creating dispatching rules with greater interpretability and semantic validity is to limit the allowable interaction between functions and terminals using STGP. STGP works by directly specifying which children each non-terminal can have; which is done indirectly in STGP by specifying the data types of each argument of each function as well as the data types returned by each terminal and function [85].

   The following sub-objectives will be investigated:

   (a) Develop terminal types and grammar(s) to specify the allowable interactions between these types which can be enforced with the use of STGP.

   (b) Investigate how the level of interpretability and performance compares between rules evolved using STGP, those evolved by the traditional GP based generation of dispatching rules, and manually designed rules from the literature.

(c) Evaluate whether the use of STGP can be justified (if performance is worse) as an acceptable compromise for improvement in interpretability.

(d) Investigate what insight can be gained into how a measure might be developed to evaluate and compare interpretability of automatically evolved rules.

4. *Investigate how multiobjective GP can be used to discover dispatching rules for dynamic JSS which are effective in terms of scheduling objectives and are also more easily understood by human operators.*

   The aim of this objective is not the development of a new GP algorithm, but analysis of how we can better use GP for automatic generation of DRs. Scheduling is inherently a multiobjective problem, for example, there is a trade-off between the size of the planning horizon and the computational cost of creating the schedule, or between training time and robustness. This is a well identified issue.

   We believe that DR performance is also multiobjective; performance in terms of the scheduling objectives is important but so is the interpretability of DRs and the trust that practitioners are able to place in DRs. In particular we investigate the trade-off between the size of the GP trees (compactness of heuristics/solutions) and understandability/interpretability of heuristics/solutions, and specifically we will:

   (a) Investigate whether interpretability can be improved by the inclusion of objectives relating to the size of the DR.

   (b) Determine which of the two MO algorithms, NSGA-II [32] and SPEA2 [146] performs better for GP to evolve interpretable rules.

5. *Investigate using the knowledge implicitly discovered through GP to perform feature manipulation to improve dispatching rule performance.*

Attribute/feature selection is an important part of developing an effective GP based approach for the automatic discovery of dispatching rules. While it has been acknowledged that a scheduling rule itself is often not as important as the interrelationships of the attributes in the rule, and the relevance of these attributes to the problem structure [42], the study of the attributes used, and in what combinations, is generally neglected in the scheduling community. The knowledge automatically discovered by GP through the terminals selected can be viewed as feature selection and can be used to alter the terminal set. Feature manipulation using GP has not been investigated in the JSS literature.

Analysis of the best-of-run evolved dispatching rules, searching for commonly occurring fragments of terminals and functions, can direct us to perform feature construction. Common fragments will be used as "higher-level constructed features". Using this additional information from the results of GP runs will enable us to reduce the use of terminals which are redundant or less useful and reduce the size of the search space. Including additional higher-level features may improve the learning performance, and as each higher-level feature will be interpreted before being included in the terminal set, this has the potential to improve the interpretability of the output.

Specific sub-objectives that will be investigated are:

(a) Investigate whether the restriction of the search space through STGP reveals good components or combinations of components that we can use to further restrict or alter the terminals included in the terminal set.

(b) Investigate whether feature manipulation guided by GP can improve interpretability.

Table 1.1: Mapping of contributions (Contrib) to research objectives (RO).

	Contrib 1	Contrib 2	Contrib 3	Contrib 4
RO1	×			
RO2		×		
RO3			×	×
RO4				×
RO5				×

1.4 Major Contributions

This thesis makes the following major contributions. The mapping of contributions to research objectives is shown in Table 1.1.

1. This thesis presents an investigation into evolving dispatching rules for the static two-machine job shop with makespan objective function. This is the first time that GP has been used to evolve dispatching rules that are optimal for this problem. This was proved by showing that the evolved rule scheduled queued jobs in such a way that the makespan is always equal to the makespan attained by using the optimal scheduling algorithm, Jackson's algorithm. This work validates both the genetic programming based approach for generating dispatching rules for the JSS problem, and the representation used.

   Part of this contribution has been published in:

   Hunt, R., Johnston, M. and Zhang, M. "Evolving machine-specific dispatching rules for a two-machine job shop using genetic programming". In *Proceedings of the IEEE Congress on Evolutionary Computation* (2014), pp. 618–625.

2. This thesis presents an approach to developing "less-myopic" dispatching rules through the inclusion of additional terminals in the

feature set and the use of local search to provide additional feedback on the fitness of dispatching rules during evolution. The results show that inclusion of less-myopic terminals improves the mean performance, and decreases the standard deviation of performance of the best evolved rules, and that the inclusion of local search in evaluation leads to the evolution of dispatching rules which make better decisions over the local time horizon, and attain lower total weighted tardiness.

Parts of this contribution have been published in:

Hunt, R., Johnston, M. and Zhang, M. "Evolving "less-myopic" scheduling rules for dynamic job shop scheduling using genetic programming". In *Proceedings of the Genetic and Evolutionary Computation Conference* (2014), pp. 927–934.

Hunt, R., Johnston, M. and Zhang, M. "Using local search to evaluate dispatching rules in dynamic job shop scheduling". In *Proceedings of Evolutionary Computation in Combinatorial Optimization* (2015), pp. 222–233.

3. This thesis proposes the first use of strongly typed GP for the automatic discovery of dispatching rules with better interpretability. This work explores the ease of interpretation of a dispatching rule evolved by GP in depth and considers the ease of interpretation as an important trait of an effective dispatching rule. The results show that the performance of the dispatching rules evolved in the semantically constrained search space is not as good (on average) as the performance on the unconstrained search space. However, there were still effective rules evolved under STGP, and the interpretability of evolved rules is substantially improved. The results also show that there is a real trade-off between total weighted tardiness and inter-

pretability. Introducing conditional operators with predetermined conditions makes the rules easier to interpret, due to the condition tested being known, however performance is not as good as with the traditional conditional operator. The interpretability of dispatching rules is an essential aspect to be explored if rules that are automatically discovered by evolutionary computation techniques are ever to be employed in real world job shop scheduling environments, as managers and human operators must have a level of trust in and understanding of how the rule is working.

Part of this contribution has been submitted to Evolutionary Computation Journal.

Hunt, R., Johnston, M. and Zhang, M. "Evolving dispatching rules with greater understandability for dynamic job shop scheduling" (2015), 36pp. (under second review).

4. This thesis presents a multiobjective genetic programming approach with objectives of total weighted tardiness, dispatching rule size, and the number of distinct terminals appearing in the genetic programming tree. This approach aims to evolve dispatching rules with greater interpretability while also revealing more information about important features of the search space and scheduling environment. The results show that the inclusion of higher-level constructed terminals and specialised conditionals improves the scheduling performance. Including the dispatching rule size as an objective leads to dispatching rules with similar performance but smaller size and better interpretability. Including the number of distinct terminal nodes as an objective highlights redundancies in the terminal set, revealing that some features are more useful as part of a larger constructed feature. This is the first work that considers multiple objectives of dispatching rules in terms of the trade-off between scheduling behaviour and interpretability in an automatically generated dispatch-

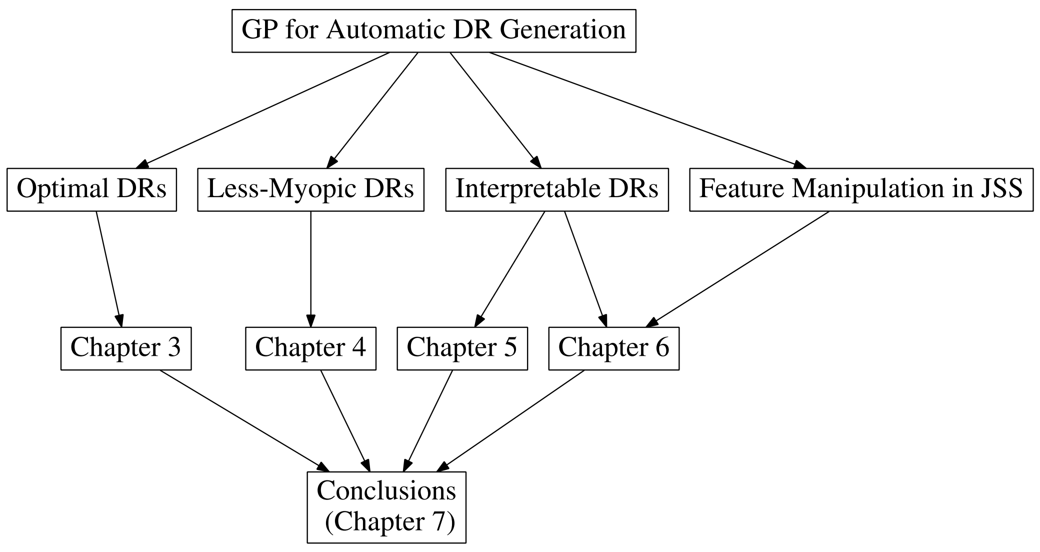


Figure 1.1: Overall structure of the thesis contribution chapters.

ing rule. Not only must dispatching rules evolved by genetic programming be competitive with those from the literature in terms of, e.g., the total weighted tardiness obtained, but they must also be able to be trusted. People are more likely to choose something they understand (and can explain) over something that works better.

Part of this contribution has been submitted to IEEE Transactions on Cybernetics.

Hunt, R., Johnston, M. and Zhang, M. "Multiobjective genetic programming for feature selection and dispatching rule generation in job shop scheduling" (2015), 14pp. (under review).

1.5 Organisation of Thesis

The remainder of this thesis is organised as follows. Chapter 2 presents a literature review of related works. Chapters 3–6 present the main contributions of the thesis, which can be seen in Figure 1.1. Chapter 7 concludes the thesis.

Chapter 2 presents descriptions of the job shop scheduling problem and genetic programming methodology used for this research. The basic concepts of genetic programming, heuristics, hyper-heuristics for automatic generation of heuristics and grammar guided genetic programming are presented. This chapter gives a review of current research using genetic programming for the automatic generation of new dispatching rules.

Chapter 3 proposes a GP system to evolve dispatching rules for the static two-machine job shop scheduling problem with the makespan objective function. Jackson's algorithm, which is known to give optimal solutions, is described, followed by how it can be represented as a dispatching rule. Modifications to the feature set are investigated and evolved dispatching rules are analysed. Chapter 3 also compares two representations for scheduling in the dynamic two-machine job shop: one dispatching rule used at both machines or two rules evolved simultaneously, one for each machine.

Chapter 4 investigates how GP can be used to discover dispatching rules which are "less-myopic". An extended terminal set is proposed, including more information about the state of the wider shop system at the point that scheduling decisions are made. Further, a new fitness evaluation scheme is proposed, where local search is used to evaluate dispatching rule performance over an extended decision horizon.

Chapter 5 proposes the use of strongly typed genetic programming to enforce semantic constraints and enable the automatic discovery of semantically correct dispatching rules with greater interpretability. Incremental grammars exploring additional types, terminals and functions are investigated.

Chapter 6 builds on the work of Chapter 5 proposing the use of additional objectives to encourage greater interpretability of evolved dispatching rules. This approach is combined with feature construction and selection, based on analysis of results, to discover useful components and relationships between features, improving the effectiveness of the search.

Chapter 7 summarises the main findings of the thesis and draws overall conclusions. The thesis concludes with further research opportunities based on the results attained in this thesis.

Chapter 2

Literature Review

This chapter begins by providing an introduction to the basic concepts which are essential to this thesis: scheduling, heuristics and hyper-heuristics, and evolutionary computation. Then job shop scheduling is described in greater depth, introducing job properties, notation, and performance measures used widely in the literature. Next, a review of scheduling methods in static and dynamic job shop environments, from optimal algorithms to heuristic rules is presented. This chapter then presents genetic programming, focusing on the key elements of representation, evolutionary operators and the basic algorithm. An introduction to the local search heuristic and feature manipulation is given. The final section of this chapter provides a summary of the literature on using hyper-heuristics to generate heuristics, in particular genetic programming based generation of heuristics for scheduling.

2.1 Introduction

The purpose of this section is to provide an introduction to the basic concepts of scheduling problems, heuristics and hyper-heuristics and evolutionary computation.

2.1.1 Scheduling

Whenever there is a choice in the order in which a number of tasks can be performed, we have a scheduling (sequencing) problem [28]. These problems occur commonly, daily, and everywhere. Scheduling is a decision-making process [110] which aims to assign tasks to scarce resources over time, subject to constraints, optimising some measures of customer satisfaction or delivery speed [48, 75]. Such problems are combinatorial optimisation problems, with very large search spaces for even a relatively small number of tasks.

Scheduling, and closely related problems, are found not only in manufacturing, but also in transportation and distribution, communication and service provision [28], as well in information processing environments [110].

Scheduling combines both allocation of tasks to resources, sequencing of tasks, and determining when the tasks are to be performed [6]. Problems are categorized by the nature of the tasks and the configuration of the resources available to process them [6]. The resources can vary in number and in layout, occurring in sequence or in parallel. Task availability splits problems into static and dynamic problems: in static systems the set of available jobs does not change with time whereas in dynamic systems tasks appear over time and there is no knowledge of them prior to their arrival [6]. Systems are also classified as either deterministic or stochastic dependent on whether conditions are known with certainty, or follow probability distributions [110]. In stochastic scheduling systems, decisions also have to be made with limited information available [110].

The majority of scheduling problems are *NP*-hard [75], which are among the most difficult problems to solve [62]. The commonness of scheduling problems has lead to an extensive body of research, beginning in the early 1950s [75, 116]. Research began with the development of algorithms, arising from computational analysis, and progressed to enumerative search techniques such as branch-and-bound [72], mathematical programming approaches including Lagrangian relaxation [49], heuristic approaches such as local search [1] and meta-heuristic approaches.

2.1.2 Heuristics, Meta-heuristics and Hyper-heuristics

Heuristics. Heuristics are "rules of thumb" which seek good solutions at reasonable computational cost but do not guarantee optimality [131]. The need for heuristics arises from real-world problems which are difficult to solve. Michalewicz and Fogel [81] give five reasons why real-world problems are so difficult to solve: large search spaces, complicated problems which can only be modelled in severely simplified forms, noise of time-dependent evaluation functions, heavily constrained search spaces, and inadequately prepared human problem solvers. Solving these computational problems using exact methods is often impractical due to these challenges, and due also to the computational resources or time required.

Meta-heuristics. Meta-heuristics are problem independent (provided the right encoding) strategies which "guide" the search process in order to find optimal, or near-optimal solutions [13] to hard optimisation problems. Meta-heuristics use a variety of techniques, including random moves and penalties, to escape local optima. Meta-heuristic methods include simulated annealing [68], Tabu search [45], and artificial intelligence techniques.

Hyper-heuristics. Hyper-heuristics have been defined as heuristics which choose or generate heuristics [29, 21]. More formally, hyper-heuristics are search methods for choosing or generating heuristics (or components of heuristics) to solve a range of optimisation problems [21]. Chakhlevitch and Cowling [23] define a hyper-heuristic as being a higher level heuristic which manages a set of lower level heuristics, searches for a good method to solve the problem rather than a direct solution, and uses limited problem-specific information.

The manual development of heuristics is time consuming, requires domain knowledge, and generally leads to a heuristic for a specific problem instance. Although heuristic methods have been successfully applied to real-world problems, they often do not generalise well to new or similar problems to what they were created for [20]. Hyper-heuristics have arisen from the desire to counter these issues of manual development and automate the process of designing and adapting heuristics for complicated computational search problems [18] with greater generality [21]. The first uses of what would now be classified as hyper-heuristic approaches were in the early 1960s [18]; interestingly these approaches were applied to job shop scheduling. Hyper-heuristic approaches can be separated into two categories:

Heuristic selection aims to develop hyper-heuristic frameworks to adaptively select suitable existing heuristics based on the current state of the problem which is being solved and the history of the problem solving process [20].

Heuristic generation builds new heuristics by combining various small components such as common statistics or operators used in existing heuristics. Heuristic generation makes it feasible to design heuristics for individual problem instances, which could even outperform manually designed heuristics, due to the process being less time and resource intensive [20].

Many meta-heuristics have been used as hyper-heuristic frameworks, including ant colony optimisation [19], genetic algorithms [29] and genetic programming [21]. The study of meta-heuristics, machine learning and operations research converge with the study of hyper-heuristics, combining new methods with existing problem domain knowledge [20]. There is great potential for these methods with greater generalisation to find improved solutions to difficult real-world combinatorial optimisation problems [20].

2.1.3 Evolutionary Computation

Evolutionary computation (EC) is the field of study within computational intelligence which takes its inspiration from biological evolution. EC methods simulate evolution, most often on a computer [3]. Methods are population-based, and feature four processes which form evolution in this population: reproduction, random variation, competition and selection [3]. The evolutionary process is an optimisation process. A *population* of hundreds or thousands of individuals is evolving, rather than a single individual [70]. There are two main areas of evolutionary computation: evolutionary algorithms and swarm intelligence.

Evolutionary Algorithms

Evolutionary algorithms (EAs) are a subset of evolutionary computation based on Darwinian evolutionary theory. EA methods take a population of individuals which is evolved through selection, mutation, reproduction and crossover [3]. Individuals which have greater fitness are more likely to survive and reproduce, hence contributing to the future population [3]. Four popular EAs, genetic algorithms, genetic programming, evolutionary strategies and evolutionary programming are briefly introduced here, in the form they originated in.

Genetic Algorithms (GAs) [51]. GAs were developed in the 1970s, and were the first form of EA to be widely accepted [136]. In GAs each individual, often known as a chromosome, is typically represented as a fixed length array. The array can be integers, real numbers, or bits. Selection in GAs, is based on genotype. The genotype is the "underlying genetic coding" [37]; in a GA this genetic coding is found in the chromosomes. In most forms of GAs recombination is more important than mutation [136].

Genetic Programming (GP) [70]. GP extends genetic algorithms through representing individuals as variable length executable computer programs. The most common program structure is a tree-based representation. The reproduction operators used in GP are elitism, crossover and mutation. GP is a "search-based automatic programming" technique [114]. GP is explained in detail in Section 2.3.

Evolutionary Strategies (ES) [8]. ES individuals have an object parameter set and objective function value which represents its fitness, and also a set of evolvable strategy parameters. The strategy parameters are specific to ESs, are subject to mutation (this is known as self-adaptation), and control the mutation operator which is applied to the individual's object parameters. Often ESs use only mutation (i.e. there is no recombination). Selection in ES is a deterministic process which ensures that only the best individuals (based on fitness values) are selected for the next generation.

Evolutionary Programming (EP) [39]. EP evolves finite state machines (F-SMs). FSMs take a set of input symbols and predict output symbols for the input symbols. Individual fitness is based on how well it is able to predict the next symbol in a known sequence [38], given all symbols observed thus far. Selection in EP is based on phenotypic behaviour ("the manner of response contained in the behaviour, physiology, and morphology of the organism" [37]) rather than on the genotype [38].

De Jong's "unified" model of EC [30] argues that these are all based on a "Darwinian notion of an evolutionary system", each having a population of individuals, measure of fitness, birth and death cycle based on fitness, and the idea of inheritance. Under this model, the various forms of EC are differentiated by their representations, and choices based on the representation to suit the problem being addressed.

Swarm Intelligence

Swarm intelligence (SI) is an area of evolutionary computation which uses the social behaviour of swarming or flocking creatures. There are five basic principles of swarm intelligence: proximity, quality, diverse response, stability and adaptability. A swarm is a "population of interacting elements that is able to optimise some global objective through collaborative search of a space" [65]. The two most popular forms of SI are particle swarm optimisation and ant colony optimisation.

Particle Swarm Optimisation (PSO) [64]. PSO is a simple algorithm based on simulating the social behaviour of a flock of birds or a school of fish. Each particle (solution) in the swarm knows its position in n dimensional solution space, and the fitness of the position. Particles remember the best position they have found, and know the best position that has been found globally by the swarm. The positions of particles are updated by moving towards the best positions. This aims to direct the swarm towards the best solution in the solution space.

Ant Colony Optimisation (ACO) [34]. ACO is inspired by the foraging behaviour of ants, who seek the shortest path between their colonies and a food source. ACO uses a population of ants (solutions) who store information on searched paths for food as pheromone trails. The pheromone on better solutions becomes stronger, and more attractive to ants. The best solution is thus the path with the most pheromone.

2.2 Job Shop Scheduling

Flow shop scheduling requires the construction of a schedule to process N jobs through a series of M machines. Each job is processed by every machine and all jobs follow the same route through the machines [110]. Job shop scheduling is the task of creating a production schedule for a set of N jobs to be processed through the M machines in the shop, so as to optimise a measure of delivery speed or customer satisfaction. In a job shop the number of machines a job visits can vary, and each job has its own predetermined route [110]. Shop scheduling environments are able to be succinctly described using $\alpha\,|\beta\,|\gamma$ notation [110]. The machine environment is described in the α field: Jm denotes a job shop environment with m machines and Fm denotes a flow shop with m machines. The β field provides characteristics of the processing and any constraints. This field can be left blank, or even contain multiple entries. If jobs have a release date which they cannot begin processing before, this field contains the entry r_j representing "release dates". The final field, γ, denotes the objective function which is to be *optimised* [110]. In a job shop, each job is made up of a sequence of operations, where each operation must be processed on a particular machine for a specified processing time, in a predefined order [12].

Job shop scheduling problems can be categorised as either static or dynamic [42], depending on the availability of information about imminent job arrivals. In static job shop scheduling, all job properties are known at the beginning of the scheduling period. In dynamic job shop scheduling, jobs arrive at the shop according to a stochastic process. Prior to a job's arrival in the shop there is no knowledge of the job. Dynamic scheduling is of more commercial interest than static scheduling. It is rare to know all details of a scheduling problem at the start, including when the scheduling task will end [48]. Developing schedules for dynamic job shop scheduling problems is difficult, as due to the dynamic nature of the task, rescheduling must be done whenever the shop situation changes significantly, e.g., due to the arrival of new jobs into the shop system [48]. The benefit of generating dispatching rules rather than schedules is that they cope well with dynamic problems, as scheduling decisions are made only at the points when machines become available, and hence consider all jobs currently waiting at that machine, whether they have been waiting since the beginning or only just joined the queue [16]. Generating dispatching rules also gives the potential for transferability and reusability [16], as dispatching rules which are discovered can be used in other job shop scheduling problems, for example, into shops of larger size, with different processing distributions and behaviours, or with different objective functions.

2.2.1 The Job Shop Model

Once a job arrives into the shop, it joins the queue at the machine which is required to process its first operation. Upon completion of an operation it moves to the next machine required, or exits the shop if its final operation has been completed.

Parameters. The following job parameters are fixed at the point in time of a job's arrival into the shop system.

- N_j: The number of operations of job j.
- $O_j = \{\sigma_{j,1}, \sigma_{j,2} \cdots, \sigma_{j,N_j}\}$: Set of operations of job j.
- $p(\sigma_{j,i})$: Processing time of operation i of job j.
- $m(\sigma_{j,i})$: Machine required to process operation i of job j.
- w_j: The importance weighting of job j.
- d_j: The assigned due date of job j.
- r_j: The release date of job j into the shop.

Variables. Further job properties change dependent on the scheduling decisions made as the job moves through the shop.

- $r(\sigma)$: The ready time of operation σ. This is r_j for the job's first operation, and the completion time of the previous operation for all subsequent operations.
- R_m: The time machine m becomes idle.
- C_j: The completion time of job j.
- f_j: The flowtime of job j, where $f_j = C_j - r_j$.
- T_j: The tardiness of job j, where $T_j = \max\{0, C_j - d_j\}$.

Constraints. There are many constraints which schedule construction and job dispatch must take into account [110].

- Operation $\sigma_{j,i}$ must be completed before operation $\sigma_{j,i+1}$ can begin processing.
- No job can be in process on more than one machine at any given time.
- Only one job can be processed by a machine at any given time.
- No job can begin processing before its arrival into the shop.
- Job routing is fixed with no alternative routing allowed.

Objective Functions. The following performance measures are commonly used in job shop scheduling. In each case the notation for the γ field and the defining formula (to be minimised) are given [110].

- Makespan (maximum completion time): $C_{max} = \max_j C_j$.
- Maximum lateness: $L_{max} = \max_j (C_j - d_j)$.
- Total weighted completion time: $\sum_j w_j C_j$.
- Total weighted tardiness (TWT): $\sum_j w_j T_j$.

- Mean tardiness: $\frac{1}{N}\sum_j T_j$, where N is the number of jobs.
- Weighted number of tardy jobs: $\sum_j w_j U_j$, where

$$U_j = \begin{cases} 1 & \text{if } T_j > 0 \\ 0 & \text{if } T_j = 0 \end{cases}$$

These objective functions have been used frequently in job shop scheduling research.

Simplifications to the Job Shop Model. Common assumptions made in job shop scheduling models used in research investigating dispatching rules include:

- Pre-emption, when an operation currently being processed is able to be interrupted and removed from the machine before the operation is completed [28], is not allowed.
- Re-circulation, when a job is allowed to visit a machine more than once [110], is not allowed.
- There are no machine breakdowns, i.e., all machines are continuously available to process jobs.
- Sequence-dependent set-up times, the time taken to set up the machine for job k following job j, are assumed to be zero, or included in the processing time.

Types of Schedules. Given any schedule for processing jobs, idle time can be added in infinitely many ways. Therefore there are infinitely many possible schedules for any job shop scheduling problem [28]. A schedule is called:

Active if it is not pre-emptive and it is not possible to make another schedule where at least one operation finishes earlier than the original schedule, and no operation finishes later than the original schedule [110]. That is, it is not possible to modify the schedule so an operation finishes earlier without making another operation finish later.

Semi-active if it is not pre-emptive and it is not possible to make an operation complete processing earlier without changing the processing order of operations on any of the machines [110].

Non-delay if a machine cannot be idle when there is an operation waiting to be processed [110]. That is, a machine is only permitted to be idle when there are no queued operations waiting in the machine's queue.

2.2.2 Scheduling Algorithms for Static JSS

In this section scheduling methods are presented based on the environment they are used in; first static and then dynamic. There are a number of very different techniques for scheduling in job shops, including disjunctive programming, shifting bottleneck, constraint programming and local search techniques [110].

An optimal schedule for a static JSS problem instance can be found by implicit enumeration (e.g. branch-and-bound), however with a large number of jobs this will be impractical [110]. Early contributions to JSS problems were in the form of combinatorial analysis to develop algorithms. There are algorithms for finding optimal solutions to special cases of static JSS problems, for example Jackson's algorithm [55] for the two-machine job shop with makespan objective function ($J2\,|\,|\,C_{max}$).

Jackson's Algorithm. The most basic environment of all job shop scheduling environments is the static two-machine job shop. In the case where

Algorithm 1 Johnson's algorithm for two-machine flowshop ($X \to Y$) [60].

```
1: while any jobs are not yet scheduled do
2:     Find the smallest processing time $p(o_{j,n})$ of unscheduled jobs
3:     if $m(o_{j,n}) = X$ then
4:         Schedule job $j$ immediately *following* jobs already scheduled at the front of the schedule
5:     else
6:         Schedule job $j$ immediately *preceding* jobs already scheduled at the back of the schedule
7:     end if
8: end while
```

the objective of interest is to minimise the makespan ($J2\,|\,|\,C_{max}$) the *optimal* solution can be found using Jackson's algorithm (Algorithm 2) [55]. Jackson's algorithm uses Johnson's algorithm (Algorithm 1) [60] as a subroutine. Johnson's algorithm is used to solve the static two-machine *flow shop* problem with makespan objective function ($F2\,|\,|\,C_{max}$). In a flow shop every job follows the same route through the machines [110].

Let the two machines of the job shop be called A and B. For Jackson's algorithm, jobs are split into four categories:

1. *A-only* jobs that visit only machine A;
2. *B-only* jobs that visit only machine B;
3. *A* $\to$ *B* jobs that start at machine A and move to machine B; and
4. *B* $\to$ *A* jobs that start at machine B and move to machine A.

Jobs that visit only one machine are ordered arbitrarily, and Johnson's algorithm (Algorithm 1) is used to order the $A \to B$ and to order the $B \to A$ jobs.

For example, consider the $J2\,|\,|\,C_{max}$ problem instance with 20 jobs presented in Table 2.1, where a_j is the processing time required on machine

Algorithm 2 Jackson's algorithm for two-machine jobshop [55].

```
1: Put the jobs with an operation on Machine $A$ only in an arbitrary order: $S_A$.
2: Put the jobs with an operation on Machine $B$ only in an arbitrary order: $S_B$.
3: Use Johnson's algorithm to order the jobs to be processed A→B: $S_{A \to B}$.
4: Use Johnson's algorithm to order the jobs to be processed B→A: $S_{B \to A}$.
5: Order jobs on Machine $A$ as follows: ($S_{A \to B}$, $S_A$, $S_{B \to A}$).
6: Order jobs on Machine $B$ as follows: ($S_{B \to A}$, $S_B$, $S_{A \to B}$).
```

A and b_j is the processing time required on machine B. For jobs with two operations, the order in which machines must be visited is denoted by • in the table.

Placing *A-only* jobs in arbitrary order we have $S_A = (5, 6, 15, 17)$, and placing machine *B-only* jobs in arbitrary order we have $S_B = (2, 12, 20)$. Applying Johnson's algorithm to the $A \to B$ jobs gives processing order $S_{A \to B} = (1, 11, 16, 10, 7, 19, 9)$ and applying Johnson's algorithm to the $B \to A$ jobs finds the sequence $S_{B \to A} = (8, 4, 13, 18, 3, 14)$. Hence the optimal sequences for processing jobs at each machine are:

$$(S_{A \to B}, S_A, S_{B \to A}) = (1, 11, 16, 10, 7, 19, 9, 5, 6, 15, 17, 8, 4, 13, 18, 3, 14)$$

for machine A and

$$(S_{B \to A}, S_B, S_{A \to B}) = (8, 4, 13, 18, 3, 14, 2, 12, 20, 1, 11, 16, 10, 7, 19, 9)$$

for machine B. This is illustrated in the Gantt chart in Figure 2.1.

Branch-and-Bound. Branch-and-bound [72] is an enumerative search algorithm which is guaranteed to find the optimal solution to a combinatorial optimisation problem. Branch-and-bound uses a branching rule which partitions the set of solutions into subsets, and a lower bounding rule

Table 2.1: Processing times and paths of 20 jobs for $J2\,|\,|\,C_{max}$.

Job	1	2	3	4	5	6	7	8	9	10
a_j	1		3	7	3	3	5	6	4	7
b_j	10	5	7	3			4	2	3	6
$A \rightarrow B$	●						●		●	●
$B \rightarrow A$			●	●				●		
Job	11	12	13	14	15	16	17	18	19	20
a_j	3		8	1	5	5	2	7	6	
b_j	5	2	7	3		6		8	3	4
$A \rightarrow B$	●					●			●	
$B \rightarrow A$			●	●				●		

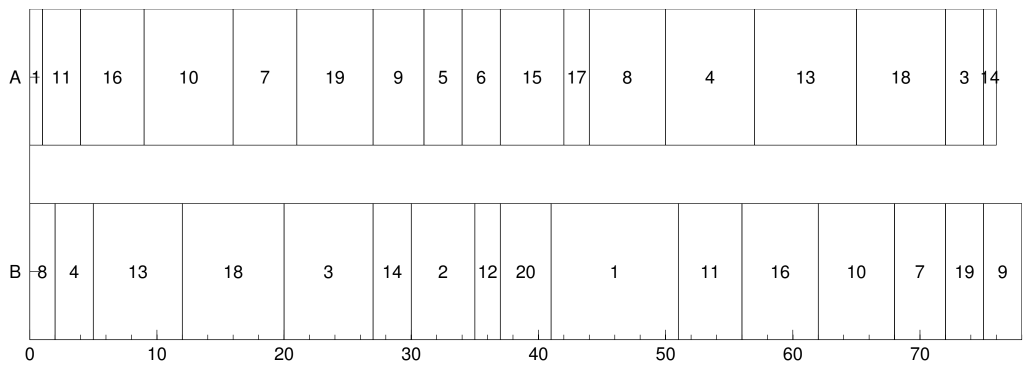


Figure 2.1: Gantt Chart showing optimal processing schedules at machines A and B for the example static two-machine job shop problem instance in Table 2.1.

which computes a lower bound on the cost of any solution in the solution subset being considered [116]. Branch-and-bound was first applied to scheduling problems in the 1960s [116].

Lagrangian Relaxation. Lagrangian relaxation [49] is a strategy for finding an approximate solution to difficult optimisation problems by solving a related optimisation problem. Lagrangian relaxation removes specific integer-valued constraints and instead penalises violation of the constraint in the objective function. Like branch-and-bound, Lagrangian relaxation is computationally expensive for large scheduling problems [62]. Lagrangian relaxation is often combined with branch-and-bound.

Meta-heuristic Scheduling Methods. Local search based methods such as simulated annealing [128] and Tabu search [104] are improvement heuristics. They begin with a complete schedule and try to improve the schedule through manipulation. These methods require a schedule representation, neighbourhood design, neighbourhood search operator and acceptance-rejection criterion. The neighbourhood structure required for a complex scheduling environment, where, e.g., preemption is allowable, often needs to be more complex. The simulated annealing approach proposed by Laarhoven et al. [128] improved the makespan on static JSS problems compared to benchmark methods. However, the computational cost was very large. The Tabu search approach proposed by Nowicki and Smutnicki [104] was used only on very small static JSS problem instances.

Local search based approaches are able to be used in static JSS environments. However, in dynamic environments the available jobs are constantly changing with new (and unknown) arrivals from outside the shop system. This means that the local search solution will quickly become out of date. Local search methods are computationally intensive and to be rescheduling repeatedly would have very large computational consequences.

Artificial intelligence and evolutionary computation methods have been used extensively for scheduling problems. Gene expression programming [36] has been used for scheduling, including the single machine problem [103], dynamic job shop scheduling [102] and flexible dynamic job shop scheduling [101]. Eguchi et al. [35] trained a neural network as a priority rule for dynamic job shop scheduling. The neural network outperformed dispatching rules from the literature on scheduling instances with a variety of utilisation levels and due date tightness parameters. Weckman et al. [134] also investigated training a multi-layer perceptron neural network as a job shop scheduler. A GA was used to generate multiple optimal schedules to small problem instances which the neural network was trained on. The trained neural network performed comparably with a GA scheduler on test instances with six machines and six jobs. The neural network also performed "satisfactorily" on larger test instances up to 20 machines and 20 jobs.

GAs have been one of the most popular meta-heuristic methods for scheduling, with a variety of hybridisation approaches [24]. Gao et al. [40] proposed a hybrid GA and variable neighbourhood search method for the multiobjective flexible job shop scheduling problem. The flexible job shop has a number of work centres, each of which has a set of identical machines. A job has a specified route through the work centres, but any machine within the work centre can be used [110]. The GA chromosome has two parts, a machine assignment vector and an operation sequence vector. Variable neighbourhood search is used to improve the convergence speed of the GA. Variable neighbourhood search [47] is a meta-heuristic which makes systematic changes to a neighbourhood through a descent phase and perturbation phase. The results showed that the proposed method equaled or outperformed existing methods on 157 out of 181 benchmark problems. Zhou et al. [145] also create a hybrid GA with neighbourhood search. Existing scheduling rules which are known to be effective for minimising the makespan are used for scheduling successive operations after the GA has scheduled the initial operation of each machine. Wang and

Zheng [132] combined GAs with simulated annealing in a hybrid optimisation framework for job shop scheduling problems. This framework keeps the generality of GAs and simulated annealing and is easily implemented.

PSO has also been used for job shop scheduling. Sha and Hsu [122] developed a hybridisation of PSO and Tabu search for job shop scheduling. Two particle position representations were compared: priority-based representation and the new preference list-based representation. The results showed that hybrid PSO attained better solutions than other methods. Zhang et al. [142] also developed a hybridisation of PSO and Tabu search. In this work the scheduling environment was the multiobjective flexible job shop. The results showed the hybrid approach to be effective and efficient on tested instances, the largest of which had 15 jobs and 10 machines. Xia and Wu [138] developed a hybridisation of PSO and simulated annealing, also applied to the multiobjective flexible job shop. PSO is used to assign operations to machines, and simulated annealing schedules operations on each machine in the shop. The conflicting objectives were makespan, total workload and critical machine workload. The results showed this approach to be effective, providing good solutions in comparison to three other algorithms from the literature on problems up to 20 machines and 20 jobs.

2.2.3 Scheduling Methods in Dynamic Environments

Dispatching Rules. A dispatching rule (DR) is a scheduling heuristic, so it is not guaranteed to find an optimal solution, but should be capable of finding a reasonable solution in a short period of time [110]. Job shop scheduling has been studied extensively for more than 50 years [116, 48], and DRs have been studied for several decades [110]. This has led to the existence of a large number of dispatching rules for dynamic job shop scheduling. DRs are able to cope with dynamic JSS as scheduling decisions are made the instant that a machine becomes available.

A dispatching rule is used to select which of the jobs currently waiting in the queue at a particular machine will be processed next using various properties of the job, machine and shop. Whenever a machine becomes available, and there are jobs waiting in its job queue, a dispatching rule is used to select the job that will be processed next on that machine. The dispatching rule assigns a priority value to each queued job, and the job with the highest priority value is selected. When a job's operation σ finishes its specified processing time $p(\sigma)$ on machine $m(\sigma)$, the job moves to the next machine on its route, or if the job has now completed all of its operations then the job exits the shop (at completion time C_j) and is delivered to the customer. Dispatching rules are able to provide good solutions in real-time [62].

Pinedo [110] classifies DRs as either static or dynamic, and as either local or global. Static rules are not time dependent whereas dynamic rules are time dependent. Hence, when a dynamic rule is used the priorities assigned to a queue of waiting jobs may change through time, whereas when a static rule is used the priorities will remain constant. Local rules use information from only the queued jobs at the machine. Global rules can use the same information as local rules, as well as information from other machines.

Composite DRs combine a number of basic rules. Each basic rule contributes to the overall ranking given to a queued job by a scale factor, which can be fixed or variable [110]. Dispatching rules are constructive heuristics, as they gradually construct a schedule, one job at a time.

There are many dispatching rules in the literature. No dispatching rule is known to outperform others across all objective functions and shop environments. Some of the simplest and most commonly used rules, which will be used as benchmarks throughout this thesis, are listed below.

First-come-first-served (FCFS) also called first-in-first-out (FIFO) is the traditional queuing model, the job with the smallest $r(\sigma_{j,i})$ is scheduled next.

Shortest processing time (SPT) schedules the job with the shortest processing time, $p(\sigma_{j,i})$, next.

Weighted shortest processing time (WSPT) is the weighted counterpart of SPT, and orders jobs by increasing weighted processing time at the machine [42], i.e., the job that has the maximum weighted shortest processing time, $w_j/p(\sigma_{j,i})$, is processed next.

Earliest due date (EDD) schedules the job with the smallest d_j next.

Minimum slack (MS) The minimum slack of job j is defined by

$$MS_j = d_j - \sum_{l=h}^{N_j} p(\sigma_{j,l}) - t$$

where t is the ready time R_m of the machine, i.e., the time that the scheduling decision must be made and the job with the smallest MS_j is dispatched next.

Slack per remaining processing time (S/RPT) assigns job j the priority

$$\frac{d_j - t}{\sum_{h=i}^{N_j} p(\sigma_{j,h})} \tag{2.1}$$

and schedules the job with the smallest S/RPT next.

SPT and WSPT are components of the Apparent Tardiness Cost (ATC) [130] and the weighted version of COVERT (Cost over Time) [130] dispatching rules, which have good performance for dynamic job shops with the TWT objective.

Apparent tardiness cost (ATC) assigns job j the priority

$$\frac{w_j}{p(\sigma_{j,i})} \exp\left(-\left[\frac{d_j - t - p(\sigma_{j,i}) - \sum_{q=i+1}^{N_j}(W_{ji} + p(\sigma_{j,q}))}{kp_{avg}}\right]^+\right) \tag{2.2}$$

where t is the current time, W_{ji} is the waiting time at operation i which is estimated by $W_{ji} = bp(\sigma_{ji})$ [28, 130], and p_{avg} is the average processing time of the waiting jobs at the machine. Leadtime estimation parameter b is fixed at 2.0 and lookahead parameter k is set to 3.0 as in [130]. The $[A]^+$ notation takes the maximum of A and 0.

WCOVERT assigns priority values as the expected tardiness cost per unit of imminent processing time:

$$\frac{w_j}{p(\sigma_{ji})}\left[1-\frac{(d_j-t-\sum_{q=i}^{N_j}p(\sigma_{jq}))^+}{k\sum_{q=1}^{N_j}W_{ji}}\right]^+ \tag{2.3}$$

where the lookahead parameter $k = 2$. WCOVERT assigns priority 0 if the slack "exceeds some generous 'worst case' estimate of the waiting time" [130].

For an extensive summary and comparison of dispatching rules see [105, 121, 59, 117, 53, 50].

2.3 Genetic Programming

Genetic programming (GP) [70] is an evolutionary computation technique which "aspires to induce a population of computer programs that improve automatically as they experience the data on which they are trained" [7]. A randomly generated population of computer programs is evolved through the application of genetic operators, so that subsequent generations are hopefully fitter (better) than their predecessors.

The optimal length of a solution or function is not always known. One benefit of GP is the variable length of individual programs in the GP population [21]. This allows a far greater number of potential programs to be discovered than fixed length representations such as genetic algorithms.

The most popular representation of GP is tree-based GP [70]. Other representations include linear GP [14], Cartesian GP [82], graph-based GP (e.g. parallel distributed GP [113]) and grammar-based GP [80].

The remainder of this section will describe the key elements of tree-based GP, as this is the form used throughout this thesis. We also introduce grammar-based GP and multiobjective GP as they are incorporated in this thesis.

2.3.1 Representation

In genetic programming the structures undergoing adaptation through the process of evolution form a population that are most frequently represented as syntax trees. They are hierarchical structures, unlike the one-dimensional fixed length linear strings used in genetic algorithms [70].

The search space of genetic programming is the space of all possible rooted trees with ordered branches, where internal nodes are labelled with functions from the function set, and external (leaf) nodes are labelled with terminals from the terminal set. An example of a GP individual is shown in Figure 2.2: this tree represents the expression $((2 * (x + 1)) - y)$. Here $+$, $-$ and $*$ are all elements of the function set. Functions take a specified number of arguments (with minimum of one argument) and cannot be in leaf nodes. Here 1, 2, x and y are all elements of the terminal set, and can only be located as leaf nodes. This search space is equivalent to the search space of all possible LISP S-expressions created by recursively selecting compositions of the available functions and terminals. The LISP S-expression [79] for the GP tree in Figure 2.2 is $(-\ (*\ 2\ (+\ x\ 1))\ y)$.

The available functions and terminals are determined based on the problem domain. The GP function and terminal set should satisfy *closure* and *sufficiency*. The *closure* property requires every function be able to accept any value or data type that can be returned by any function in the function set or terminal in the terminal set [70]. The *sufficiency* property requires that some combination of functions in the function set and terminals in the terminal set is capable of giving a solution to the problem. [70].

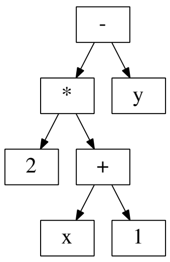


Figure 2.2: GP syntax tree of $((2 * (x + 1)) - y)$.

2.3.2 Population Initialisation

The initial population of genetic programs is generated randomly. The most commonly used approach to generating tree is called the *ramped-half-and-half* method, combining two of the simplest tree generation approaches *grow* and *full* with equal probability. The *full* method generates trees in which all leaf nodes have the same depth; nodes are selected from the function set until the maximum depth is reached, and after that nodes are chosen from the terminal set to complete the tree. The *grow* method generates trees with more variety in shape and size; nodes are selected from the combined set of functions and terminals until the maximum depth is reached, when as for the full method only terminals may be chosen.

The ramped-half-and-half method [70] generates 50% of trees using the full method and 50% of trees using the grow method. Both methods are used with a range of maximum depth limits from two to the maximum depth. This is so that the initial population consists of trees with a variety of shapes and sizes [114]. Figure 2.3 shows three possible tree shapes generated under ramped-half-and-half with a maximum depth of four; Figure 2.3(a) is generated using grow, and Figures 2.3(b) and (c) are both generated using the full method.

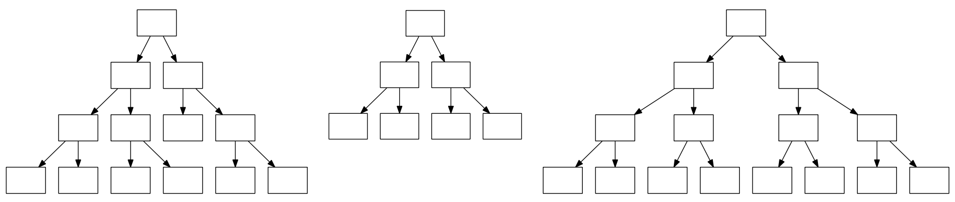
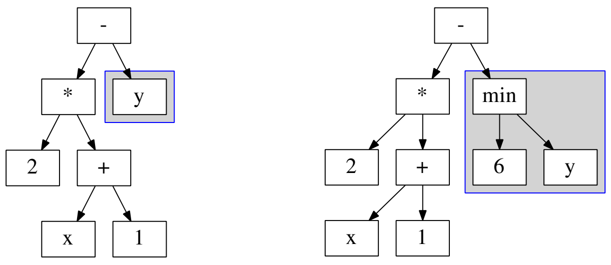

(a) Grow. (b) Full. (c) Full.

Figure 2.3: GP tree initialisation methods, under the ramped-half-and-half method trees of these shapes are able to be generated for any maximum depth greater than three.

2.3.3 Evaluation

Fitness is the “driving force of Darwinian natural selection” [70], and hence of genetic programming. The fitness of an individual in a population in nature is the probability that it survives to reproductive age and reproduces [70], therefore contributing to the future generation. In genetic programming the fitness is a measure used in evolution of “how well a program has learnt to predict the output(s) from the input(s) - that is, the features of the learning domain” [7]. Fitness is measured by a fitness function, which can be either implicit or explicit, and this is used in determining which individuals are selected for the evolutionary operators. The fitness function used will vary depending on the problem domain.

Each GP run will run until a termination criteria is met. This is normally either a predefined number of generations or level of fitness. Once this criteria is met, the best-of-run individual is tested on unseen instances. In multiobjective problems the front of non-dominated individuals (see Section 2.3.7, page 51) is tested on unseen instances. The result of the GP run is this individual or set of individuals, and the results that they are able to attain.



(a) Parent. (b) Child.

Figure 2.4: GP subtree mutation.

2.3.4 Selection

As in Darwinian evolutionary theory, where individuals with greater fitness are more likely to contribute to the gene pool in future generations, genetic programming individuals are selected probabilistically for genetic operators based on their fitness. Tournament selection and fitness proportionate selection are the most commonly used selection methods [114]. *Tournament selection* is based on competition within a population subset [7]. With tournament size n, n individuals are randomly chosen from the population. The n individuals 'compete' against each other, and the individual with the best fitness is chosen to be the parent [114]. Mutation requires a single parent individual, so one tournament is used. Crossover requires two parent individuals so two tournaments are used, one to find each of the parents. In *fitness proportionate* selection, an individual i has probability proportional to its fitness f_i of contributing to the next generation. This probability is given by

$$p_i = \frac{f_i}{\sum_j f_j}. \tag{2.4}$$

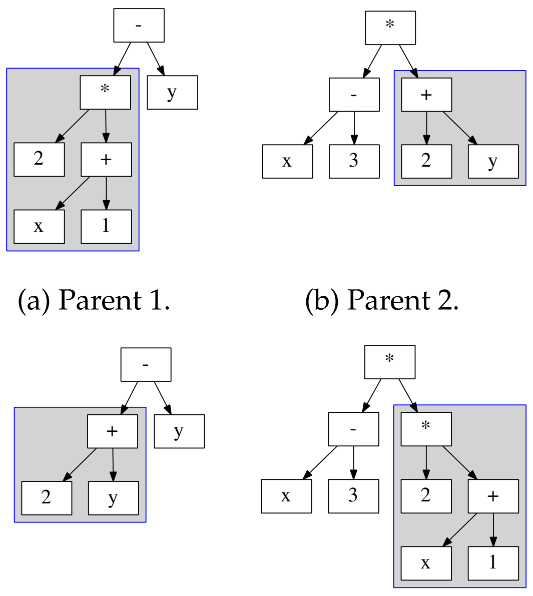
2.3.5 Genetic Operators

There are four main genetic operators used in genetic programming: mutation, crossover, reproduction and elitism. Variation is important in biological populations, since if all individuals are genetically identical then they are all predisposed to the same weaknesses, and the whole population can be at risk. In genetic programming, variation is important to ensure that throughout evolution child populations are not identical to their parents. There are two main genetic operators causing variation in nature which have been taken into genetic programming [7]: mutation of genes within an individual, and the recombination of genes through sexual reproduction. The three operators, described below, are used with specified probabilities to create the child population from the current parent generation.

Mutation occurs in nature when errors are made copying genes [73]. In genetic programming the most common form of mutation is subtree mutation [114], in which a mutation point is randomly chosen, and the subtree routed at that point is replaced with a randomly generated subtree to create the child individual. Figure 2.4 gives an example of mutation: the parent individual, with subtree routed at the mutation point highlighted by the shaded grey box, is shown in Figure 2.4(a), and in Figure 2.4(b) the child individual is shown, with the mutated subtree shaded in grey.

Crossover is the process in which genes of two parent individuals are selected, manipulated and recombined to produce a new set of genes for the child individual [73]. Within genetic programming the most commonly used crossover operation is subtree crossover [114]. This is illustrated in Figure 2.5. Subtree crossover randomly chooses a crossover point (node) in each of two parent individuals. Figures 2.5(a) and 2.5(b) show the subtrees routed at the crossover point highlighted by a shaded grey box. New child individuals are produced by swapping the subtree routed at the selected node of Parent 1 with the subtree routed at the selected node of Parent 2. This gives the two children shown in Figures 2.5(c) and 2.5(d).

Reproduction is when an individual is copied into the next generation. As for mutation and crossover, the selection mechanism is used to select



(a) Parent 1. (b) Parent 2.

(c) Child 1. (d) Child 2.

Figure 2.5: GP subtree crossover.

an individual from the population.

Elitism, like reproduction, copies individuals from the parent population into the population of the next generation. However, elitism selects only the fittest individuals of the parent population up to the prescribed proportion. This ensures that the fittest individuals will not be lost to the evolution process.

2.3.6 Basic GP Algorithm

A basic genetic programming algorithm with minimisation objective function is given in Algorithm 3. This algorithm will return the best performing individual program found through the evolutionary process. This algorithm uses the stages of initialisation, evaluation, selection and genetic operations discussed above. First, the initial population is randomly generated, up to the size specified, N. The evolutionary process then proceeds

Algorithm 3 Basic Genetic Programming Algorithm [114].

```
    Initialisation. Randomly generate an initial population $P$ of $N$ individuals using the available functions and terminals.
 1: $P \leftarrow \{p_1, \ldots, p_N\}$
 2: Set $p^* \leftarrow null$
 3: Set $fitness(p^*) \leftarrow +\infty$
 4: $generation \leftarrow 0$
 5: **while** $generations < maxGenerations$ **do**
    Evaluation. Run and evaluate the fitness value of all individuals.
 6:   **for** all individuals $p_i \in P$ **do**
 7:     **if** $fitness(p_i) < fitness(p^*)$ **then**
 8:       $p^* \leftarrow p_i$
 9:       $fitness(p^*) \leftarrow fitness(p_i)$
10:     **end if**
11:   **end for**
    Generation of next generation's population $P^+$.
    Selection. Select one individual (mutation) or two individuals (crossover) from the population with fitness dependent probability for genetic operators.
    Genetic Operations. Apply genetic operators (with specified probabilities), creating new individuals.
12:   $P^+ \leftarrow \emptyset$
13:   **while** $|P^+| < N$ **do**
14:     obtain $p^+$ by applying genetic operators to individuals selected from $P$.
15:     $P^+ \leftarrow P^+ \cup p^+$
16:   **end while**
17:   $P \leftarrow P^+$
18:   $generations \leftarrow generations + 1$
19: **end while**
20: Return the best individual found, $p^*$.
```

until the stopping criterion, maximum number of generations, is met. At each generation, every individual p_i in the population P is evaluated according to the defined fitness function. If the individual has better fitness than the currently recorded best, the best is updated. Once fitness is determined for all individuals, the next generation is developed through the application of mutation, crossover and elitism to individuals selected using the selection mechanism. The algorithm ends when the maximum number of generations is attained, and the best individual found is returned.

2.3.7 Multiobjective Genetic Programming

Multiobjective GP (MOGP) has been actively studied over the past 18 years [114]. Multiobjective optimisation deals with problems which have multiple conflicting objectives; many real-world problems fall into this category. The aim of multiobjective optimisation is to find solutions which are acceptable for all of the conflicting objectives at the same time [114].

The first approach to multiobjective optimisation combines multiple objectives into a single fitness function. One commonly used form of this aggregate scalar fitness function [114] is a weighted linear combination $f = \sum_i w_i f_i$, where w_i and f_i are the weight and measured fitness of objective i respectively. This approach is easy to implement within standard GP. The disadvantage of this approach is that the relative importance of each objective must be specified *a priori*. *One* GP individual is the result of each GP replication.

The second approach keeps the multiple objectives separate, using the idea of Pareto dominance. Pareto dominance seeks the set of individuals which are *not* dominated by any other individuals. This is known as the *non-dominated* front. A GP individual is non-dominated if it weakly dominates all other genetic programs in the current population. An individual, x, weakly dominates another genetic program, y, if for all i, $x_i \leq y_i$ and there exists a j such that $x_j < y_j$ for i, j in objectives (all objectives to be minimised) [76]. One individual *dominates* another if it is better on at least one objective, and not worse on the remaining objectives. The result of a single replication of MOGP is a Pareto front of non-dominated GP individuals. Two commonly used MOGP algorithms will be used in this thesis.

Non-dominated Sorting Genetic Algorithm II (NSGA-II) [32] is a parameterless elitist strategy. The parent and offspring populations are combined before non-dominated sorting is performed, followed by crowding distance assignment. Non-dominated sorting requires the dominance count, n_p (how many solutions dominate the solution p), and the set of solutions which are dominated by the solution p. Hence, a non-dominated solution with have $n_p = 0$. The dominance count of each individual which is in the set of solutions dominated by p is reduced by one. If this reduces the dominance count of a solution to zero then this is placed in the next non-dominated front. This process repeats until all solutions have been placed in a front. Crowding distance is calculated by calculating the average distance of the two points either side of the solution in its front. The best solutions (by fitness and spread) are selected (using tournament selection) for the genetic operators to create the next generation's population.

Strength Pareto Evolutionary Algorithm 2 (SPEA2) [146] is also an elitist method, and has a separate fixed size archive population. The archive population is updated at each generation with non-dominated individuals. A truncation method is used when the non-dominated front exceeds the fixed size. The fitness of individuals consists of a strength value, which is the number of solutions in the archive and current population dominated by the individual, and density information, which uses the inverse of the distance to the k-th nearest neighbour.

There are other MO tools in the literature, such as MOEA/D [144], ε-MOEA [31], and NSGA-III [84], which are not to be used in this thesis.

2.3.8 Semantic Constraint in GP

Tree-based GP allows unrestricted composition of available functions and terminals, subject only to the tree-depth restriction, which is sufficient for many problem domains, but there is also a history of constraining the search space of GP. Koza [70] introduced the use of "constrained syntactic structures" where programs were restricted by special "problem-specific syntactic rules of construction". The constrained syntactic structures were used, e.g., for symbolic multiple regression where programs were restricted to having a `LIST2` node at the top of the GP tree, to return a fixed number of component values instead of just a single value. The implementation of this required the initial population of programs to have the required syntactic structure, and, further, all of the genetic operators used in evolution were constrained so that the constrained syntactic structure of the program was preserved. The fitness evaluation method may also need to be modified to take into account the structure of the programs.

Keijzer and Babovic [63] developed a system called dimensionally aware GP which uses the domain knowledge that is contained in units of measurement, in order to enhance the search efficiency and the interpretability of resulting output. The introduction of dimensionality to GP was expected to improve search efficiency. This is implemented by making GP nodes maintain knowledge of the units of the measurement it uses. The variable and constant terminals have exponents in their units of measurement. In their experiments, dimensions of length, time and mass were used, with GP tasked with finding the formula of scientific laws from which data was generated. The results show that this form of GP which adds a semantic component is useful, producing output formulations that are "closer-to-correct and easier-to-interpret".

Another way of limiting the allowable expressions within an evolved dispatching rule is to use grammar-based GP [80]. A grammar is used to describe the restrictions on expressions that can be used. A context-free grammar is given by (S, N, Σ, P) notation, where S is the start symbol,

N is a set of non-terminals, Σ is the set of terminals, and P is the set of production rules of the grammar [135]. Grammar-based GP first appeared in the mid-1990s [80, 135]. A major benefit of using a grammar to define program structure in GP is that it is a relatively easy means of restricting the search space and encoding knowledge of the problem [80]. We can use the grammar to generate a population of programs defined by a language and ensure that the programs in the population continue to follow the rules of the language throughout evolution [135].

Montana [85] introduced strongly typed genetic programming (STGP) as an enhanced version of GP. Standard GP is designed for a single data type. Under the closure assumption of GP (see page 44), any function should be able to take as an argument any value returned from a terminal or other function. Strong typing enforces data type constraints, only generating trees which satisfy the constraints. STGP is one form of grammar-guided GP. In STGP the grammar is used to prescribe the type system. STGP does not have all the benefits of other forms of grammar-guided GP; online grammar adaption is not possible in STGP. Basic STGP is equivalent to Koza's constrained syntactic structures [70], except that rather than defining syntax by directly specifying which children each non-terminal can have, this is done indirectly in STGP by specifying the data types of each argument of each function as well as the data types returned by each terminal and function [85]. Montana [85] describes this difference as "relatively minor".

Whigham [135] used a context-free grammar to "define the structure of the language manipulated by a genetic program". The grammar was used in the generation of the program population and to ensure closure was still satisfied with the genetic operators. Crossover was restricted so it can only occur when non-terminals of the same type are selected as crossover points. Mutation replaces a non-terminal with a new sub-tree generated by the grammar with the non-terminal as root node of the sub-tree. In both cases the program is restricted by the maximum depth parameter.

2.4 Local Search

Local search is an improvement heuristic which provides a "robust approach to obtain high-quality solutions to problems of a realistic size in a reasonable time" [1]. Local search uses the idea that making small changes to a given solution may improve it [127]. Local search methods start with an initial solution and try to find better solutions by searching neighbourhoods.

Many local search algorithms for job shop scheduling have been developed [127]. Four elements that must be defined to consider local search of a schedule are: the representation of the schedule, the neighbourhood, the search process within the neighbourhood, and the search criterion [110].

Some basic neighbourhood definitions for local search that can be applied to schedules are transposition, insertion and swap [1]. Let n be the number of jobs in the queue of the machine. *Transpose* swaps two adjacent jobs in the queue (neighbourhood has size $O(n)$). *Insert* moves one job from one position to another and *Swap* swaps two jobs that can be anywhere in the queue (both have neighbourhood size $O(n^2)$).

Choi and Choi [26] combine a local search scheme with mixed integer programming to consider static job shop scheduling environments with sequence dependent set-up times and alternative operation sequences. González-Rodríguez et al. [46] use local search methods for static job shop scheduling with uncertain processing time durations.

With dynamic scheduling problems, the jobs available to be scheduled are frequently changing with the arrival of jobs from outside the shop system and from other machines. This makes the application of local search difficult, as we do not know when another job will arrive, or any of its properties, therefore an optimised queue order is unlikely to still be optimal when more jobs arrive, or when we consider the final objective value function.

2.5 Feature Manipulation

Feature manipulation is the process of altering the input space of a (machine learning) task to improve the performance and (learning) quality [91]. Two forms of feature manipulation are feature selection and feature construction. *Feature selection* is a combinatorial optimisation problem which aims to find a subset of the original features which is as small as possible but still sufficiently describes the problem space [67]. There is often redundancy amongst features in a data set with a large number of features. When the feature set size is reduced, the size of the search space is reduced (and hence the learning performance can potentially be improved). Feature selection methods can be split into wrapper and filter approaches. Wrapper approaches use a learning algorithm to explore the search space; this makes evaluating individual solutions expensive as a classifier must be trained and tested. Filter approaches are independent of any learning algorithm [69], and instead use more computationally efficient heuristics. Filter approaches are more general and less computationally expensive than wrapper approaches [139]. *Feature construction* [77, 92] is another form of feature manipulation; existing features are combined to make new high-level features.

2.5.1 GP based Feature Manipulation

GP has been used for feature selection in many areas. Muni et al. [86] developed an online feature selection approach for classification using GP. This was a wrapper based approach. Neshatian and Zhang [90] used GP in a filter approach to feature selection for binary classification tasks. Their approach explores sets of features, using a multiobjective approach with objectives of maximising the relevance of the feature subset and minimising the size of the feature subset. The results showed this approach improved the classification performance of classifiers and decreased their complexity. Neshatian and Zhang [89] also developed a GP filter approach to feature selection for discovering relevant subsets of features in classification tasks with multi-modal class distributions. The proposed system was effective at ranking subsets and performing feature selection. Ramirez and Puiggros [119] used GP for feature selection and classifier training for classifying the cognitive state of a person based on magnetic resonance imaging data. Chien and Yang [25] used a rough set and GP based method to select significant features and classify numerical data.

GP has also been used for feature construction. Bishop et al. [11] proposed a method for automatically exploring features for classifying images. The results showed improvement on existing manually designed features. Smith and Bull [125] used GP to construct new features for use in the C4.5 decision tree algorithm. Across the 10 data sets used this approach led to an improvement. Krawiec [71] uses GP for feature construction from a set of original attributes in a learning-from-examples paradigm, which also preserves useful features discovered through the evolutionary process. Neshatian et al. [93] developed a new approach using GP for construction of multiple high-level features. The constructed high-level features were then used for classification by decision trees.

2.5.2 Feature Manipulation for Scheduling

Feature manipulation has been used for scheduling problems in several instances. Shiue et al. [124] developed a wrapper feature selection approach using an ensemble of GA based classifiers for real time scheduling. The results show that the method outperformed three machine learning based real time scheduling systems. Piramuthu et al. [112] investigated constructing features for use in scheduling in a flexible flow system. Flexible flow systems are similar to flexible flow shops, except that jobs are able to skip an intermediate operation. The results of constructing boolean features from primitive features were promising, and the proposed framework led to improved representation through the compact size of the decision trees and the smaller set of relevant features.

It has been acknowledged that one challenge of hyper-heuristic approaches to automatically discover dispatching rules is to identify a suitable set of attributes [17]. This is because there are so many possible attributes in scheduling environments, and it is "important to include all relevant characteristics of the problem environment" [17]. However, the larger the number of attributes used in the terminal set, the larger the search space.

2.6 GP for Generating Heuristics for JSS

Hyper-heuristics are "automated methodologies for selecting or generating heuristics to solve hard computational search problems" [18]. Hyper-heuristics search the search space of heuristics (rather than searching the solution space directly), with an aim to increase the robustness of search methods [21].

Many evolutionary computation methods have been used as hyper-heuristics. Hyper-heuristic methodologies have been applied to many application domains [20], including examination timetabling using grammar-based GP [4, 109], bin packing using GP [22], workforce scheduling using reduced neighbourhood search [120], sports scheduling using reinforcement learning heuristic selection [44], quay crane scheduling using GP [97] and vehicle routing using a hybrid heuristic selection approach [41].

Hyper-heuristic approaches can be classified as either heuristic generation or heuristic selection. *Heuristic selection* develops frameworks to adaptively choose between appropriate existing heuristics dependent on the current state of the shop [20]. This thesis focuses on *heuristic generation*, in which *new* heuristics are created. The new heuristics are created from small component building blocks, such as common statistics and operators used in existing heuristics [20].

2.6.1 Genetic Programming for Automatic Generation of DRs

GP based hyper-heuristics (GPHH) are very useful in dynamic JSS environments, and in large scale static JSS environments. In smaller scale static JSS problems optimal solutions are able to be found using optimisation methods.

Single Machine Scheduling

Jakobović and Budin [56] used a GP based hyper-heuristic approach to create priority functions for the static and dynamic single machine shop and for the static job shop. Their GP approach "scheduling with adaptive heuristic" used three trees: a decision tree to determine which heuristic should be used at a given time, and two scheduling heuristics which were used dependent on the value returned by the decision tree. Results showed that for given problems the heuristics evolved performed better than existing scheduling methods. Dimopoulos and Zalzala [33] used GP to investigate the one machine scheduling problem with total tardiness objective function. A number of the evolved rules generalised well, and performed comparably with human designed rules. Yin et al. [140] investigated the one machine problem with stochastic machine breakdowns. In this case both tardiness and schedule stability were objectives. They proposed a tree structure with two subtrees; one for assigning the priority value and the other for calculating the idle time to be inserted before processing which acted as a buffer against disruptions. The results showed this performed better than known heuristics.

Geiger et al. [42] proposed a GP based approach to automatically learn effective DRs for several single-machine environments. In the static and dynamic single machine environments three different objectives were considered (total completion time, maximum lateness and total tardiness). The system evolved rules that were equivalent to known optimal DRs in the situations where there are known optimal DRs, and where no known optimal solutions the system evolves rules that perform no worse than the best known DRs from the literature.

Flow Shop Scheduling

Geiger et al. [42] also investigated the static balanced two-machine flow shop (all jobs follow the same path through the machines) with makespan objective ($F2\,||\,C_{max}$). A flow shop is balanced when the average processing times of operations is the same on all machines. Two approaches were suggested: using a single dispatching rule for both machines, and using a separate dispatching rule for each machine. Johnson's algorithm [60] is known to provide an optimal solution. The best evolved individuals were quite competitive with Johnson's algorithm. However it is not clear if any of the evolved rules will schedule jobs so as to obtain the optimal makespan, and thus be equivalent to Johnson's algorithm. The two-machine job shop is more complicated than the two-machine flow shop, as there are jobs with different paths through the machines and some jobs with only one operation.

Vázquez Rodríguez and Ochoa [129] proposed a GPHH approach to finding variants of the Nawaz, Enscore and Ham (NEH) heuristic [87]. The NEH heuristic is for the permutation flow shop (the flow shop where queues are assumed to use the FIFO rule [110]) with all jobs available at the same time and makespan objective function. It ranks and sorts jobs according to a certain criterion dependent on job parameters and uses this ranking to build a schedule. The aim was to find a variant of the original algorithm that is more effective on problems with extra constraints or different objective functions that are "more in line with real world requirements." Five objective functions were considered independently. Results showed that the evolved variant, NEH$_{GP}$, outperformed original NEH on the cases tested. NEH$_{GP}$ was also shown to scale well with problem size.

Job Shop Scheduling

Jakobović et al. [57] investigated a GP based approach for the development of priority dispatching rules for the parallel machine environment, with between 3 and 20 machines (and 12–100 jobs). The parallel machine environment has m machines in parallel, each with a specific speed (s_i) at which it processes operations. This means the time taken to process a job's operation is different from machine to machine, given by $p(\sigma)/s_i$. For the simple static case, GP achieved the best result for tests with TWT. However, for weighted flowtime and makespan test objectives, existing rules were better. When sequence dependent set-up times were included, GP obtained very good performance over more criteria than existing rules. In the dynamic job shop environment, experiments were again conducted with and without sequence dependent set-up times, using two objective functions: TWT and makespan. "Inserted idleness with arbitrary priority" was used, so that waiting for jobs that have not yet arrived were allowed but the priority function was required to take this time into account. Results showed that GP could "easily outperform other heuristics for arbitrary scheduling criteria." However it is noted that a single heuristic is unlikely to perform well over multiple criteria; however to obtain good performance for a specific objective, GP evolved heuristics provide a "good choice". As for the static case with sequence dependent setup times, in the dynamic case the evolved rules easily outperformed existing heuristics.

Complex Scheduling Environments

Pickardt et al. [107, 108] developed a two-stage (GP and GA) approach for evolving work centre-specific rules for semi-conductor manufacturing. A work centre is a set of 1 to m identical machines, and the shop contains 36 machines in 31 work centres. The method uses GP to generate composite dispatching rules and a GA, where each gene is a specific work centre, to search for a good combination of work centre specific rules to minimise the mean weighted tardiness. The two-stage method is dependent on the composite rules evolved at the GP stage. The two-stage method was shown to be superior to benchmark rules, although only 20 independent runs were completed which is usually not enough for statistical significance testing. Further the two-stage method was shown to be more effective than applying GP or EA separately and selecting the better rule set.

Myopic Nature of Dispatching Rules

The majority of scheduling rules evolved by most GP based approaches lack a global perspective, as the features of the jobs, machines and the shop used are mainly limited to the current state of the current machine and its queue [42, 2]. This means that decisions are made independently based on current and local conditions, without consideration of how movement of jobs amongst machines or the state of the wider shop may be affected [17]. In real-world scheduling situations there is interaction between scheduling decisions, and these interactions must be investigated [94]. A schedule is *non-delay* if a machine cannot be idle while there is an operation awaiting processing [110]. Dispatching rules generally create non-delay schedules.

This lack of global perspective has been noted as a major disadvantage of dispatching rules [17, 108]. The "limited horizon" of dispatching rules may also be why no single rule has been found which is able to outperform all other rules across all shop scheduling environments [108]. The challenge is to design "local, decentralised rules which result in a good global performance of a given complex manufacturing system" [15].

Several works [43, 98, 50] incorporate "look-ahead" in which a machine is notified that a job will be arriving once it is finished its current operation. When the jobs in the queue at the machine are evaluated, if the job which has not yet arrived has the highest priority then the machine is able to enforce idle time, and wait for the job to arrive.

Gere [43] proposed several heuristics which were used in conjunction with priority rules. One heuristic, the "look-ahead" heuristic, checks if there is a critical job due to reach the machine at a future time, but before the scheduled job is finished processing. If there is such a critical job this can replace the scheduled job. Results showed that combining look-ahead heuristics which consider the future state of the shop with dispatching rules improved schedules significantly.

Nguyen et al. [98] developed a GP method to learn iterative dispatching rules (IDRs) for the static JSS problem. IDRs are functions of not only the properties of a machine and the job queued there, but also of the recorded properties of a schedule obtained previously. This allows the IDR to potentially correct mistakes that have been made in previous iterations. The evolved IDRs perform better than benchmark DRs. Evaluation may take longer in the proposed GP method than existing GP. A look-ahead strategy component was also investigated, and results showed it to "play a very important role to enhance the performance of IDRs and the proposed method can effectively evolve suitable look-ahead strategies to deal with specific JSS problems." Look-ahead allows consideration of active or hybrid schedules, not only non-delay schedules.

Hildebrant et al. [50] developed dispatching rules for the dynamic ten-machine job shop, with the minimise mean flowtime objective. They used two "less-myopic" terminals in their GP terminal set. These were the processing time of the next operation in the job, and work in next queue (WINQ). WINQ estimates the waiting time before the job could begin processing on the next machine using the sum of the processing times of jobs waiting in the queue already. They compared the approach of changing the problem instances used for the simulations at each generation throughout evolution with using the same problem instances at every generation throughout evolution. Their results suggested that changing the problem instances at every generation gives the most improved performance. The results with look-ahead incorporated had a high variance. In some runs only "rather bad" solutions were found, yet the best performing rule outperformed the best found without using look-ahead. The authors note that if look-ahead is not used properly, in situations with high utilisation rates the system could end up with utilisation greater than one, in which case a queue will build up.

Interpretability of GPHH evolved Dispatching Rules

The interpretability of evolved dispatching rules is important, as the better a dispatching rule is able to be interpreted, the more trust can be placed in its performance. Individual dispatching rules can be examined and modified, but this is a time consuming process requiring trial and error and human domain knowledge.

There is very little work which considers the interpretability of dispatching rules evolved by GP. However it has been noted that the rules discovered by GP are not generally semantically correct [58]. Jakobović and Marasović [58], in their work evolving priority functions through GP within a meta-algorithm to form a scheduling heuristic, noted that "GP solutions are not 'analytically correct' ", further commenting that this is often true for all GP applications. This is due to how standard tree-based GP allows the unrestricted composition of available functions and features, subject only to the tree-depth restriction. The authors suggest that solutions could be restricted so that only semantically correct expressions remain (e.g. not allowing processing time minus the number of unscheduled jobs). The authors state that preliminary results showed no statistically significant differences compared to standard tree-based GP. However it is not stated whether comparisons were made only in terms of performance on test cases or also included consideration of the interpretability, reliability and generalisation ability (robustness) of the evolved rules, which we believe are important aspects that must be taken into account when considering how 'good' a DR is.

Means of improving the interpretability of genetic programs previously employed include online rule simplification techniques [66, 143], analysis and visualisation of priority indices [15], and identifying weaknesses in decision logic of a given dispatching rule [17].

One means of forcing dispatching rules to be analytically correct would be to use one of the forms of semantic constraint in GP discussed in Section 2.3.8 (see page 53). There is very little work using grammar-based GP approaches for the JSS problem.

Nguyen et al. [95] investigated three GP representations for JSS. One representation, R_1, uses a grammar to "make the rules more readable and explainable". The grammar aims to "provide the adaptive dispatching rules the ability to apply different simple dispatching rules based on machine attributes." The two other GP representations used were an arithmetic representation, R_2, and a "mixed representation", R_3, combining the arithmetic and grammar based approaches. Experimental results showed that the evolved rules performed competitively with hybrid GAs from the literature, and performed effectively in dynamic environments [95]. Further, the evolved rules provided better interpretability of *how* the decisions on sequencing jobs should be made than conventional meta-heuristic approaches. Representation R_3 produced rules with better fitness (total weighted tardiness) than representations R_1 and R_2 [95].

The two works already mentioned, Jakobović and Marasović [58] and Nguyen et al. [95], do not compare the interpretability of the DRs evolved with those evolved without semantic constraint. Tree-based GP is the ideal representation for discovery of dispatching rules. However, existing DRs discovered using GP are generally not semantically correct. The few works which have used semantically constrained GP have not shown a detailed performance comparison, let alone any analysis of whether the interpretability is improved. In using GP to discover new DRs, GP is rediscovering domain knowledge. DRs which have greater interpretability will reveal more about the important interactions within job shop environments than DRs that we are unable to understand.

Another factor in the interpretability of evolved DRs is their size. The allowable tree depth for most approaches using GP allows DRs which are much bigger than most DRs from the literature. The maximum tree depths used include eight [96], 10 [83], 14 [58], and 17 [57, 50, 108, 126]. A relatively full tree of depth 17 would be very hard to interpret. The increased size allows more complex DRs, which may be able to discover more of the relationships and interactions amongst shop properties and therefore improve performance. However, the larger the DR, the more difficult it is to understand how it works and compare how a change in, e.g., processing time, would alter the priority of a given job, and the more likely it is to be overfitting to training instances.

Grammar-based GP has been successfully used as a hyper-heuristic for the automatic generation of timetabling heuristics for the exam timetabling problem [4]. The grammar used in the system was based on effective graph colouring heuristics for timetable construction, and slot allocation heuristics. The developed framework was tested on widely used benchmarks and shown to be competitive with other construction techniques, and on many occasions outperformed other hyper-heuristic frameworks.

Existing GP approaches to discovering new dispatching rules for the JSS problem do not include domain knowledge beyond the use of properties of the jobs, machines and system as a whole. The use of grammar-based GP methods offers the benefits of including the domain knowledge of the types of properties that exist, e.g., counts and times. Further, the search space at the heuristic level which GP operates in is very large, and the restriction of using a grammar-based approach will reduce the search space. These benefits have the potential to improve human understanding of the evolved rules. For the JSS problem we want to better understand what makes a DR perform well, and of almost equal importance, what makes a rule perform badly. The aim of using a grammar-based or strongly typed GP system is to evolve rules which are more interpretable. It is desirable to know *why* a rule works – not just whether it does – as this will help us to learn about the shop system and whether the rule would transfer to another shop environment.

Coping with Multiple Objectives

When there are multiple objectives of interest, single objective GP should be used if the objectives are not conflicting and MOGP should be used if the objectives do conflict. There have been a large number of works using MOGP for various problems [9, 10], including a small number of works which use multiple objectives within GP for automatic generation of DRs. Some combine objectives using a weighted average, relying on the assumption that the relative weighting of importance of each objective is known in advance. Further, this approach does not enable the discovery of more information about the properties of jobs and machines in JSS problems that are useful when different objectives are of interest. Miyashita [83] compared three approaches to automated generation of dispatching rules for static JSS: a homogeneous agent model, where every machine uses the same rule evolved by a single GP process; a distinct agent model, where every machine learns a distinct rule; and a mixed agent model, where a bottleneck agent and a non-bottleneck agent learn rules. The biased combination of weighted tardiness and work in progress (WIP) was the objective to minimise. Tay and Ho [126] used the weighted mean of makespan, mean tardiness and mean flowtime in their work using GP to evolve composite dispatching rules for the dynamic multiobjective flexible job shop problem. They aimed to evolve rules which are robust and effective and outperform existing DRs from the literature. The results showed that the composite rules evolved by GP were robust and outperformed the single DRs and composite DRs from the literature on five datasets.

One significant work by Nguyen et al. [100] develops four new multi-objective GPHH (MO-GPHH) methods for automatic design of scheduling policies. Each scheduling policy consists of a DR and a due date assignment rule (DDAR). Two commonly used MO algorithms are used: non-dominated sorting genetic algorithm II (NSGA-II) [32] and strength Pareto evolutionary algorithm 2 (SPEA2) [146] (refer to Section 2.3.7 on page 51), as well as a harmonic distance based multiobjective evolutionary algorithm (HaDMOEA) [133]. The harmonic average distance of an individual to its k nearest neighbours is the number of neighbours divided by the sum over all neighbours of the inverse distances. The first approach investigated represents the DR and DDAR components as subtrees of a GP tree. NSGA-II, SPEA2 and HaDMOEA are used to explore the Pareto front of non-dominated scheduling policies. In addition, a new approach, diversified multiobjective cooperative evolution (DMOCC), uses coevolution [115] to evolve a dispatching rule in one sub-population and a due date assignment rule in a second sub-population, and selects individuals from the sub-populations for genetic operators based on the crowding distance and non-dominated rank. Binary tournament selection is used to select individuals for collaboration between the sub-populations. The objectives of to be minimised were makespan, total weighted tardiness and the mean absolute percentage error of the rules performance to a known solution. Makespan and total weighted tardiness are conflicting objectives. It is easiest to see this by looking at a single machine with a queue of jobs. When the queue is processed (without any further arrivals) the makespan will be the same no matter how the jobs are ordered, whereas the total weighted tardiness will vary dependent on queue order, as it compares each job's completion time (which is dependent on the job's position in the queue) to its due date. Results showed that the evolved scheduling policies outperformed existing scheduling policies formed by combining popular DRs and dynamic DDARs, and that DMOCC evolved Pareto fronts were better than NSGA-II and SPEA2.

Nguyen et al. [96] investigate a MOGP based hyper-heuristic approach for dynamic job shop scheduling. The MOGP approach aims to minimise five objectives: mean flowtime, maximum flowtime, percentage of tardy jobs, mean tardiness, and maximum tardiness. A front of non-dominated dispatching rules is evolved, which was shown to include very effective, robust rules. Flowtime and tardiness are conflicting objectives. Flowtime, how long a job is in the shop system, is not dependent on a job's due date, whereas tardiness is how late a job is completed after its due date. Therefore minimising mean flowtime will not necessarily minimise mean tardiness.

Nguyen et al. [99] developed a two stage learning and optimisation system to solve the multiobjective order acceptance and scheduling problem. This problem not only has to schedule jobs, but must first choose which jobs arriving in the system will be accepted. The objectives of interest are to maximise the total revenue and minimise the mean absolute error. The system first uses MOGP to evolve a set of non-dominated scheduling rules. The second stage is applied when order acceptance and scheduling decisions need to be made. Each rule constructs a solution, which are turned into a population of solutions for an evolutionary multiobjective method (EMO). The EMO method searches the solutions and outputs a non-dominated front. This method is shown to be competitive with other optimisation methods.

2.6.2 Other Evolutionary Computation Based Methods for Scheduling

Section 2.2.2 (see page 38) mentioned several works where PSO and GA have been used for scheduling.

Ingimundardotti and Runarsson [54] developed a supervised learning approach using logistic regression to find JSS schedules with optimal makespan. The model used a simple linear combination of features found using a linear classifier based on optimal schedules (the optimal sequence of jobs at each machine is known). The effectiveness of the approach is shown by improving on existing dispatching rules for JSS. Results show that the evolved rules are not robust towards different data distributions in some cases, but that they outperformed the three conventional single priority based DRs tested from the literature: SPT, MWRM (most work remaining) and LWRM (least work remaining) [28].

Chryssolouris and Subramaniam [27] proposed a GA based schedule permutation approach to scheduling. Mean job tardiness and mean job cost are the objectives in a dynamic job shop with alternate routings and unreliable machines. The results showed that the proposed method was significantly superior to common dispatching rules.

Shen and Yao [123] used a multiobjective evolutionary algorithm (MO-EA) to solve dynamic flexible JSS problems. They used a measure of schedule stability during rescheduling and maximal machine workload as objectives alongside makespan and tardiness. The MOEA approach is very different from using GP to develop DRs for dynamic scheduling. The EA uses an operation sequence vector and machine assignment vector to construct a schedule of jobs to be processed at each rescheduling point. In other words, an initial schedule is created and then rescheduling is triggered each time an urgent job arrives or a machine breakdown occurs.

2.7 Chapter Summary

This chapter has reviewed the necessary key concepts of genetic programming, job shop scheduling, hyper-heuristics, multiobjective optimisation, feature manipulation and local search. A review of related work in the area of automatic discovery of dispatching rules has also been presented, showing the automatic discovery of new, more effective dispatching rules for job shop scheduling problem through genetic programming is an interesting and promising direction for continued exploration. The ideal nature of GP representation for the evolution of dispatching rules has been discussed, as has other benefits, leading to the wide body of work exploring the automatic generation of dispatching rules in this way rather than the time consuming and human knowledge based traditional approach of manual development. There are some identified limitations of the existing body of work, which motivate the work of this thesis.

- The job shop model has many simplifications that are usually assumed in GP based approaches to evolving DRs. One of the common simplifications is the assumption that one DR is sufficient for scheduling at all machines in a job shop, even when they are unbalanced (i.e. the average processing time is not the same across all machines). In simple static shop scheduling environments, there are known optimal approaches. The work of Geiger et al. [42] investigated the static two-machine flow shop with makespan objective function evolved a rule for each machine. The best evolved rules were characteristic of Johnson's algorithm, which always provides the optimal solution, in that the order of jobs was determined at the first machine, and processed as first-in-first-out at the second machine. The best evolved rules were "quite competitive" with Johnson's algorithm, but are not verified to be equivalent. The two-machine job shop is more complex than the two-machine flow shop, but GP evolved DRs are good at scheduling in the flow shop, we want to see whether GP is capable of evolving optimal DRs in this environment. There is also investigation to be done to see whether in unbalanced job shops one rule is sufficient.

- It has been widely acknowledged in the literature that one limitation of dispatching rules evolved using evolutionary computation techniques is their lack of a global perspective. A variety of look-ahead approaches have been proposed. However, the job shop environment has a large number of potential attributes that can be used in the terminal set for GP approaches. There are more complex, wider-looking, properties that can be investigated in comparison to simple properties. Although local search has been used to solve scheduling problems, its use has mainly been limited to static scheduling problems. It is difficult to apply local search in dynamic JSS environments.

- The interpretability of the evolved rules is an interesting issue that has been noted in existing work using GP for the automatic discovery of dispatching rules for job shop scheduling environments. In scheduling problems, particularly in real-world problems, it is important for the human operators involved on the shop floor to be able to understand why a scheduling method is scheduling well (or why it is not) and to be able to trust the rule's performance. This will be of particular importance if rules are to be used in environments which requires generalisation to different processing distributions, number of machines or utilisation levels.

- Job shop scheduling environments have a very large number of potential properties that can be used as terminals for a GP based approach. There are properties of jobs, machines, and the shop as a whole, before even considering more complex terminals. This large number of potential terminals makes JSS an environment in which feature manipulation could be very beneficial to the quality of results obtained. Feature construction for JSS has not previously been explored with GP. Particularly as we seek to evolve DRs with better interpretability, feature manipulation could be very useful in helping to meet this aim.

The following chapters will address how GP can be used to address these issues. Chapter 3 will develop a GP system to investigate the static and dynamic two-machine job shops. Chapter 4 will investigate how GP can be used to discover dispatching rules which take into account the wider state of the job shop and make better decisions over the local and extended decision horizons. Chapter 5 proposes the use of strongly typed genetic programming to enforce semantic constraint and encourage the evolution of semantically correct dispatching rules with greater interpretability. Chapter 6 will continue to investigate evolving DRs with greater understandability using GP, and will explore how the results from GP can be used to perform feature selection and feature construction.

Chapter 3

Dispatching Rules for Two-Machine Job Shop Scheduling

3.1 Introduction

The purpose of this chapter is to explore the capability of GP to automatically discover dispatching rules for some of the simplest job shop environments: the static and dynamic two-machine job shops. Although these are the simplest JSS environments, they provide a test ground for GP methods with lower computational cost than in larger m-machine job shops.

The first focus of this chapter is to determine whether GP *can* automatically discover DRs which are competitive with known optimal approaches on *static* JSS problems. The static two-machine job shop with a makespan objective function can be solved exactly by using Jackson's algorithm [55] (Algorithm 2, see page 36).

The dynamic two-machine job shop environment is more complex than the static two-machine job shop problem but is the simplest dynamic job shop environment. Assuming that the representation of GP is valid for discovering optimal dispatching rules in the static case, the dynamic case is then considered.

The second focus of this chapter is to investigate the trade-offs between simplifications of *dynamic* job shop models and the difficulty/efficiency of evolving dispatching rules for solving these simplified models with a limited amount of computational resources. One common assumption made is that machines in a job shop should be treated identically. The typical approach is to use the same dispatching rule to dispatch jobs at *every* machines in the shop, regardless of whether the machines expect to see the same volume of jobs, or if the machines see jobs (operations) with similar properties. The simplification of the job shop model is that only one dispatching rule is used.

3.1.1 Chapter Organisation

The remainder of this chapter is organised as follows. Section 3.2 determines whether Jackson's algorithm for the static two-machine job shop is able to be represented as a dispatching rule. Section 3.3 investigates whether GP is able to evolve dispatching rules which are competitive with Jackson's algorithm on the static two-machine problem. Sections 3.4 explores the dynamic two-machine job shop. Section 3.5 concludes with the findings of this chapter.

3.2 Optimal Dispatching Rule Representation for Static JSS

This section focusses on the simplest job shop scheduling environment: the static two-machine job shop. The static two-machine job shop with a makespan objective function, $J2||C_{max}$, can be solved exactly by using Jackson's algorithm [55] (see page 36) in $O(n \log n)$ time, where n is the number of jobs. There are a very large number of attributes of jobs, machines and the job shop system as a whole that can be used as features in the GP terminal set. Some of these attributes may not be necessary or useful as components of a dispatching rule. In Johnson's algorithm for the two-machine flow shop $F2\,|\,|\,C_{max}$ (see page 35) the only properties that are needed are the processing times of the job's two operations. In Jackson's algorithm for the two-machine job shop $J2\,|\,|\,C_{max}$ (see page 36), again the processing times of each job's operation or operations are required and the machine sequence (A$\rightarrow$B, B$\rightarrow$A, A-only or B-only); no wider shop or machine properties are needed. The notation used in this section is given in Table 3.1; for descriptions of the attributes of jobs and operations see Section 2.2 (see page 31).

The traditional way of finding an optimal schedule for a static scheduling problem is to examine all the jobs and construct an entire production schedule before processing begins. Using a dispatching rule to schedule jobs in the static environment is very different. The dispatching rule is called to make a scheduling decision each time a machine becomes available, choosing from amongst the jobs already available at the given machine.

Considering optimal scheduling at a particular machine with a queue of jobs using a dispatching rule in the $J2\,|\,|\,C_{max}$ environment, jobs with two operations remaining *must* be assigned a higher priority than jobs which have only one operation remaining (regardless of whether it is their only operation or if their first operation has been completed and only their second operation remains). However the jobs with only one operation remaining can be scheduled arbitrarily, as the order of their completion does not affect the total processing time, therefore the makespan remains the same for all orderings.

The optimal processing order at Machine A can be simplified to:

$$(S_{(A \rightarrow B)@A}, T_{A \cup ((B \rightarrow A)@A)}),$$

and the optimal processing order at machine B can be simplified to:

$$(S_{(B \rightarrow A)@B}, T_{B \cup ((A \rightarrow B)@B)}).$$

Table 3.1: GP terminal set for static two-machine JSS problem with makespan objective GP system.

Notation	Description	Value
PR	Processing time of current operation	$p(\sigma_{j,h})$
RT	Remaining processing time of job	$\sum_{l=h}^{N_j} p(\sigma_{j,l})$
LV	Arbitrarily large constant integer value	

Here $S_{(A\to B)@A}$ is the schedule of $A \to B$ jobs which are currently at machine A, sequenced according to Johnson's algorithm (Algorithm 1, page 35). $T_{A\cup((B\to A)@A)}$ is the set of jobs which only visit machine A, and $B \to A$ jobs which have already visited machine B, which can be ordered arbitrarily.

A job with two remaining operations will have a longer total remaining processing time than current operation processing time, i.e., RT>PR.

All jobs with two operations remaining that the DR sees must have the current machine for their first operation (otherwise they would be at the other machine). Hence, these jobs need to be ordered according to Johnson's algorithm (Algorithm 1, see page 35). Assuming arbitrarily that the machine in question is machine A, jobs with a shorter processing time on machine A than on machine B will have $\texttt{PR} < \texttt{RT} - \texttt{PR}$, that is, $\texttt{PR} - (\texttt{RT} - \texttt{PR}) < 0$ and these jobs must have decreasing priority value as PR increases. Jobs with equal or longer processing time on machine A than machine B will have $\texttt{PR} - (\texttt{RT} - \texttt{PR}) \geq 0$, and these jobs must have decreasing priority value as RT−PR decreases, but these jobs must all have lower priority than the jobs with shorter processing time on machine A.

This leads to the following expression, shown as a tree in Figure 3.1, which is an optimal dispatching rule:

```
(if>0 (- PR (- RT PR)) (- RT PR) (- LV PR)).
```

The if>0 function takes three arguments; if the first argument is greater

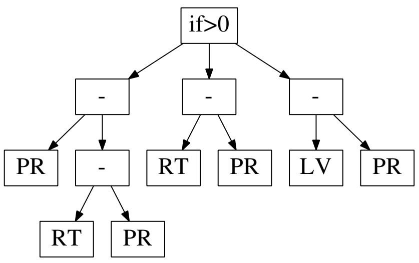


Figure 3.1: An optimal dispatching rule for the static two-machine job shop with makespan objective, represented using tree-based GP.

than 0 then it returns the second argument, else the third argument is returned.

Assume the machine that has just become available is machine A. In the dispatching rule shown above, if there is only one operation remaining on the job then `(- RT PR)` $= 0$ and so `(- PR (- RT PR))` $=$ `PR` > 0. For all such, jobs the priority value returned is `(- RT PR)` which is 0. If `(- PR (- RT PR))` > 0 then the processing time of the first operation is longer than the second, the priority given is the processing time of the next operation, `(- RT PR)`. This gives these jobs decreasing priority, ordered by decreasing processing time at the next machine. As `(- RT PR)`> 0 these jobs will have higher priority than the jobs which are on their last operation. Otherwise, if `(- PR (- RT PR))` < 0, the priority given is `(- LV PR)`, i.e., jobs with shorter `PR` at this machine are given higher priority. Therefore this dispatching rule creates an ordering which gives the same makespan as Jackson's algorithm, although the ordering of jobs it gives may be different.

To illustrate how this optimal dispatching rule works, we will revisit the $J2\,||\,C_{max}$ problem instance with 20 jobs presented in Table 2.1 (see page 37). At time 0, the jobs available at machine A are J1, J5, J6, J7, J9, J10, J11, J15, J16, J17 and J19. The remaining jobs, J2, J3, J4, J8,J12, J13, J14, J18 and J20, available at machine B at time 0. The working for evaluating each job's priority is given in Table 3.2. We have split the jobs by which

Table 3.2: Evaluation of priorities of jobs J1 to J20 using the dispatching rule in Figure 3.1. T indicates `(- PR (- RT PR))` is greater than 0 (i.e. `(- PR (- RT PR))` > 0 is true) and F indicates that `(- PR (- RT PR))` is less than or equal to 0.

Job	`PR`	`RT`	`(- PR (- RT PR))`	>0	`(- RT PR)`	`(- LV PR)`	Priority
			Jobs at Machine A				
J1	1	11	−9	F	–	LV−3	LV−1
J5	3	3	3	T	0	–	0
J6	3	3	3	T	0	–	0
J7	5	9	1	T	4	–	4
J9	4	7	1	T	3	–	3
J10	7	13	1	T	6	–	6
J11	3	8	−2	F	–	LV−3	LV−3
J15	5	5	5	T	0	–	0
J16	5	11	−1	F	–	LV−5	LV−5
J17	2	2	2	T	0	–	0
J19	6	9	3	T	3	–	3
			Jobs at Machine B				
J2	5	5	5	T	0	–	0
J3	7	10	4	T	3	–	3
J4	3	10	−4	F	–	LV−3	LV−3
J8	2	8	−4	F	–	LV−2	LV−2
J12	2	2	2	T	0	–	0
J13	7	15	−1	F	–	LV−7	LV−7
J14	3	4	2	T	1	–	1
J18	8	15	1	T	7	–	7
J20	4	4	4	T	0	–	0

machine they are at at time 0.

We will first look at the queue of jobs waiting at machine A. There are four jobs which have only one operation, J5, J6, J15 and J17. We see in Table 3.2 that the for all these jobs `(- PR (- RT PR))` $=$ `PR` > 0. Following across each job's row of the table, we see that the priority value is assigned by `(- RT PR)` which is 0. The three jobs which have a longer processing time at machine B than machine B are J1, J11 and J16. These jobs have `(- PR (- RT PR))` ≤ 0, hence priority is assigned by `(- LV PR)` giving priorities of `(- LV 1)`, `(- LV 3)` and `(- LV 5)` for J1, J11 and J16 respectively. These priorities are clearly higher than the priority of 0 given to J5, J6, J15 and J17. The remaining jobs, J7, J9, J10 and J19, all have two operations and longer processing time at machine A (`PR`) than machine B (`RT`$-$`PR`). These jobs have `(- PR (- RT PR))` > 0, and priority is assigned by `(- RT PR)`. This gives priorities of 4, 3, 6 and 3 for jobs J7, J9, J10 and J19 respectively. These priorities are greater than 0, but always less than the priorities assigned by `(- LV PR)`. The job with the highest priority is J1 with priority `LV`-1. J1 will be dispatched at time 0 to process for one unit of time, then it will move to machine B.

Similarly, the priority values of all jobs which begin at machine B are found in the final column of Table 3.2. The job with the highest priority is J8 with priority `LV`-2. J8 will be dispatched at time 0 to process for two units of time, then it will move to machine A.

Note that this dispatching rule assigns priority that is independent of time, the priority value assigned to a job does not change while it is in the queue of a specific machine. The priority only needs to be recalculated when the job arrives at a new machine. Therefore we can easily look at the next decision point in this example, at time 1 when J1 finishes processing on machine A. J8 has not arrived from machine B yet, so the waiting job with the highest priority is J11 with priority `LV`-3, which is dispatched for processing for three units of time. At time 2, when machine B becomes available, J1 has joined the queue at machine B. J1 has priority 0 as it has

`(- RT PR)` $= 0$. Therefore, J4 has the highest priority at machine B and is dispatching for processing for three units of time. Following through this example will give the same makespan as obtained using Jackson's algorithm, however, due to arbitrary choices when jobs have equal priority the order in which jobs are processed will differ from that given in the Gantt chart of Figure 2.1.

3.3 Can GP Evolve Optimal DRs for Static JSS?

It has been shown in Section 3.2 that it is possible to create a dispatching rule that will always dispatch jobs in such a way as to minimise the makespan. The next step is to investigate the effectiveness of the GP approach in evolving such a rule given that the required functions and terminals are included in the GP system (i.e. optimal rules exist within the defined search space of GP). The aim of this section is to determine whether GP *can* automatically discover DRs which are competitive with this known optimal approach on the static two-machine JSS problem, $J2||C_{max}$. In this section we outline the GP algorithm, parameter settings and the function and terminal sets necessary.

3.3.1 GP System for Static Two-Machine JSS

The GP system used to evolve dispatching rules for the static two-machine job shop is as follows, using the GP algorithm as described in Chapter 2 (see page 50). We use ECJ20 [78] to implement this system. The GP system randomly initialises the population, where each individual represents a dispatching rule using the ramped-half-and-half method. At each generation, each individual will be applied to the set of training instances. The average makespan value across these instances is assigned as the individual's fitness. After each individual has been evaluated, the genetic operators (crossover, mutation and reproduction (elitism)) are applied to the individuals of the current generation to create new individuals for the next generation. The evolutionary process repeats until the maximum number of generations is reached. At the end of evolution, the individual in the final population with the highest fitness is returned as the best-of-run individual. This individual is evaluated in the testing phase.

Scheduling Algorithm

A simple scheduling algorithm is implemented. While there are unscheduled operations to be processed on the machines, every time a machine becomes available to begin processing an operation, the GP tree is used to calculate the priority of all the available operations waiting at that machine. The operation with the highest priority is scheduled, and the job shop system is updated. Ties are broken using the SPT rule (see page 41). Only jobs that are currently waiting in the queue are able to be scheduled.

Function and Terminal Sets

The terminal set is $\{$`PR`, `RT`, `LV`$\}$ as defined in Table 3.1. The function set is $\{+, -, *, \%,$ `if>0`$\}$. The arithmetic operators take two arguments. The first three arithmetic operators, $+$, $-$, $*$, have their usual meanings. The $\%$ is as usual division except when dividing by zero where the value returned is zero. The `if>0` function is as described earlier.

Parameter Settings

The ramped-half-and-half method [70] is used to generate the initial population with minimum depth of two and maximum depth of four. The population size is 1024 and evolution is for 50 generations. This is a reasonably large population size which has been used in the literature [83, 95], and 50 generations has also been used in the literature [108, 95] although larger numbers of generations have also been used [50, 126, 56]. This population size and number of generations keeps the time of the evolutionary process reasonable. The maximum tree depth is restricted to four, which is deep enough for the DR discussed in Section 3.2.

The rates of genetic operators crossover, mutation and elitism are 40%, 55% and 5% respectively, and individuals are selected for these genetic operators using tournament selection with a tournament size of four. These parameters are different to those usually used in GP, as we are trying to see if it is possible for GP to discover optimal rules.

Training Scenarios

In training, problem instances consisting of exactly 10 jobs were randomly generated so that the job shop is unbalanced. A shop is unbalanced when the expected processing times of jobs is not equal across all machines. The proportions of job types follow 12 configurations given in Table 3.3. The proportions were chosen so that there are problem instances where it is important for jobs with two operations to be scheduled ahead of jobs with only one operation, as well as instances where it is important to select jobs with longer processing times for their second operation ahead of those where the first operation has longer processing time, etc.

The average makespan is found from applying the dispatching rule to eight problem instances of each of the 12 configurations, giving a total of 96 problem instances.

A GP system would not normally have access to the optimal makespan value, so this is a normal GP set up and we are testing whether it can find optimal DRs.

The processing times of operations follow discrete uniform distributions: operations at machine A follow Uniform(100,110) distribution, and operations at machine B follow a Uniform(200,220) distribution.

This work compares using the same problem instances for the entire evolutionary process with changing the problem instances every 10 generations, so the set of problem instances for generations 1–10 is distinct to the set of problem instances used for generations 11–20, etc. This follows

Table 3.3: Configurations of proportions of job types for static two-machine training problem instances.

	Job Types					Job Types			
	A only	B only	A→B	B→A		A only	B only	A→B	B→A
P1	0.40	0.40	0.15	0.05	P7	0.45	0.05	0.45	0.05
P2	0.50	0.30	0.10	0.10	P8	0.35	0.05	0.55	0.05
P3	0.30	0.50	0.10	0.10	P9	0.25	0.05	0.65	0.05
P4	0.60	0.10	0.25	0.05	P10	0.00	0.00	1.00	0.00
P5	0.50	0.20	0.25	0.05	P11	0.00	0.00	0.10	0.90
P6	0.10	0.60	0.10	0.10	P12	0.10	0.10	0.45	0.35

the work of Hildebrant et al. [50].

Testing Best-of-Run Dispatching Rules

The traditional testing phase was replaced by a set of simple test cases to filter out rules that are certainly not optimal. This testing phase leaves us with DRs that made the correct scheduling decisions with all test cases. However it is possible that a DR could correctly make all these decisions, and still not be guaranteed to *always* return the optimal makespan, hence human checking is the final step.

3.3.2 Initial Experimental Results on Static Problems

100 GP replications with distinct pseudo-random number generator seeds were performed with the same problem instances through all generations and 100 GP replications with the *same set* of distinct seeds were performed with changing problem instances every 10 generations, and the best-of-run individual of each was put through the test cases. This is relatively high number of GP replications because we expect GP to find *optimal* DRs rela-

tively infrequently. The pseudo-random number generator seeds are used to set the seed of the psuedo-random number generator which provides the stream of random numbers needed for the evolutionary process. The first random numbers from this stream are used to build the initial population of GP individuals. Therefore, GP replications that use the same pseudo-random number generator seed will have the *same* initial GP population.

Using the same training instances, *no* GP individuals passed the testing phase. Using changing problem instances, of the 100 best-of-run individuals, five GP individuals which passed the testing were manually inspected and shown to be optimal. These five GP individuals are all different variations of the DR proposed in Figure 3.1. Although an unusually high mutation rate is used in obtaining these results, we want to show that it is *possible* for GP to evolve an optimal DR, not necessarily that it is easy for GP to do so. This shows the difference that parameter settings make to GP, which is one of the known disadvantages of GP.

3.3.3 Improvements through Parameter Settings

The rates of genetic operators crossover, mutation and elitism are now altered to 85%, 10% and 5% respectively, and individuals are now selected for these genetic operators using tournament selection with a tournament size of seven. These are more commonly used rates of these genetic operators [95]. With these rates, 100 GP runs with the same set of distinct seeds as the initial experiments were performed with the same problem instances, and 100 GP runs were performed with changing problem instances.

With the same training instances, once again *no* GP individuals passed the testing phase. Of the DRs evolved with changing problem instances, of the 100 best-of-run individuals, 14 passed the testing phase. Manual (human) inspection revealed that 11 of these 14 individuals will always give the optimal makespan. There are nine distinct GP individuals evolved,

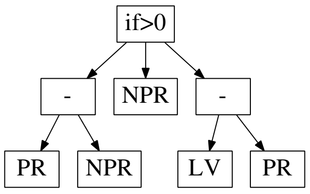


Figure 3.2: An optimal dispatching rule for the static two-machine job shop, using the NPR terminal.

and they are all different variations of the DR proposed in Figure 3.1. This shows that there are many possible dispatching rules within the search space which can guarantee the optimal makespan for any static two-machine job shop scheduling problem. This has almost doubled the number of optimal dispatching rules found.

When the new genetic operator rates were used and the tournament selection size remained at four, no DRs were found that passed the testing phase. This suggests that the high mutation rate was countering the lower tournament size.

3.3.4 Improvements in Representation

We now investigate whether the representation can be improved, through the addition of one more terminal. The terminal NPR is the processing time of the job's next operation. If the job is on its second (and final) operation then the job does not have a next operation and NPR returns 0. This simplifies how the optimal dispatching rule of Figure 3.1 can be written, as shown in Figure 3.2.

We performed 100 independent GP runs with the augmented terminal set {PR, RT, NPR, LV}, and the rates of genetic operators crossover, mutation and elitism as 85%, 10% and 5% respectively and tournament size of seven. This yielded 25 best-of-run evolved DRs which passed the test phase. Upon manual (human) inspection 12 of these 25 are definitely optimal. This is a large increase in the number which pass the testing phase,

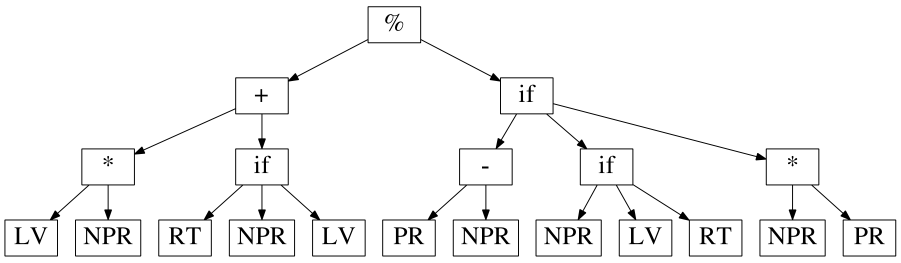


Figure 3.3: An optimal dispatching rule for the static two-machine job shop evolved by GP.

and also an increase in the number which are truly optimal.

The evolved rules which will schedule jobs in such a way that the makespan is always optimal can look very different to the rules shown in Figures 3.1 and 3.2. One such rule is shown in Figure 3.3. To see that this rule schedules the jobs to achieve the optimal makespan, first consider the case that $\texttt{NPR} = 0$, i.e., the job has only one operation remaining. Then this rule returns $\frac{\texttt{LV} * \texttt{NPR} + \texttt{NPR}}{\texttt{RT}} = 0$. This leaves two cases with $\texttt{NPR} > 0$ where the rule will return values from different formulae: $\texttt{PR}>\texttt{NPR}$ and $\texttt{PR}\leq\texttt{NPR}$. When $\texttt{PR}>\texttt{NPR}$, the rule simplifies to $\frac{(1 + \texttt{LV}) * \texttt{NPR}}{\texttt{LV}} \approx \texttt{NPR}$ and it ranks jobs in the same order as NPR as LV is a very large value. The value returned from this is always greater than 0, so these jobs will have a higher priority than those jobs that are on their final operation. Further, jobs with a larger NPR value have higher priority than jobs with a smaller NPR value. When $\texttt{PR}\leq\texttt{NPR}$, the rule simplifies to $\frac{(1 + \texttt{LV}) * \texttt{NPR}}{\texttt{PR} * \texttt{NPR}} = \frac{(1 + \texttt{LV})}{\texttt{PR}} \approx \frac{\texttt{LV}}{\texttt{PR}}$. The value returned by this is always positive, and as LV is a positive constant, the value will be greater than NPR. Further, jobs with smaller PR have higher priority than jobs with larger PR. Hence this rule schedules jobs in the same way as described above, and the resulting schedule will minimise the value of the makespan.

The question being addressed in this section was whether GP can *discover* a dispatching rule which is equivalent to Jackson's algorithm. It was shown that a dispatching rule representation equivalent to Jackson's algorithm existed. This showed that GP could in theory discover such a DR, but not that it will. The second aspect therefore addressed whether GP actually does discover any such DRs. Several representations and parameter settings were investigated, to show that we cannot guarantee that GP will discover an optimal solution, the importance of parameter settings and the impact of the choice of features included in the terminal set. The terminal `NPR` is still a 'simple' feature, as it is directly a job's property.

3.4 Machine Specific Dispatching Rules for Dynamic Job Shop Problems

The aim of this section is to investigate whether the simplification to the job shop model of using the *same* dispatching rule at all machines in an unbalanced job shop (where the mean processing time of jobs is not equal across all machines) is sufficient, compared to using machine specific dispatching rules. There is a trade-off between simplifying job shop models and the difficulty of evolving dispatching rules which perform well in these simplified models with a limited amount of computational resources. The typical approach is to use the same dispatching rule to dispatch jobs at *every* machine in the shop, regardless of whether the machines expect to see the same volume of jobs and jobs (operations) with similar properties.

Here, the question is: in an unbalanced job shop, should dispatching rules be *specialised* to each machine or should *one* dispatching rule be applied to all machines separately? The objective of interest is to minimise the total weighted tardiness, $TWT = \sum w_j T_j$.

The objectives of this section are, within the dynamic two-machine job shop:

1. investigate the performance of GP systems with different representations of rules to machines;

2. compare the performance of evolved rules from all representations with well known dispatching rules; and

3. analyse some of the evolved rules.

The remainder of this section is organised as follows. Section 3.4.1 introduces the two representations. Section 3.4.2 presents the experimental settings and the experimental results are shown in Section 3.4.3. Section 3.4.4 analyses some of the individuals evolved by GP.

3.4.1 Representation of GP Individuals for Machine Specific DRs

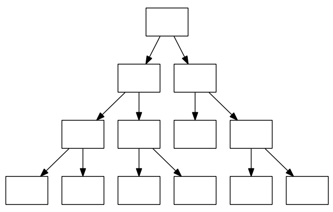
Two representations of scheduling rules are considered. The first representation, $R1$, treats each machine the same; each GP individual consists of one tree, representing a scheduling rule which is used to schedule jobs at both machines independently. The second representation, $R2$, treats machines differently; each GP individual now consists of two trees in which operations on machine A are scheduled by the first tree, and operations on machine B are scheduled by the second tree. Figure 3.4 illustrates the two representations.

3.4.2 GP System for Dynamic JSS

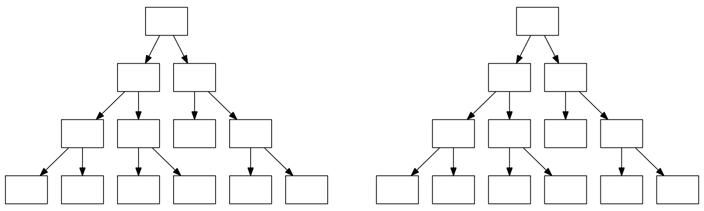
In this section the GP system is developed in detail. First the system for evolving DRs is detailed, followed by the fitness evaluation, training and testing scenarios, and GP parameter settings.

GP System

The GP based system for evolving dispatching rules for the dynamic two-machine job shop is as follows. The GP system randomly initialises the population, where each individual represents a dispatching rule. At each



(a) Example of R1 Individual for the dynamic two-machine job shop GP system. This individual consists of one tree which is used to evaluate the queued jobs at both machines.



(b) Example of R2 Individual for the dynamic two-machine job shop GP system. This individual consists of two trees; the left tree is used to evaluate the queued jobs at machine A and the right tree is used to evaluate the queued jobs at machine B.

Figure 3.4: Representations of tree-based GP dispatching rules for the dynamic two-machine GP system.

generation, each individual will be applied to the set of training instances. After each individual has been evaluated, the genetic operators (crossover, mutation and reproduction (elitism)) are applied to the current generation of individuals to create new individuals for the next generation. In R2, the crossover operator is restricted so that crossover occurs only between trees that schedule jobs on the same machine. The evolutionary process repeats until the maximum number of generations is reached. At the end of evolution the individual in the final population with the highest fitness is returned as the best-of-run individual. This individual is evaluated in the testing phase. A more in depth description of the elements of the GP system is provided in Section 2.3 (see page 43).

Fitness Evaluation

The fitness of a DR is evaluated by a discrete-event simulation model [74] of the job shop, which implements a simple scheduling algorithm. While there are unscheduled operations to be processed on the machines, every time a machine becomes available to begin processing, the individual's appropriate tree (machine independent in R1 or machine dependent in R2) is used to evaluate the priority of all the available operations waiting at that machine. The operation with the highest priority is scheduled, and the job shop system is updated. Ties are broken using the SPT rule (see page 41). Only jobs that are currently waiting in the queue are able to be scheduled. The average total weighted tardiness, $TWT = \sum w_j T_j$, value across a number of problem instances is assigned as the individual's fitness.

Training and Testing Problem Instances

In each problem instance, jobs arrive stochastically according to a Poisson process with rate λ. This is standard in queueing systems and fits real-world queues well [88]. The processing times for machine A and machine B are generated according to an Exponential distribution with mean μ_A and a Uniform$(0.2\mu_B, 1.8\mu_B)$ distribution respectively. Different processing time distributions are used to further differentiate between the two machines. The expected utilisation (the proportion of time a machine is busy), ρ, of machine M is calculated as

$$\rho = \frac{(\lambda \times p_M)}{(1/\mu_M)}, \tag{3.1}$$

where p_M is the proportion of jobs that need to be processed at machine M.

We randomly create problem instances so that the shop is unbalanced. If the job shop was balanced then the same DR is likely to work well, therefore we look at unbalanced job shop problems to see whether there are scenarios where machine-specific DRs may work better. We use the term *scenario* to describe the combination of properties defining the shop, and *problem instance* for an instantiation of the problem scenario with a particular psuedo-random number generator seed. Therefore for one problem scenario there can be many problem instances. Only one problem instance is used from each scenario for both training and testing. This is because each instance requires lots of scheduling decisions to be made. Four different *training* configurations (scenarios TC1 to TC4, shown in Table 3.4) of proportions of job types, arrival rate, mean processing times and utilisation rates are specified. Scenarios TC1 and TC3 are relatively more balanced, TC2 has 95% of jobs visiting machine A and only 70% visiting machine B, and TC4 is even more unbalanced with 90% of jobs visiting machine A and only 30% of jobs visiting machine B. These configurations are designed to ensure the job shop is unbalanced, i.e., the machines see different proportions of jobs, and have different utilisation levels, because it is in this situation that machine-specific DRs are likely to be more useful than a symmetric job shop where each machine is expected to have the same utilisation. Fourteen different configurations (scenarios C1 to C14, shown in Table 3.4) are used in *testing*. The test scenarios include the training scenarios, and additional scenarios with a greater range of utilisation levels.

Due dates are set using Equation (3.2) [5],

$$d_j = r_j + h \times \sum_{l=1}^{N_j} p(\sigma_{j,l}), \tag{3.2}$$

where $h = 1.3$ is a due date tightness parameter. Jobs are given weight 1, 2 or 4, with probability $(0.2, 0.6, 0.2)$ [111]. This is based on the premise of the 20/60/20 rule, that is 20% of jobs are of low importance, 60% of jobs are of average importance, and 20% of jobs are of high importance. The objective being minimised is TWT. The training fitness of an individual is the mean TWT from four simulation runs, one with each of TC1–TC4. A warm up period of 1000 jobs is used, and we collect data from the next 5000 jobs to arrive ($N = 5000$), however new jobs keep arriving in the

Table 3.4: Configurations of proportions of job types for training scenarios (TC1–TC4) and testing scenarios (C1–C14) for dynamic two-machine job shop problem instances.

				Job Types				Utilisation	
	λ	$1/\mu_A$	$1/\mu_B$	A only	B only	A→B	B→A	Mach. A	Mach. B
TC1=C1	0.80	0.80	0.50	0.40	0.05	0.50	0.05	0.95	0.96
TC2=C2	0.80	1.20	0.60	0.30	0.05	0.60	0.05	0.63	0.93
TC3=C3	0.70	0.65	0.40	0.45	0.10	0.30	0.15	0.97	0.96
TC4=C4	0.65	0.60	0.20	0.70	0.10	0.10	0.10	0.98	0.98
C5	0.80	0.50	0.80	0.05	0.40	0.05	0.50	0.96	0.95
C6	0.80	0.60	1.20	0.05	0.30	0.05	0.60	0.93	0.63
C7	0.70	0.40	0.65	0.10	0.45	0.15	0.30	0.96	0.97
C8	0.65	0.20	0.60	0.10	0.70	0.10	0.10	0.98	0.98
C9	0.70	0.60	0.40	0.45	0.15	0.20	0.20	0.99	0.96
C10	0.70	0.60	0.50	0.45	0.15	0.20	0.20	0.99	0.77
C11	0.90	0.80	0.50	0.45	0.15	0.20	0.20	0.96	0.99
C12	0.70	1.00	0.50	0.30	0.10	0.30	0.30	0.63	0.98
C13	0.40	0.30	0.30	0.25	0.25	0.25	0.25	1.00	1.00
C14	0.10	0.70	0.40	0.10	0.20	0.35	0.35	0.11	0.23

system until the 6000th job is completed. In testing, the TWT of a single simulation run using each of the fourteen test scenarios, C1 to C14, given in Table 3.4 are recorded.

We also investigate using the same problem instances at every generation against regenerating the set of training problem instances at every tenth generation, following the work of Hildebrant et al. [50]. In the changing problem instances method, the problem instances used at each

Table 3.5: GP terminal set for dynamic two-machine GP system.

Notation	Description	Value
PR	Processing time of current operation	$p(\sigma_{j,h})$
RT	Remaining processing time of job	$\sum_{l=h}^{N_j} p(\sigma_{j,l})$
RO	Remaining number of operations	$N_j - h + 1$
RJ	Operation ready time	$r(\sigma_{j,h})$
W	Job weight	w_j
DD	Job due date	d_j
RM	Machine ready time	R_m
NQ	Number of operations in the queue	

generation will still be the same for all GP individuals, but the problem instances for the first 10 generations are distinct from the problem instances for the next ten, and so on. Our initial results in Section 3.2 showed that changing the problem instances greatly improved the quality of the dispatching rules found by GP for the static two-machine JSS problem.

Function and Terminal Sets

The terminal set is given in Table 3.5. This is a larger set than used in the static JSS experiments, which only included PR and RT from this table. A description of the attributes of jobs and operations is found in Section 2.2 (page 29). The dynamic JSS problem is more complex than the static JSS problem, and as the weighted tardiness objective requires knowledge of the due date and importance weighting these are included in the terminal set.

The function set is $\{+, -, *, \%,$ if>0, max, min, abs$\}$. The abs function takes one argument and returns the absolute value. The max and min functions take two arguments and return the maximum and minimum of their arguments respectively. These additional non-arithmetic operators help for calculating time differences between the current time and due dates, and whether jobs are tardy, etc. This is a function set which has been used in the literature [108].

Parameter Settings

The population size is 1024, and the initial population is generated using the ramped-half-and-half method [70] with an initial depth of six. GP individuals have a maximum depth of six. Evolution is for 50 generations. Tournament selection with a tournament size of four is used to select GP individuals from the population for genetic operators, as for the initial experiments of Section 3.3. This is larger than a tournament size of three which is used in some works [56, 57] but still less than the standard size of seven. The genetic operators crossover, mutation and elitism are used with rates of 85%, 10% and 5% respectively.

3.4.3 Dynamic Two-Machine JSS Experimental Results

Here we present the test results of each combination of representations R1 and R2, and the same problem instances at every generation (S) versus changing problem instances every ten generations (C), for the dynamic two-machine JSS problem. This gives four methods: R1S uses representation R1 with the same problem instances at every generation, R1C uses representation R1 with problem instances changing every ten generations, R2S uses representation R2 with the same problem instances at every generation, and R2C uses representation R2 with problem instances changing every ten generations.

For each we perform 40 GP evolutionary runs, using 40 common independent seeds.

The boxplots in Figure 3.5 show that the relative performance of the four GP based methods depend on the configuration. Table 3.6 gives the order of performance, from worst to best, of the four methods for each of

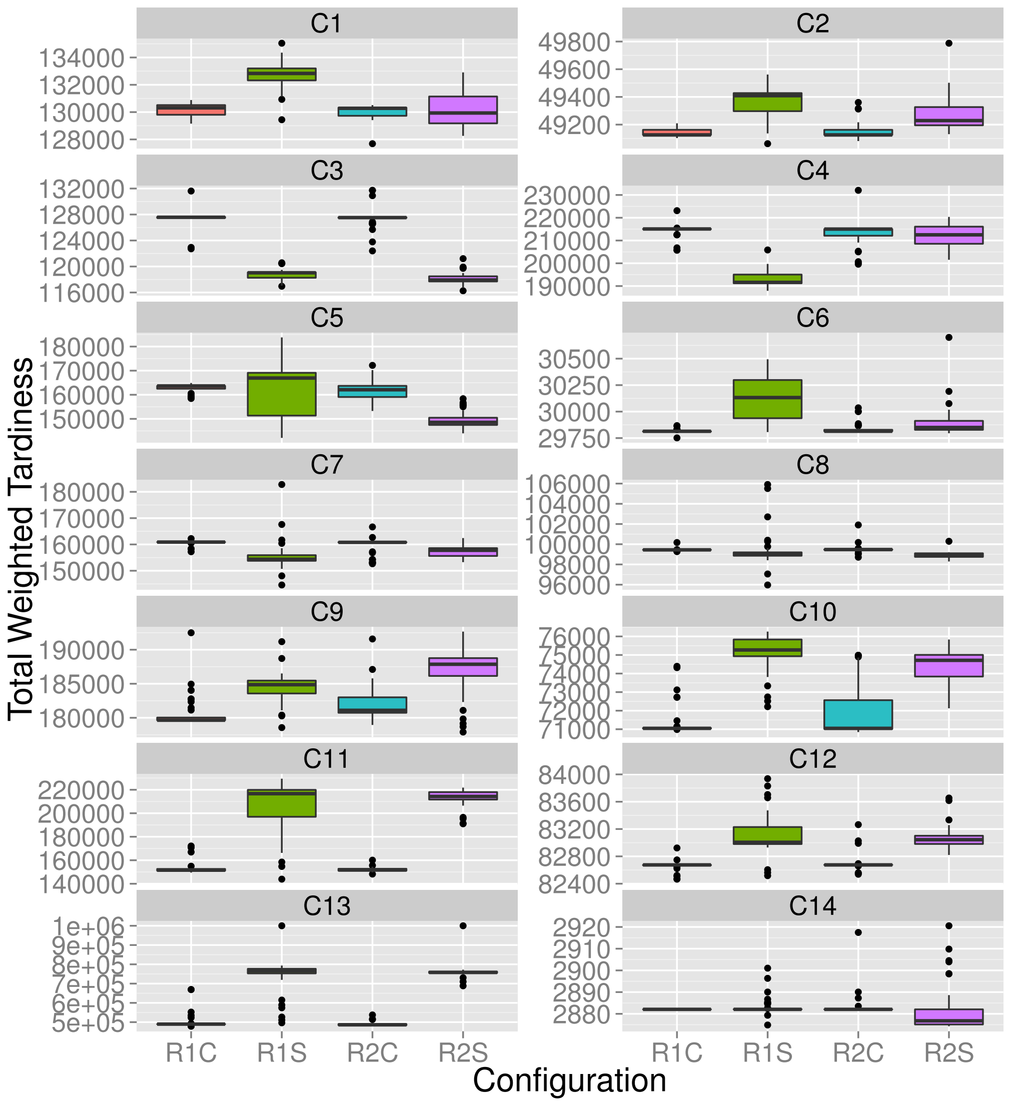


Figure 3.5: Boxplots showing results (lower TWT is better) of methods R1S, R1C, R2S and R2C across the 14 configurations for dynamic two-machine job shop (vertical scales differ between subfigures).

the 14 scenarios. The orders are established by performing pairwise comparisons between GP runs with the same initial population of GP individuals for each combination of methods used. We examined the ratio of the two rules over these combinations for each of the 40 initial populations to see if rules evolved by one method were consistently better than another. Table 3.6 shows that although there is no consistent order, there are some patterns that can be observed. Over the four scenarios with largest imbalance in expected utilisation (C2, C6, C10, C12) the C methods outperform the S methods. Interestingly this is also the case for C13, which is symmetric (machines have the same arrival rate, mean processing time and expected utilisation) and therefore a balanced shop. For the other scenarios where the expected utilisations are within 0.02 and the shop is not symmetric, the S methods outperform the C methods. When the performance of two methods is similar, the C methods are inseparable, and the S methods are inseparable. However we cannot say conclusively that any method is better than any other.

Table 3.7 gives the mean performance (TWT) of three benchmark dispatching rules (WSPT, FCFS, MS) across the 14 test scenarios, and the mean $\pm$ standard deviation of the average performance of each of the 40 GP runs for R1S, R1C, R2S and R2C. The standard deviation of performance is much smaller for the R2 methods both with the same and changing problem instances. Table 3.7 also gives the mean and standard deviation of the evolution time in minutes and the total testing time (for all 14 test configurations) in milliseconds. The R2 methods have the lowest mean evolution time and smaller standard deviations than the R1 methods. The testing time is lower with changing problem instances than using the same problem instances and the standard deviation is also smaller. Further, R2S has lower mean and smaller standard deviation for testing than R1S, and likewise R2C has lower mean and smaller standard deviation than R1C. R2C has the quickest mean test time (approximately 0.4 seconds) and the smallest standard deviation.

Table 3.6: Approximate order of performance of methods on each scenario of dynamic two-machine job shop.

Scenario	Worst $\rightarrow$ Best
C1	R1S $\rightarrow$ {R1C,R2S, R2C}
C2	{R1S,R2S} $\rightarrow$ {R1C, R2C}
C3	{R1C, R2C} $\rightarrow$ {R1S,R2S}
C4	{R1C,R2S, R2C} $\rightarrow$ R1S
C5	{R1C, R2C} $\rightarrow$ R1S $\rightarrow$R2S
C6	{R1S,R2S} $\rightarrow$ R2C $\rightarrow$ R1C
C7	{R1C, R2C} $\rightarrow$R2S $\rightarrow$ R1S
C8	R2C $\rightarrow$ R1C $\rightarrow$R2S $\rightarrow$ R1S
C9	R2S $\rightarrow$ R1S $\rightarrow$ {R1C, R2C}
C10	{R1S,R2S} $\rightarrow$ {R1C, R2C}
C11	{R1S,R2S} $\rightarrow$ {R1C, R2C}
C12	{R1S,R2S} $\rightarrow$ {R1C, R2C}
C13	{R1S,R2S} $\rightarrow$ {R1C, R2C}
C14	R2C $\rightarrow$ R1C $\rightarrow$ R1S $\rightarrow$R2S

Comparison of results from the best-of-run individuals evolved by GP and existing dispatching rules from the literature (WSPT, FCFS, MS) in the table shows that GP is able to evolve rules that are significantly better than these rules.

Due to the way in which the GP crossover operator is restricted in R2 so that crossover occurs only between trees that schedule jobs on the same machine, only one of the GP individual's trees is affected by crossover at each generation. This means that the R2 GP runs could be run for twice as many generations as R1 GP runs [70] (or do twice as many crossover oper-

Table 3.7: Comparison of mean performance, mean±standard deviation of training/evolution and testing times of the four GP methods over 40 runs and three benchmark dispatching rules for dynamic two-machine job shop.

	R1S	R1C	R2S	R2C
TWT	159560±8201	140335±3188	161937±3117	139372±1216
Train (min)	117±31	118±27	80±16	90±21
Test (ms)	613±349	469±153	500±245	392±103

	WSPT	FCFS	MS
TWT	382778	554842	534326
Test (ms)	250.0	290.0	202.0

ations per generation) for the trees to have the same number of crossover operations to see if this makes a difference. However, this is increasing the computational time involved considerably.

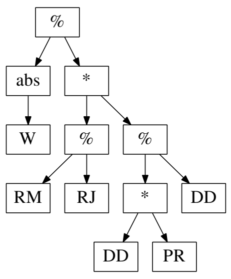
3.4.4 Analysis of Evolved Programs

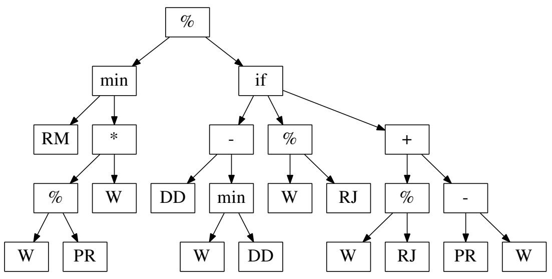
To analyse *why* the dispatching rules work well, we choose some of the best evolved rules to look at more closely.

The two GP individuals from method R1C in Figure 3.6 performed identically. The dispatching rule in Figure 3.6(a) simplifies to

$$\frac{\mathtt{W} * \mathtt{RJ}}{\mathtt{PR} * \mathtt{RM}}$$

As RM will be the same for all jobs in the queue, this can be further simplified to $(\mathtt{W}/\mathtt{PR}) * \mathtt{RJ}$. This is very similar to the WSPT rule weighted by the ready time of the operation. Jobs with earlier ready times (smaller RJ) have lower priority, all else being equal. Interestingly the due date does not appear in this individual, although DD is related to RJ, since both in-





(a) (b)

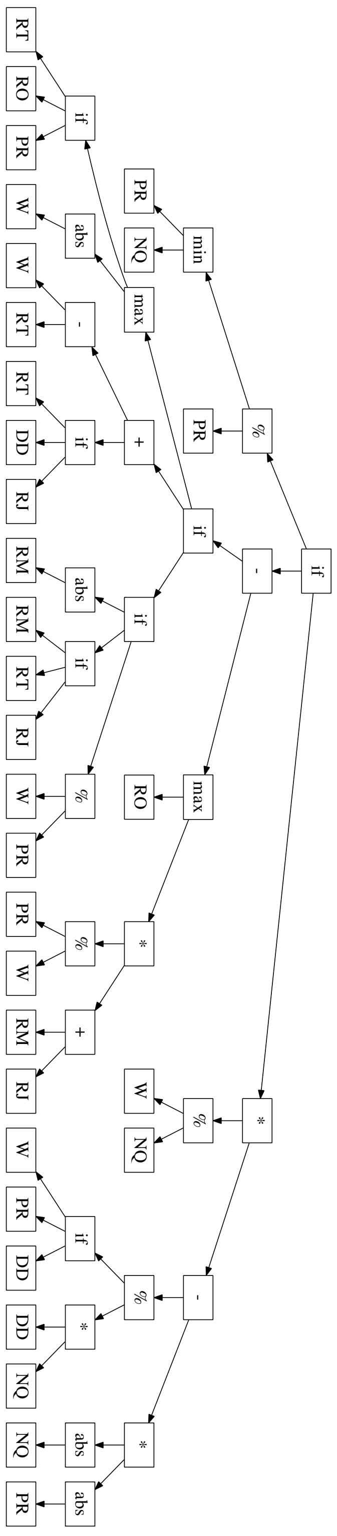
Figure 3.6: Best evolved R1C individuals for dynamic two-machine job shop.

crease throughout simulation time. The dispatching rule of Figure 3.6(b) will generally simplify to be the same as the the dispatching rule of Figure 3.6(a) because in general `DD>W`, and `RM>W*W/PR`. It is interesting to note that both trees do not contain the `DD` terminal in their simplified form, which we would expect to appear as it is used in calculating the tardiness of jobs for the objective function.

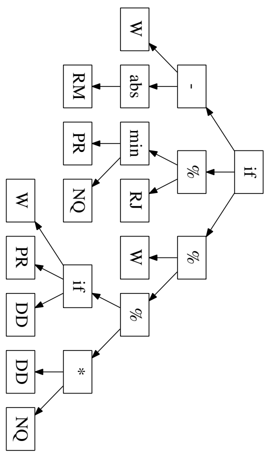
A similar structure is found in the machine B tree of the R2C individual shown in Figure 3.7(b). Once a few jobs have been processed, the ready time of the machine will be greater than the weights of the job (under our settings at least), and from then on the third branch of the `if>0` branch will always be scheduling the jobs. As all job weights are positive this branch simplifies to

$$(\mathtt{W}/\mathtt{PR}) * \mathtt{DD}$$

as the number of jobs in the queue will be the same for all jobs evaluated at a given time. This rule may work well because of the randomly generated data sets. As this is only a two-machine job shop, and jobs have either one or two operations, there is not as much difference in total processing time (and hence due date as it is based on a total processing time multiplied by factor 1.3) as in a larger job shop. Using this rule to schedule *all* jobs in the shop at both machines gives an average TWT of 132229 across the 14



(a) Machine A.



(b) Machine B.

Figure 3.7: Best evolved R2C individual for dynamic two-machine job shop.

scenarios, compared to an average TWT of 137270 across the 14 scenarios when the machine specific dispatching rules are used. This is an unexpected result; we would expect this DR to not be as effective at the other machine in the shop. If the training scenarios had been of more similar form, e.g., machine A always has high utilisation and machine B always has lower utilisation, this result may not occur. Although there is no significant performance difference between the R2 and R1 methods overall, the evolved rules of R2 are evolved for the particular task of scheduling at a specific machine. Not all DRs evolved by R2 will be effective, and we should not expect that those that are effective on their specific task will be effective on a second task (i.e. scheduling at the other machine).

The machine A tree of the GP individual in Figure 3.7(a) is harder to analyse. The middle branch will always be used, and simplifies to

$$\mathtt{W} + \mathtt{DD} - \mathtt{RT} - \max\{\mathtt{RO}, (\mathtt{W}/\mathtt{PR}) * (\mathtt{RM} + \mathtt{RJ})\}.$$

This rule takes the latest time that processing of a job would need to start for the job to not be tardy, adds the weight (jobs with greater weight therefore have higher priority) and subtracts a measure of waiting and processing time. Using this rule to schedule *all* jobs in the shop at both machines gives an average TWT of 2423119 across the 14 configurations, which is considerably worse than the performance of the machine B tree.

Further, all terminals and functions appear in at least one of these "best" evolved trees; however NQ does *not* contribute to the priority value in the trees it appears in.

3.5 Chapter Summary

This work is the first time that GP has been used to evolve dispatching rules that are optimal for the static two-machine JSS problem with the makespan objective function. This was proven by showing that the evolved rule was equivalent to an optimal scheduling algorithm, Jackson's algorithm. From 100 GP runs with the initial parameter settings, five best-of-run rules were found that are optimal, i.e., will always schedule jobs to give the minimum possible makespan. When the GP parameters were changed 11 out of 100 best-of-run rules were found to be optimal. This validates both the GPHH approach for generating dispatching rules for the JSS problem, and the representation used. Further, including the additional terminal `NPR` in the feature set was shown to improve the results, increasing the number of rules found which will always give the minimum possible makespan to 12 out of 100. This has highlighted that even in the simplest JSS environment, the selection of properties from the jobs has a large impact on how well the GP system works, and therefore as the JSS environments studied get increasingly more complex, with a larger number of machines, the selection of appropriate features is likely to have an even more significant impact on the results that GP is able to attain.

Static JSS problems are simpler than their dynamic counterparts, and most real-world scheduling environments are dynamic. These initial experiments are very different to the later experiments. Here we *know* that an optimal solution exists and that it can be represented as a DR. Hence we know what terminal and function sets are sufficient and what tree depth is sufficient. In later experiments with dynamic job shop and total weighted tardiness objective, there is no optimal solution, so the tree depth and terminals required to represent the best possible solution are not known. Here in this situation we can explore the differences to search efficiency by increasing the maximum tree depth beyond what we know is sufficient to find an optimal solution, and to include additional terminals and functions which are not necessary to find the optimal solution. This makes the search space larger, so GP might find it harder to find effective solutions — because GP is a heuristic, it is not guaranteed to find the best possible DRs and fitness evaluation is only an estimate through a discrete-event simulation. With the validation of the genetic programming based approach for generating dispatching rules proven in the simpler static environment, the following chapters will focus on more complex dynamic JSS problems.

This chapter has also investigated the combined effects of changing the problem instances throughout evolution, and using one scheduling rule versus machine specific scheduling rules in non-symmetric dynamic two-machine job shops. Results show that R1 methods do not consistently outperform R2 methods in terms of TWT, neither do the C methods outperform the S methods. However, the mean performance of R2 methods do have a smaller standard deviation than their R1 counterparts. As we cannot separate which methods are best, we cannot assume that one rule for all machines is sufficient on all possible configurations. When we consider run times, the R2 methods have shorter mean evolution times with smaller standard deviations than their R1 counterparts. C methods have longer mean evolution times but smaller standard deviations than the S methods. Further the R2 methods have shorter mean testing times with smaller standard deviations than their R1 counterparts and C methods have shorter mean evolution times and smaller standard deviations than the S methods.

The two-machine dynamic job shop is the simplest of dynamic job shop environments. Following from this initial experimentation of dynamic job shop scheduling, the following chapters will move on to use the ten-machine dynamic job shop. This is a more realistic size of job shop for manufacturing environments, and results obtained in this larger job shop are expected to generalise better across different sized job shops than rules developed in the two-machine job shop. Other measures of the current state of job queues will also begin to be incorporated.

The idea of evolving GP individuals consisting of multiple dispatching rules for use in different situations may be of more use in larger job shops; however, the difficulty lies in determining when each dispatching rule is to be used. There are many different properties which could be used to differentiate between machine states, and being able to dynamically determine which dispatching rule to use is a difficult challenge.

The next chapter begins by investigating how GP can be used to develop dispatching rules which are “less-myopic”. This is increasingly important as the size of the job shop increases, and each job has the potential for more operations.

Chapter 4

Automatic Discovery of "Less-Myopic" Dispatching Rules

4.1 Introduction

A major disadvantage of dispatching rules is their lack of a global perspective of the current and potential future state of the shop [17]. The majority of current research into JSS lies at two extremes: a global approach through optimisation methods where an entire schedule is constructed and, at the other extreme, local approaches using dispatching rules.

An approach to developing "less-myopic" dispatching rules could focus on several different dimensions: it could be "less-myopic in space", taking into account the current state of more than just the queue of the machine in question or it could be "less-myopic in time", taking into account more than just the current state of the job, e.g., including an approximation of the priority value that would be assigned in future at the next machine the job visits for its next operation. Based on the conclusions of Chapter 3 we will now always use the *same* DR at every machine.

4.1.1 Chapter Goals

The purpose of this chapter is to investigate how to develop less-myopic, i.e., less shortsighted, DRs for scheduling in dynamic JSS environments. Two methods of developing less-myopic DRs will be investigated. The first is to explore whether modifications to the feature set enable the evolution of less-myopic DRs. The objectives of this method are:

1. Determine which possible attributes of jobs, machines and the shop system as a whole are useful to improve DR performance.

2. If performance is improved, to examine which aspects improved with the inclusion of less-myopic terminals.

The second method is to include local search as an additional means of providing feedback on how fit a given dispatching rule is throughout evolution. The specific objectives of this method are:

1. Investigate whether the inclusion of local search to provide additional feedback on a DR's performance leads to more effective DRs.

2. Investigate which simple local search operator provides the best performance.

3. Determine whether the increase in computational time is a worthwhile trade-off for any improvement in performance.

4. Collect data on the state of queues in the job shops and investigate what the best use computational time is, e.g., whether it is more important to schedule well when queues are long or when they are short.

5. Determine whether a local search based fitness feedback can improve tie breaking.

4.1.2 Chapter Organisation

The remainder of this chapter is organised as follows. Section 4.2 presents the approach using a larger feature set including properties from the wider shop. Section 4.3 presents the approach using local search to provide additional feedback on the fitness of a dispatching rule. Section 4.4 uses the local search based feedback approach to investigate tie breaking. Section 4.5 summarises the key findings of the chapter and provides directions for future work.

4.2 Less-Myopic Feature Sets

The challenge being addressed is to find "local, decentralised rules which result in a good global performance of a given complex manufacturing system" [17]. This section considers using a genetic programming based approach to automatically discovering less-myopic scheduling rules. In a ten-machine job shop, features from the wider shop system will be incorporated to investigate whether the rules evolved that are less-myopic give improved performance. The aim is to find robust DRs which perform well in job shops where jobs have different properties so we will compare training with problem instances where all jobs have four operations, and training with problem instances where all jobs have eight operations to see whether the size of jobs affects their generalisation.

4.2.1 Terminal and Function Sets

We will consider GP using two different feature sets. All terminals used are shown in Table 4.1. The first, "normal GP" (NGP) consists of the terminals listed under job properties and machine properties only in Table 4.1. The second, "less-myopic GP" (LMGP), uses a feature set consisting of all the features in Table 4.1. `RJ` is the release time of the job, or the time its last completed operation finished. The wider looking properties, `NPR`,

Table 4.1: Terminal set used in GP system for dynamic ten-machine job shop.

Feature	Symbol
Job Properties	
Processing time of operation	`PR`
Remaining processing time of job	`RT`
Remaining number of operations	`RO`
Ready time of job's current operation	`RJ`
Due date of job	`DD`
Weight of job	`W`
Machine Properties	
Ready time of machine	`RM`
Number of jobs in queue	`NQ`
Average wait time of last five jobs processed	`QW`
Current time	`CT`
Wider Looking Properties	
Next operation's processing time	`NPR`
Number of jobs in queue at the next machine job visits	`NNQ`
Average wait time of last five jobs processed at the next machine job visits	`NQW`
Average wait time of last five jobs processed across all machines in the shop	`AQW`

`NNQ` and `NQW` all return zero if the job does not have an operation after the current operation. If fewer than five jobs have visited a machine, then `NQW`, and `AQW` return the average wait time of the jobs which have visited. By including `NNQ` and `NQW`, the DR has access to the same information about the *next* machine the job visits (if there is one) as it does for the machine it is currently at By including `AQW`, a rule is able to approximate the total time a job has remaining in the system. There are many possible attributes of the next machine that jobs visit which could be used as terminals in our less-myopic approach. The terminals NNQ and NQW give the rules the same machine knowledge about the next machine as about the current machine. There are many possible terminals that could be used to estimate how busy a machine is, e.g., work in the current queue (WIQ) and work in the next queue (WINQ), but this is a question for future research. The value of the WIQ can be approximated by existing terminals in the terminal set, using NQ$*$PR. The potential benefit of including the less myopic terminals is to allow the possibility of a DR which can dispatch a job to begin processing which will go on to visit a machine which is currently idle rather than dispatching a job which will go on to a machine which is processing a job already or has a large queue of waiting jobs. These values could also be used to determine which of two similar jobs has, on average, a larger expected waiting time at the next machine and therefore may need to have a higher priority to be processed by (or close to) its due date. The terminal AQW gives the DRs the ability to better estimate the time until the job finishes, and compare this with its due date. This should be a better approximation than using the average waiting time at the current machine.

The function set is $\{+$, $-$, $*$, $\%$, if>0, max, min$\}$, following Chapter 3. The arithmetic operators take two arguments. The first three arithmetic operators, $+$, $-$ and $*$, have their usual meanings. The $\%$ operator is protected division, returning zero if dividing by zero. The if>0 function takes three arguments; if the first argument is greater than 0 then it returns the second, else the third is returned. The max and min functions take two arguments and return the maximum and minimum of their arguments respectively.

4.2.2 GP System Setup

Fitness Evaluation

We implement a heuristic generation approach with a GP system using ECJ20 [78] for evolving DRs. This is using GP as a hyper-heuristic (see Section 2.1.2 on page 25). The fitness of a DR (individual) in the current GP population is evaluated using discrete-event simulations of problem scenarios of a job shop. The objective of interest to be minimised, TWT, is calculated for each problem instance, and the average over all problem instances used is the fitness of the DR. The best DR evolved is then tested on independent test problem scenarios.

Problem Instances

We randomly create problem instances with ten machines. The processing times for machines follow a discrete uniform distribution with mean μ. In each problem instance, jobs arrive stochastically according to a Poisson process with rate λ. Equation (4.1) is used to set λ so that the machines have a desired expected utilisation (the proportion of time a machine is busy), ρ,

$$\lambda = \frac{\rho}{(\mu \times p_M)}, \tag{4.1}$$

where p_M is the proportion of jobs that need to be processed at machine M. For example, in a ten-machine job shop, if each job has two operations, $\lambda = \rho/(\mu \times (2/10))$. Due dates are set using Equation (4.2) [5],

$$d_j = r_j + h \times \sum_{l=1}^{N_j} p(\sigma_{j,l}), \tag{4.2}$$

where h is a due date tightness parameter, randomly chosen from the choices available for each job. Jobs are given weight 1, 2 or 4, with probability $(0.2, 0.6, 0.2)$ [111]. This is based on the premise of the 20/60/20 rule, that is 20% of jobs are of low importance, 60% of jobs are of average importance, and 20% of jobs are of high importance. A warm up period

Table 4.2: Configurations for training and testing scenarios (mean of processing time distribution, expected utilisation and due date tightness) for dynamic ten-machine job shop problem instances.

		μ	ρ	h
Training	T1	25	0.85	{3,5,7}
	T2	25	0.95	{3,5,7}
Testing	P1	25	0.90	{2,4,6}
	P2	50	0.90	{2,4,6}
	P3	25	0.97	{2,4,6}
	P4	50	0.97	{2,4,6}

of 500 jobs is used, and we collect data from the next 2000 jobs to arrive ($N = 2000$), however new jobs keep arriving in the system until the 2500th job is completed. The warm up period of 500 jobs has been shown to be sufficient for the shop to reach steady state [52]. The two configurations used for training, and the four used for testing are shown in Table 4.2. Only one problem instance is used from each scenario for both training and testing. This is because each instance requires lots of scheduling decisions to be made. In training we use either four operations in every job (4op training), or eight operations in every job (8op training). Two training scenarios are used for evaluation at each generation of evolution, T1 and T2. The average TWT from these is used as the fitness for the individual.

In testing we also vary the number of operations in each job, using each configuration for a scenario where the number of operations is fixed at 4, 6, 8 or 10, and where the number of operations in each job varies randomly between 2 and 10 (with equal probability). Using testing scenarios P1 to P4 with each of these differing number of operations per job gives 20 test scenarios. In this chapter we are again interested in minimising the total weighted tardiness,

$$TWT = \sum w_j T_j. \tag{4.3}$$

GP System Parameters

The initial population is generated using the ramped-half-and-half method [70] with minimum depth of two and maximum depth of six, and with function and terminal sets as described above. The population size is 1024 and evolution is for 50 generations. This population size has been previously used [83, 95] as has the number of generations [95, 108]. GP trees have a maximum depth of eight. For the genetic operators crossover, mutation and elitism, we use rates of 85%, 10% and 5% respectively. Tournament selection with a tournament size of seven is used to select individuals for genetic operators. This is a common setting that has been previously used [50, 95]. The results of Chapter 3 showed that the larger tournament size had better results than a tournament size of four.

4.2.3 Experimental Results

Here we present the results of using NGP and LMGP. We also compare training in a job shop where every job has four operations (NGP-4op and LMGP-4op), with training in one where every job has eight operations (NGP-8op and LMGP-8op). There are many ways that the amount of information that is used in training DRs can be increased. One way of doing this is increasing the number of jobs used in the simulation. DRs which are trained on the training instances which use four operations per job are performing only half the scheduling decisions that the DRs trained on the training instances with eight operations per job. This is one way of investigating the gain that increasing the number of scheduling decisions are made in training leads to in final performance. For each we performed 100 GP evolutionary replications, using 100 common psuedo-random number generator seeds.

Performance on Test Scenarios

Figure 4.1 shows the results on the 20 test scenarios by the 100 best-of-run evolved DRs from both NGP and LMGP, trained with four and eight operations in every job respectively. Each graph presents the results for one test scenario, with a violin plot showing the distribution of TWT attained for each method; additionally the individual TWT values attained are overlaid in black. Table 4.3 gives the corresponding mean and standard deviation of TWT.

Figure 4.1 shows that on the P1 and P2 test scenarios, with the lower expected utilisation, NGP evolves better performing DRs, as there are a greater number of DRs with low values of TWT. However we performed Mann-Whitney U tests, which showed that the difference in means between NGP and LMGP is not statistically significant at the 5% level.

On the other hand, if we consider the P3 and P4 test scenarios with higher expected utilisation (except for 2-10ops P3), LMGP now evolves more DRs that attain low TWT values, and fewer DRs with very high TWT. The difference in means is statistically significant at the 5% significance level. These have a higher utilisation rate than P1 and P2, so it seems logical that it would be more important to take into consideration shop features that look further than the current machine, as there are likely to be longer queues at machines. From Table 4.3 we observe that on these test scenarios the standard deviation is always smaller using LMGP. Further LMGP-8op has smaller standard deviation than LMGP-4op on all but two problem instances.

The methods with four operations in every job through evolution, NGP-4op and LMGP-4op, outperform their eight operation counterparts, NGP-8op and LMGP-8op respectively, on all the P1 and P2 scenarios as well as P3 and P4 2–10ops scenarios. On scenarios P3 and P4 (except 2–10ops) there is not a statistically significant difference in performance between NGP-4op and NGP-8op, or between LMGP-4op and LMGP-8op. In training the 4-op methods are expected to see fewer operations than 8-op meth-

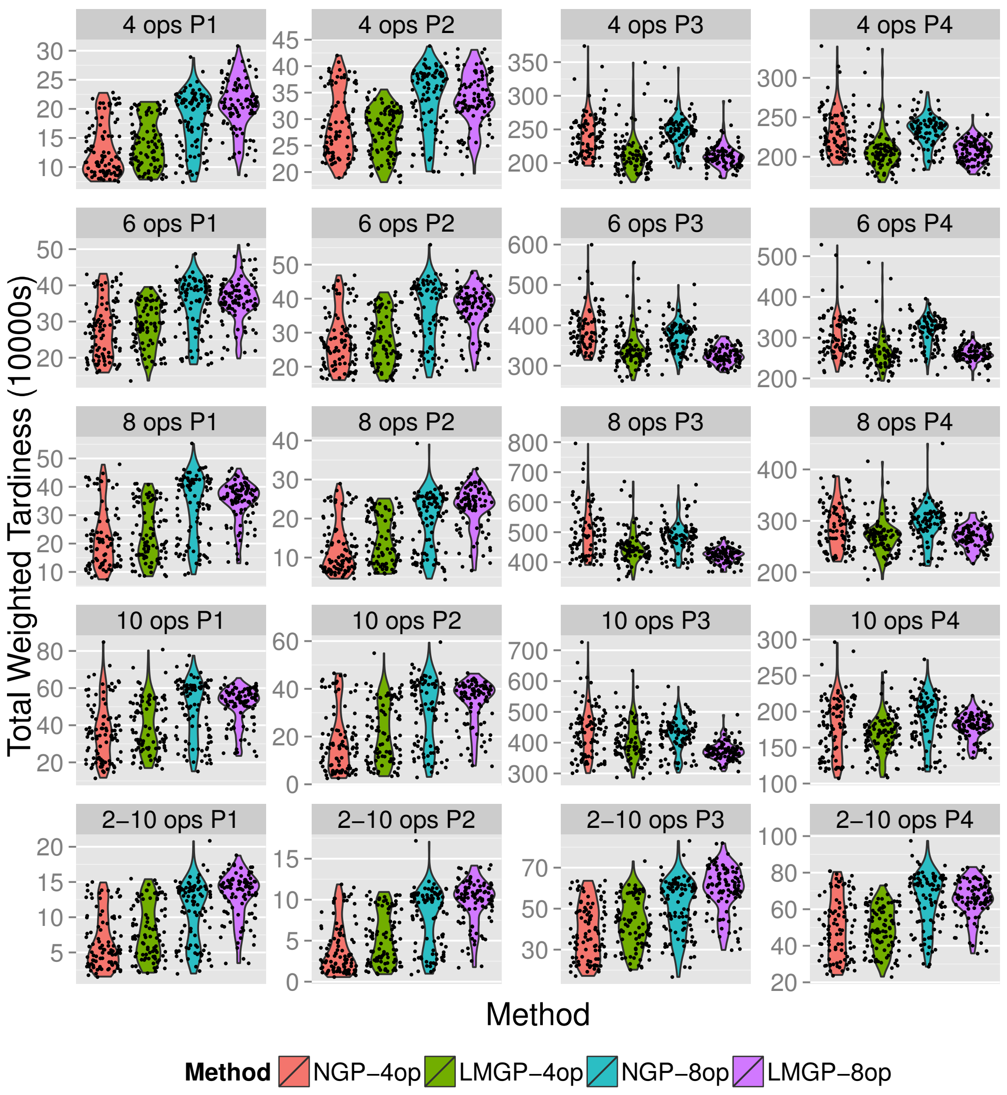


Figure 4.1: Violin plots presenting results (TWT) of test scenarios for NGP-4op, LMGP-4op, NGP-8op and LMGP-8op on ten-machine dynamic job shop. Note that the vertical axis scales are different in each subfigure.

ods, yet this does not seem to hinder the performance of the evolved DRs, but may be one reason for the smaller standard deviation in TWT performance of LMGP-8op compared to LMGP-4op. Also note that the best (lowest) values for the worst performance (highest TWT) are achieved by either LMGP-4op (7/20) or LMGP-8op (9/20) on the majority of the test scenarios.

Table 4.4 shows the average of both the mean and standard deviation of the number of jobs waiting in machine queues for each of the 20 scenarios and methods. LMGP-8op gives the lowest mean value of both mean and standard deviation on almost all of the scenarios. LMGP-4op also improves on the values of NGP-4op and NGP-8op although it is not as good as LMGP-8op. We see through these results that the improvements offered by using LMGP can be partly attributed to reducing the expected number of jobs in queues, and evening out the queue lengths at the different machines in the shop. The results of 2-10ops show a trend of increasing TWT in the order NGP-4op, LMGP-4op, NGP-8op, LMGP-8op. DRs evolved by LMGP-4op achieved the lowest TWT of the runs from the four methods on 11 of the 20 test scenarios, rules evolved by NGP-4op achieved the lowest TWT on seven of the test scenarios, and NGP-8op rules attained the lowest TWT on the remaining two test scenarios.

Table 4.5 gives the mean and standard deviation of the evolution time, in minutes, and the total testing time (for all twenty test scenarios), in milliseconds. The evolution time is lowest for LMGP-4op, followed by NGP-4op and LMGP-8op. The LMGP methods have smaller standard deviations. NGP-8op has the longest average evolution time and the largest standard deviation. Table 4.5 shows that the time taken for testing is less with LMGP-4op than NGP-4op and less with LMGP-8op than NGP-8op, and that the standard deviation is smaller for LMGP methods. LMGP-8op has the quickest mean test time of approximately five seconds.

Table 4.3: Mean and standard deviation of total weighted tardiness for ten-machine job shop.

	NGP-4op		LMGP-4op		NGP-8op		LMGP-8op	
	Mean	Stdev	Mean	Stdev	Mean	Stdev	Mean	Stdev
4op P1	130012.50	45168.51	139687.21	40866.23	179694.27	47801.36	206913.79	44592.13
4op P2	288731.45	60867.46	274289.57	43943.78	336878.11	58758.37	337541.71	48990.07
4op P3	2416134.38	320656.86	2127786.24	290851.86	2457454.99	232999.15	2103849.95	163740.18
4op P4	2299718.04	281090.74	2076490.43	251747.71	2323739.12	189996.31	2074407.22	141892.91
6op P1	278385.49	78270.10	290148.57	61636.51	340704.03	75679.02	359798.08	58445.20
6op P2	277452.76	81925.00	283216.92	73544.68	354454.73	89850.84	368857.28	64011.67
6op P3	3880850.24	505547.52	3453375.63	486524.22	3720896.20	368124.55	3258580.42	204292.28
6op P4	3027124.88	525046.22	2663811.84	447940.36	3138951.12	364598.26	2598542.76	186085.94
8op P1	220856.04	105440.89	235865.36	97670.34	327149.41	114471.19	345764.55	72574.88
8op P2	124785.09	65911.15	146828.75	62336.99	192425.64	73525.78	228515.34	55301.64
8op P3	4985465.41	799423.44	4489386.19	531549.37	4835644.40	512518.79	4225090.75	229864.51
8op P4	2901411.77	417352.74	2698279.34	339414.06	2929491.13	360604.34	2695113.12	210071.04
10op P1	365720.34	157117.61	390159.65	141723.68	481175.06	156292.44	505239.36	100390.26
10op P2	176482.48	128037.25	215141.90	123886.37	298849.21	139751.64	342768.70	93774.30
10op P3	4342695.89	809956.98	4074597.13	637837.34	4237180.37	540592.44	3723260.75	287245.04
10op P4	1804118.93	403840.85	1686780.92	260363.49	1905379.46	334249.28	1793891.66	182499.29
2-10op P1	64913.32	37814.58	79528.10	39059.80	106499.75	42883.45	130122.19	33225.58
2-10op P2	40063.65	31117.71	52725.28	31207.22	76804.04	36061.47	96762.79	27640.52
2-10op P3	388413.32	132586.12	432380.14	120921.98	508117.19	132659.56	593566.38	114601.87
2-10op P4	502883.51	164893.81	495076.13	118384.28	644351.87	151670.52	652134.71	105591.85

Table 4.4: Mean of average and mean of standard deviation of queue length for dynamic ten-machine job shop.

	NGP-4op		LMGP-4op		NGP-8op		LMGP-8op	
	Mean	Stdev	Mean	Stdev	Mean	Stdev	Mean	Stdev
4op P1	2.55	0.30	2.43	0.29	2.51	0.28	2.35	0.29
4op P2	3.49	1.08	3.17	1.03	3.44	1.00	3.04	0.97
4op P3	6.38	2.16	5.47	1.66	6.30	2.21	5.23	1.63
4op P4	6.67	2.51	5.81	2.25	6.55	2.52	5.58	2.03
6op P1	3.04	0.50	2.86	0.52	2.97	0.48	2.71	0.47
6op P2	2.84	0.57	2.67	0.50	2.76	0.54	2.54	0.46
6op P3	7.55	2.89	6.54	2.56	7.33	3.02	6.19	2.34
6op P4	6.84	1.77	5.94	1.47	6.74	1.74	5.67	1.38
8op P1	2.55	0.36	2.42	0.35	2.50	0.39	2.32	0.41
8op P2	2.09	0.24	2.02	0.22	2.04	0.22	1.94	0.22
8op P3	6.85	2.02	6.05	1.82	6.64	1.92	5.65	1.61
8op P4	5.44	1.64	4.69	1.46	5.20	1.54	4.43	1.29
10op P1	2.78	0.30	2.66	0.30	2.70	0.32	2.52	0.30
10op P2	2.23	0.20	2.15	0.19	2.18	0.22	2.05	0.20
10op P3	5.77	1.67	5.29	1.49	5.52	1.73	4.93	1.35
10op P4	3.75	0.94	3.44	0.87	3.68	0.96	3.36	0.83
2-10op P1	1.75	0.23	1.71	0.23	1.71	0.25	1.64	0.24
2-10op P2	1.56	0.18	1.53	0.18	1.53	0.18	1.48	0.18
2-10op P3	2.71	0.40	2.57	0.39	2.66	0.43	2.48	0.39
2-10op P4	2.80	0.60	2.62	0.55	2.74	0.58	2.51	0.53

Table 4.5: Mean and standard deviation of training and testing times for dynamic ten-machine job shop.

	NGP-4op		LMGP-4op		NGP-8op		LMGP-8op	
	Mean	Stdev	Mean	Stdev	Mean	Stdev	Mean	Stdev
Evolution time (min)	1283.9	826.2	1172.2	694.5	1715.0	826.2	1134.0	734.4
Test time (ms)	4596.6	3040.3	4060.3	2520.2	5957.7	3463.6	3792.7	2800.3

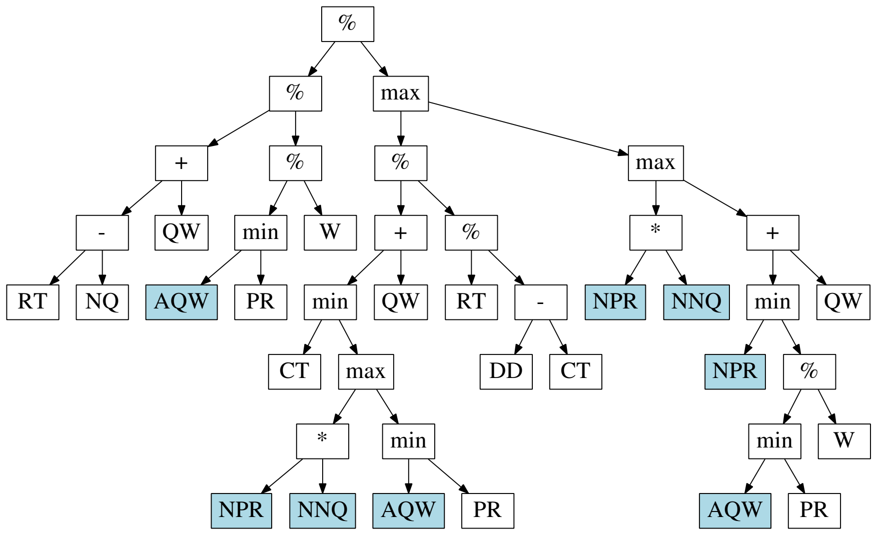


Figure 4.2: One of the best performing LMGP-4op DRs on the test scenarios.

4.2.4 Analysis of Evolved DRs

To analyse why the evolved DRs work well, we choose some of the best performing evolved rules and present them as trees. We have simplified the DRs shown in Figures 4.2, 4.3 and 4.4 slightly, though more simplification is likely possible.

Figure 4.2 shows a dispatching rule evolved using LMGP-4op represented as a tree. The shaded boxes highlight the additional terminals introduced to incorporate features of the wider shop. This rule achieved the minimum TWT on ten of the 20 test scenarios (4 ops P1–P4, 6 ops P1–P2, 8 ops P3–P4 and 2-10 ops P3–P4). A useful fragment appearing in this DR is $\texttt{DD}-\texttt{CT}$ which will be greater than 0 if the job is not overdue, and further this appears in the DR as $\texttt{RT}/(\texttt{DD} - \texttt{CT})$, which is a ratio of the remaining time to the time until due.

In Figure 4.3 one of the best performing rules from LMGP-8op is presented as a tree. It is interesting to note the appearance of the inverse of the WSPT rule in the two places in the rule, both times in the left branch of a min operator. It occurs as $(((\texttt{AQW}\%\texttt{W})\%\texttt{CT}) * \texttt{PR})$. It is interesting to observe that if there are no operations after the current operation this DR will return 0.

In Figure 4.4 we note a similar fragment to that observed in the DR of Figure 4.2, $\texttt{RT}/(\texttt{DD} - \texttt{RM})$. In the DR of Figure 4.4 we also observe the presence of the WSPT rule ($\texttt{W}/\texttt{PR}$) in left-most branch.

It is interesting that all of the DRs presented are smaller than what they could potentially be (maximum depth of eight), hence it may be promising to restrict the depth to seven, or less, and compare the behaviour of the DRs evolved under that constraint.

In the evolved DRs (trees) we frequently see a count (e.g. RO, NQ or NNQ) compared to a length of time (e.g. PR) or an absolute time (e.g. DD). The comparisons between counts and local measures using $+$ and $-$ make little sense as we try to interpret the DR, however comparisons using $*$ and $\%$ are more meaningful.

Partial fragments of the DRs, as identified above, are interpretable, however, as whole DRs these are also difficult to interpret and understand why they are, or are not, effective at dispatching jobs.

4.2.5 Feature Set Conclusions

The goal of this section was to develop a GP based approach to evolve less-myopic scheduling rules for the dynamic ten-machine job shop environment. To achieve this we introduced four additional terminals to the

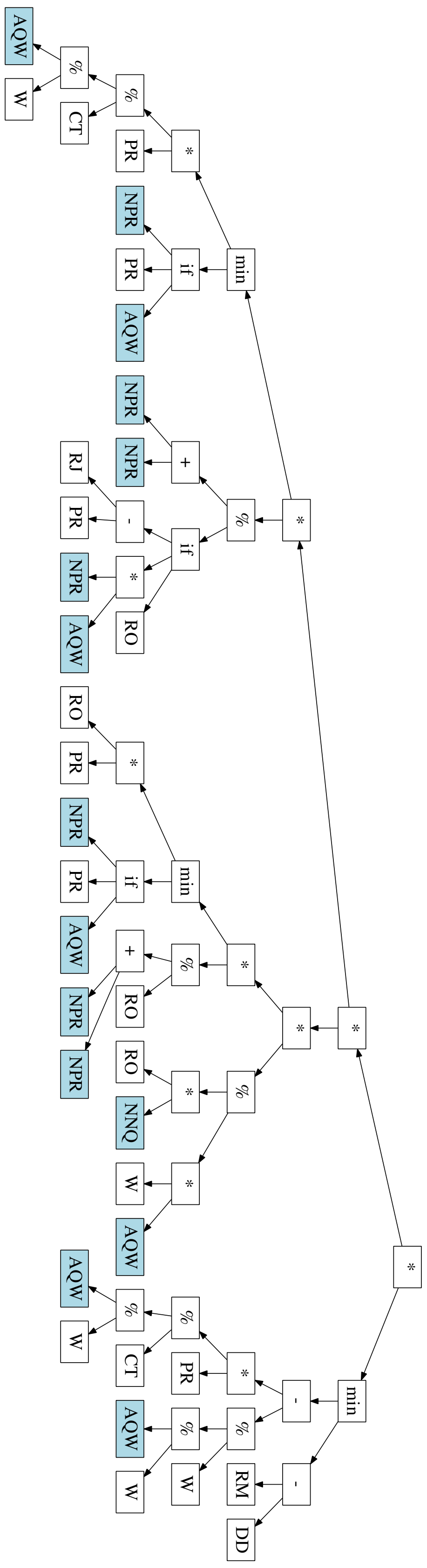


Figure 4.3: One of the best performing LMGP-8op DRs on the test scenarios.

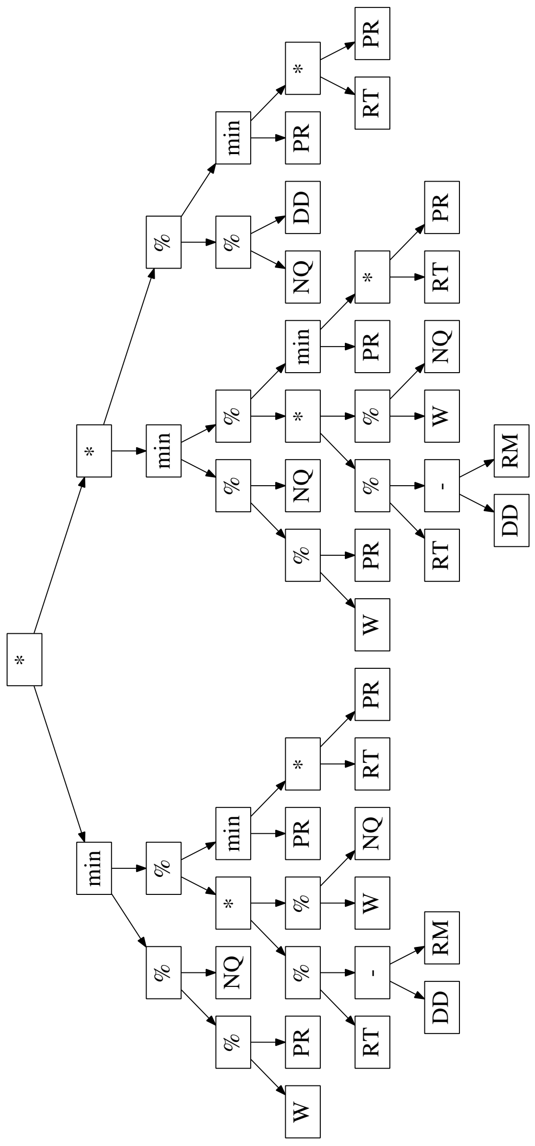


Figure 4.4: One of the best performing NGP-8op DRs on the test scenarios.

terminal set, which captured more information about the current state of the job shop beyond the current machine.

Results have shown that the inclusion of these less-myopic terminals led to the evolution of better DRs in terms of total weighted tardiness across the test scenarios with high utilisation. On these scenarios the difference was statistically significant at the 5% significance level, with the LMGP evolved DRs having lower means and smaller standard deviations, as well as shorter evolution and test times. Increasing the terminal set leads to a great increase in the size of the search space of possible GP trees; despite the larger search space, LMGP outperformed NGP.

Across all twenty test scenarios, LMGP-4op and LMGP-8op decreased the average queue length in comparison to NGP-4op and NGP-8op. In terms of mean queue length, LMGP-8op $<$ LMGP-4op $\leq$ NGP-8op $\leq$ NGP-4op. The mean standard deviation of queue length was generally less under LMGP methods than NGP methods. Further the time for testing was less with LMGP-8op than NGP-8op. On 18/20 test scenarios, the four operation methods outperformed their eight operation counterparts.

4.3 Using Local Search to Evaluate Dispatching Rules in Dynamic JSS

Dispatching rules are often short-sighted as they make one instantaneous decision at each decision point. In this section we incorporate local search into the *evaluation* of dispatching rules to investigate including local search to provide additional feedback on the quality of decisions made by dispatching rules. We want to see whether this encourages the dispatching rules to make good local decisions for effective overall performance.

In Section 4.2 we encouraged the evolution of "less-myopic" DRs through the inclusion of properties from the wider shop, and showed that the inclusion of additional "less-myopic" terminals improves the performance of DRs on scenarios with high utilisation, and reduced the expected queue length at machines. This section investigates "less-myopic" in a different way, by including an assessment of how well a DR makes local decisions over an extended decision horizon *in its fitness evaluation*.

Local search algorithms try to improve on an initial solution by searching over a defined neighbourhood of the initial solution [1]. In static job shop scheduling environments, a dispatching rule can be used to create a schedule for processing the jobs, and local search can be applied to improve the schedule. This is *not* how we are using local search in this section. The main goal of this section is to modify the GP based system for automatic generation of dispatching rules from Section 4.2 to incorporate a local search element into the fitness evaluation stage to provide more feedback to DRs on their scheduling performance, so as to encourage a more global perspective in the evolved rules. Here local search is used only to evaluate the fitness of dispatching rules, *not* to change the order in which jobs are dispatched.

The decision horizon is extended to include not only the queue of the current machine but also the queue of jobs at the next machines that jobs in the current queue visit next. When a machine becomes available, a DR builds an initial schedule of jobs at the current machine. This is evaluated over an *extended decision horizon* of the current machine and the next machine of each job. Local search is used to attempt to improve that sequence, and information on the possible improvement contributes to the fitness of the DR.

We want to find DRs that make good decisions, which we can do by looking at the wider local effect when dispatch decisions are made. We investigate whether this can encourage the evolution of DRs which are less-myopic, *better* at scheduling jobs in an order which is *better* beyond just the first job in the queue, and have *better* generalisation performance on unseen problem instances.

In the remainder of this section we will first describe how local search is incorporated into the GP system, followed by the experiment design.

Then we will discuss the results and present conclusions.

4.3.1 Local Search based Evaluation

Here we describe our new method, which uses local search as an additional evaluation of the fitness of a given DR, across an extended decision horizon. We use local search to evaluate potential queue orders, and compare these to the priority sorted queue. We are interested to see if the increase in computational time is a reasonable trade-off for better evolved DRs.

Neighbourhood Search Operators

Local search methods start with an initial solution and try to find better solutions by searching neighbourhoods. The initial solution will be found by using the GP individual (DR) to create a queue ordering through priority assignment. Three simple neighbourhood search operators will be used.

SwapFront swaps the job at the front with each other job in the queue.

MoveFront inserts each job at the front of the queue.

Transpose swaps two adjacent jobs in the queue.

These operators have been chosen as they are simple, and have small neighbourhood size, $(n-1)$ for n jobs in the machine queue, and therefore low computational cost. These three operators are illustrated in Figure 4.5 with an example queue of four jobs, J1–J4.

Evaluation Process

At each scheduling decision point, neighbourhoods will be compared by calculating the *expected* contribution to total weighted tardiness (TWT) of

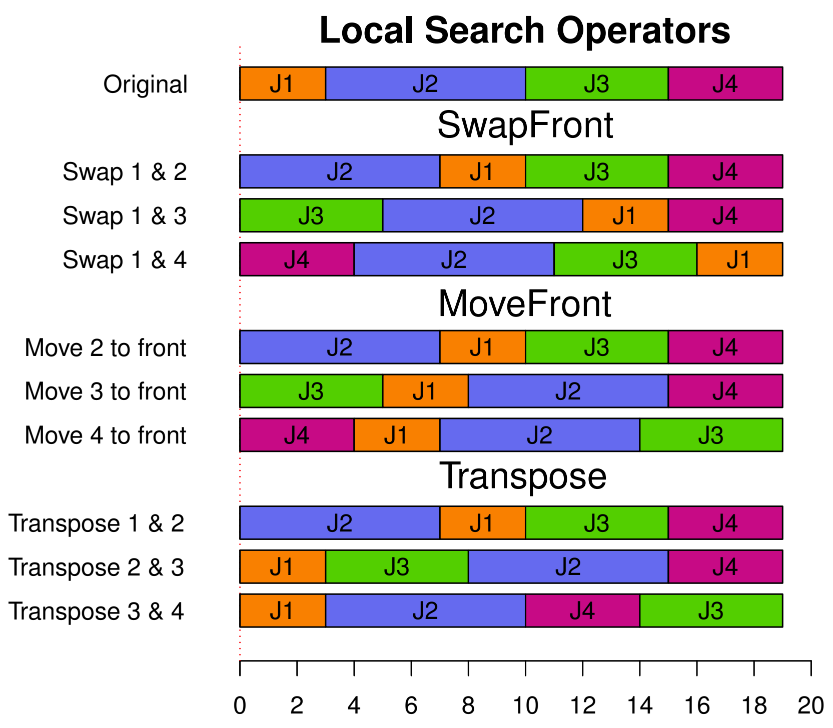


Figure 4.5: Diagram of evaluation process when a machine becomes available to process a job.

the jobs currently queued at the given machine. This is calculated by taking the expected completion time of each job, given its current position in the queue (at the current machine) and where it would fit in the queue of jobs at the next machine the job visits (assuming the current state of the job shop as static, with no new arrivals except those moving from the current queue).

For each job j in the queue, this is calculated as

$$E(C_j) = \mathtt{CT} + QC_j + \mathtt{PR}_j + QN_j + \mathtt{NPR}_j + QR_j + (\mathtt{RT}_j - \mathtt{PR}_j - \mathtt{NPR}_j) \quad (4.4)$$

For a given job d_j, CT, PR, NPR, RT are constants as in Table 4.1. QC_j is the time remaining waiting (queuing) of job j at the current machine (M_C), i.e., the sum of the processing times of jobs ahead of job j in the queue under the current ordering of jobs. QN_j is the time spent waiting (queuing) of job j at the next machine on the job's route, (M_N). This is determined by treating the next machine's queue as a one machine problem with arrivals only from the current machine and using the DR to dispatch jobs until all jobs that join this machine from the current machine have been dispatched. This gives us a lower bound on the length of time each job from machine M_C that next visits M_N is expected to be in the queue at M_N, as in the full simulation jobs may arrive into the shop and from the other machines. QR_j is the sum of the average expected waiting times at each remaining machine on job j's route after machines M_C and M_N. If a job does not have operations remaining after the current or next machine then QN_j and QR_j will have value 0. Further we are only interested in changing the order of jobs at the first machine, and calculating the predicted time in the next queue, therefore for each job QR_j is also a constant. As our objective is to minimise the total weighted tardiness, we are seeking to minimise Equation (4.5) across the neighbourhood being searched.

$$Total = \sum_{j=1}^{J} w_j \times \max\{0, E(C_j) - d_j\}. \tag{4.5}$$

Each time the DR is applied to select the next job, the expected contribution of the original queue order, $Total_0$, is calculated. If $Total_0 = 0$ then we cannot improve on the current queue ordering, so we do not apply local search. For each neighbouring solution, $Total$ is calculated, and the minimum $Total$ across all neighbours is $Total_{min}$. If at least one of the new job queue orders has an improved (smaller) expected contribution to TWT than the original, we calculate the difference $penalty = Total_0 - Total_{min}$, i.e.,

$$penalty = \max\{0, Total_0 - Total_{min}\} \tag{4.6}$$

so if no queue order improves the expected contribution to TWT, then there is no penalty. We sum all the penalties incurred during the discrete event simulation, and average over the number of times the local search was applied; this gives $penalty_{mean}$.

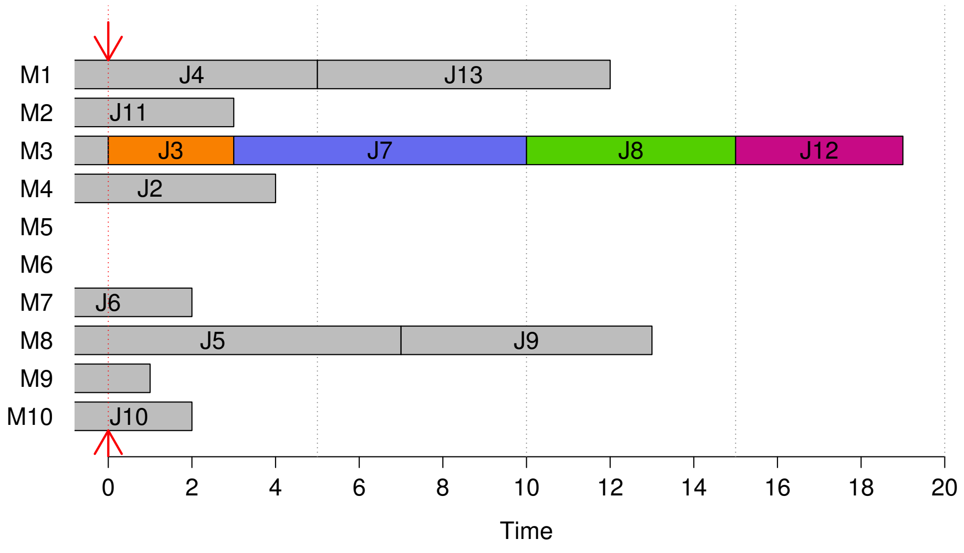


Figure 4.6: Example showing a full ten-machine job shop at a particular scheduling point (time$= 0$).

Example of Local Search Based Queue Evaluation

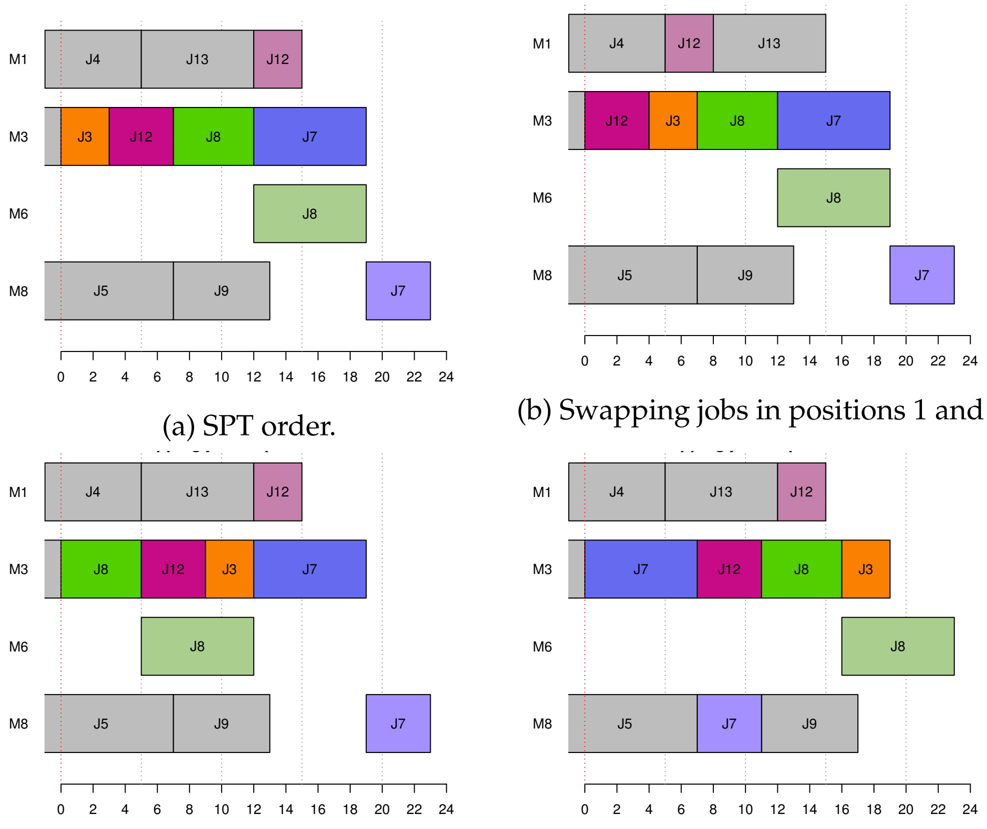
Consider the decision point shown in Figure 4.6. A job completes at machine M3, and there are four jobs currently in the queue (arranged in FCFS order for arguments sake). The four jobs waiting at M3 and their relevant properties are shown in Table 4.6. In this example, we are interested in the queue of jobs at machine M3, and the queues at other machines are assumed to not have any more arrivals from outside the system or machines that are not M3.

First, the DR is applied to order the jobs. Figure 4.7(a) shows what happen when the DR is the SPT rule. The next operation of jobs J7, J8 and J12 are shown in lighter colour, where they would be scheduled (by the SPT rule) after the completion of their current operation and joining the next machine queue. Job J3 has no further operations.

Job J3 has the highest priority, is completed at time 3 then exits the

Table 4.6: Properties of queued jobs at machine M3 in the example of Figure 4.6.

Job	`PR`	`NPR`	`(RT−PR−NPR)`	`W`	`DD`	QR
J3	3	0	0	2	10	0
J7	7	4	3	4	25	6
J8	5	7	0	2	20	0
J12	4	3	11	1	30	5



(a) SPT order.

(b) Swapping jobs in positions 1 and 2.

(c) Swapping jobs in positions 1 and 3.

(d) Swapping jobs in positions 1 and 4.

Figure 4.7: Example showing next machines visited by jobs at machine M3 under different ordering of jobs at machine M3.

shop. Job J12 is next up, and upon completion at time 12 joins the queue at M1. The queue at M1 is treated as a single machine problem from time 0, and SPT is used to dispatch jobs. J12 therefore starts processing immediately at time 12 just as J13 finishes. Continuing in this manner, the expected completion times are calculated using Equation (4.4) for the four jobs given this current order are:

$$\begin{aligned}
E(C_3) &= 0 & +0+3 && +0+0 && +0+0 && = \ \ 3\\
E(C_{12}) &= 0 & +3+4 && +5+3 && +5+11 && = 31\\
E(C_8) &= 0 & +7+5 && +0+7 && +0+0 && = 19\\
E(C_7) &= 0 & +12+7 && +0+4 && +6+3 && = 32
\end{aligned}$$

This gives expected contribution to TWT, calculated using Equation (4.5), of

$$\begin{aligned}
Total_0 &= \sum_{j\in\{3,12,8,7\}} w_j \times \max\{0, E(C_j) - d_j\}\\
&= 2\times\max\{0, 3-10\} + 1\times\max\{0, 31-30\}\\
&+ 2\times\max\{0, 19-20\} + 4\times\max\{0, 32-25\}\\
&= 29
\end{aligned}$$

This is the expected contribution of the original queue order, $Total_0$. Since $Total_0 > 0$, there may be another queue ordering which improves on this expected contribution to TWT.

With the local search operator *SwapFront* there are three neighbouring solutions to evaluate. The first (see Figure 4.7(b)) is swapping the jobs in positions 1 and 2, J3 and J12, giving order J12, J3, J8, J7. J12 is completed on M3 at time 4, and joins the queue at M1. When M1 finishes processing J4, the DR is used to choose between J12, which has arrived from M3, and J13, which was already waiting in the queue at M1 at time 0. SPT chooses

J12 ahead of J4. As we follow through the simulation, we get

$$
\begin{aligned}
E(C_{12}) &= 0 && +0+4 && +1+3 && +5+11 && = 24\\
E(C_{3}) &= 0 && +4+3 && +0+0 && +0+0 && = \ \ 7\\
E(C_{8}) &= 0 && +7+5 && +0+7 && +0+0 && = 19\\
E(C_{7}) &= 0 && +12+7 && +0+4 && +6+3 && = 32
\end{aligned}
$$

$$
\begin{aligned}
Total_{1,2} &= 1 \times \max\{0, 24-30\} + 2 \times \max\{0, 7-10\}\\
&+ 2 \times \max\{0, 19-20\} + 4 \times \max\{0, 32-25\}\\
&= 28
\end{aligned}
$$

This order improves on the expected contribution to TWT.

Swapping jobs in positions 1 and 3 (see Figure 4.7(c)) gives:

$$
\begin{aligned}
Total_{1,3} &= 2 \times \max\{0, 12-20\} + 1 \times \max\{0, 31-30\}\\
&+ 2 \times \max\{0, 12-10\} + 4 \times \max\{0, 32-25\}\\
&= 36
\end{aligned}
$$

which does not improve on $Total_0$.

Swapping jobs in positions 1 and 4 (see Figure 4.7(d)) gives

$$
\begin{aligned}
Total_{1,4} &= 4 \times \max\{0, 20-25\} + 1 \times \max\{0, 31-30\}\\
&+ 2 \times \max\{0, 23-20\} + 2 \times \max\{0, 19-10\}\\
&= 25
\end{aligned}
$$

Swapping queued jobs in positions 1 and 2, or positions 1 and 4 both improve on $Total_0$. The minimum attained is $Total_{min} = 25$. Therefore $penalty = 29 - 25 = 4$. This process is repeated whenever a machine becomes available to process a job and there is a queue of waiting jobs.

4.3.2 Initial Experiment

Due to the computational cost of applying local search at *every* decision point throughout evolution, we begin with a small scale experiment to compare two methods using local search to evaluate the fitness of DRs over the extended decision horizon, and only use local search at every 20th decision point. Initially we use only the SwapFront neighbourhood search operator.

Methods and Fitness. The fitness of a DR (individual) in the current GP population is evaluated using discrete-event simulations of problem instances of a job shop. In each method, the fitness values, TWT and penalty, are calculated on four training problem instances, and the mean of the fitness values across these problem instances used is the fitness of the DR.

Benchmark GP (BM). In the benchmark GP method, the objective of interest to be minimised, TWT, is normalised by the expected utilisation and this is used as the fitness value.

Local Search Single Objective (LS-SO). We use local search to evaluate the job queue every 20th time the DR is called to select a job for dispatch. The overall fitness of a DR for a problem instance is the sum of TWT and the penalty, $TWT + penalty_{mean}$. TWT and penalty are on the same scale of weighted time.

Local Search Multi Objective (LS-MO). We use local search to evaluate the job queue every 20th time the DR is called to select a job for dispatch. This is a multi-objective method based on LS-SO, using the NSGA-II algorithm [32] (see Section 2.3.7, page 51). The penalty is used as a distinct second objective. The first objective is TWT and the second objective is $penalty_{mean}$.

GP System Setup. GP individuals are DRs. The properties of jobs, machines and the job shop that are used as terminals of the GP system is the LMGP set of terminals, using all terminals given in Table 4.1 (see page 108). The function set is also the same set used earlier: {$+$, $-$, $\times$, $\%$, `if>0`, `max`, `min`}. The initial population is generated using the ramped-half-and-half method [70] with an initial maximum depth of six. The population size is 50 to constrain the computational time due to the incorporation of local search, and evolution is for 50 generations, a standard setting. GP trees have a maximum depth of six. Genetic operators crossover, mutation and elitism, use rates of 85%, 10% and 5% respectively. Tournament selection with a tournament size of seven is used to select individuals for genetic operators. This is a common setting that has been previously used [95].

Training and Testing. The configuration of training and testing scenarios is given in Table 4.7. Four training scenarios are used. The processing times at each machine follow a discrete uniform distribution with mean μ, i.e., $U(1, 2\mu - 1)$. A warm up period of 100 jobs is used, and we collect data from the next 200 jobs to arrive ($N = 200$), however new jobs keep arriving in the system until the 300th job is completed. This is a very low number of jobs due to the increase in computational time required by local search. Jobs are given weight 1, 2 or 4, with probability $(0.2, 0.6, 0.2)$ [111]. The four scenarios give two utilisation levels, 90% and 95%, and either "full" (one operation at each machine, giving ten operations per job) or "variable" operations (the number of operations per job is uniformly distributed on $\{2, \ldots, 10\}$). In testing, we increase the warm up period to 500 jobs and the number of jobs we collect data to $N = 1000$. In extreme testing, XT1, the processing times follow a geometric distribution with mean $\mu = 25$ (parameter $p = 0.04$), and utilisation is 0.95. We also change the weights given to jobs, including an additional weight for *very* important jobs; jobs are now given weight 1, 2, 4 or 8, with probability $(0.2, 0.5, 0.2, 0.1)$. Due date tightness is equally likely from {2, 2.5, 3}; these are the same or tighter than in training. The aim of the extreme testing instance is to see how well evolved DRs generalise to a problem scenario which is tougher (tighter due dates) and different (processing time distribution) to what they were trained on.

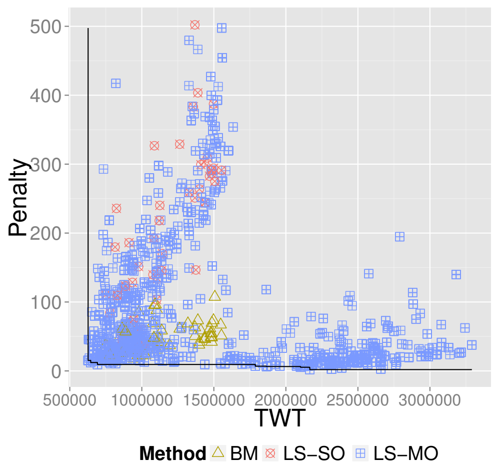
Table 4.7: Configuration of training, testing and extreme testing scenarios for local search based evaluation experiments (mean of processing time distribution, expected utilisation, due date tightness and operation setting).

		μ	ρ	h	Operations
Training	TR1	25	0.90	$\{2, 3, 4\}$	full
	TR2	25	0.95	$\{2, 3, 4\}$	full
	TR3	25	0.90	$\{2, 3, 4\}$	variable
	TR4	25	0.95	$\{2, 3, 4\}$	variable
Testing	T1	25	0.95	4	full
Extreme Testing	XT1	25	0.95	$\{2, 2.5, 3\}$	full

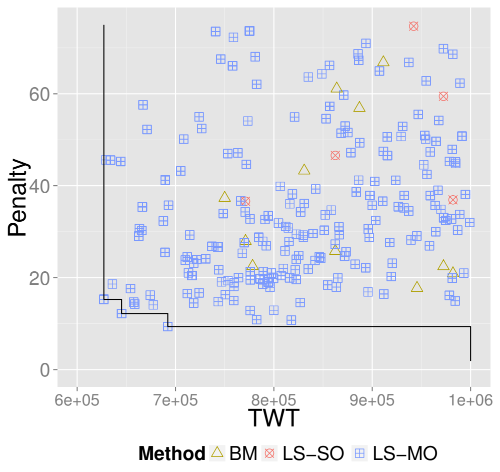
LS-SO and LS-MO Results. We performed 50 independent GP runs for each method, with the same pseudo-random number generator seeds. From these the best-of-run individual (LS-SO), or final non-dominated front (LS-MO) are tested on test and extreme test instances. The aggregate non-dominated front on each instance consists of the non-dominated DRs across *all* evolved DRs from LS-MO.

Figures 4.8 and 4.9 present the test results, TWT and penalty, of the evolved DRs on T1 and XT1. These methods are compared to the benchmark method (BM) which only used the local search penalty calculation during testing. In particular the zoom-ins on the overall best front of DRs in Figures 4.8(b) and 4.9(b) shows that the LS-MO method was able to find a large number of DRs which attain lower TWT and penalty values on the test instance and extreme test instance. All benchmark and LS-SO DRs are outperformed by many LS-MO DRs. The lowest TWT attained by LS-MO and LS-SO methods are lower than the lowest attained by the benchmark. The mean evolution time for BM (316.94±50.81) and LS-SO (308.21±45.76) was less than half that of LS-MO (786.61±100.20).

Including local search in the evaluation process has found DRs which attain good performance. There are a large number of DRs from LS-MO

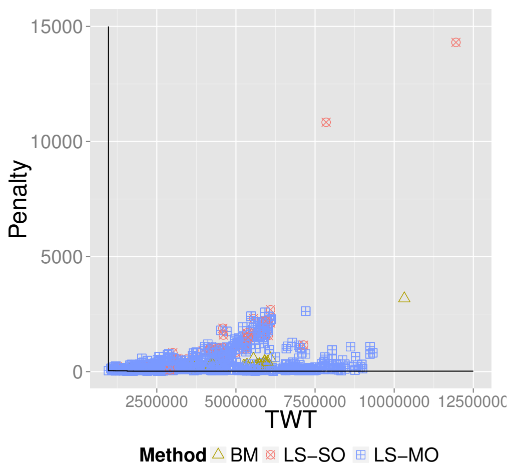


(a) Non-dominated fronts and best-of-run individuals.

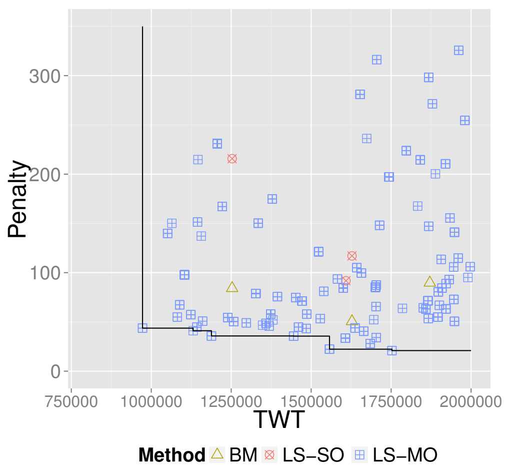


(b) Close up of aggregate non-dominated front.

Figure 4.8: Aggregate non-dominated front on problem instance from T1.



(a) Non-dominated fronts and best-of-run individuals.



(b) Close up of aggregate non-dominated front.

Figure 4.9: Aggregate non-dominated front on problem instance from XT1.

and LS-SO that achieve lower penalty than the benchmark for similar TWT values, particularly for the lower TWT values. This shows that including local search has improved the local performance of DRs.

Due to the computational cost we have investigated small examples, with a small GP population and short problem instances. This has significantly reduced the number of scheduling decisions made in training, and with the shorter warm up period, the system may not have reached steady state.

4.3.3 Extended Experiments

Next we will increase the warm up period, reduce the frequency at which local search is applied, and consider using the MoveFront and Transpose neighbourhood search operators as well as SwapFront. We continue our experiments using only LS-MO in comparison to the benchmark GP method, as on the problem instances in our initial investigation the evolved DRs from LS-MO were shown to outperform the evolved DRs from LS-SO in terms of both TWT and penalty. These experiments use the same method as above, with additional testing instances.

Changes to Problem Instances. We randomly create problem instances of job shops with ten machines. We now use 500 jobs for the warm up period, as this has been shown to be a good number of jobs for the warm up period to allow the job shop to reach steady state [52]. Performance measures from the next 1000 jobs will be collected ($N = 1000$), new jobs keep arriving in the system until the completion of the 1500th job. This is the same for both training and testing.

The same four training scenarios are used. The processing times at each machine follow a discrete uniform distribution with mean μ, i.e., $U(1, 2\mu - 1)$. In training and testing, jobs are given weight 1, 2 or 4, with probability $(0.2, 0.6, 0.2)$ [111]. Table 4.8 gives the settings for problem instances.

We now use 24 problem instances for testing, one instance per configuration, and have an additional 24 problem instances for extreme testing, one instance per configuration. The extreme testing uses a different processing time distribution and priority assignment to see how well DRs generalise to unfamiliar instances. In extreme testing problem scenarios, the processing times follow a geometric distribution with mean $\mu = 25$ (parameter $p = 0.04$). We also increase the number of possible weights given to jobs, including an additional weight for *very* important jobs; jobs are given weight 1, 2, 4 or 8, with probability $(0.2, 0.5, 0.2, 0.1)$. Due date tightness is equally likely from the available options which are the same or tighter than in training.

Changes to GP Parameter Settings. Our initial experiments had a small population size of 50, to restrain the computational time due to the incorporation of local search. We increase this to 100 in these experiments. Evolution remains at 50 generations, a standard setting. All other GP parameter settings remain the same.

Investigation of Best Use of Computational Time with Local Search

Local search is computationally expensive, hence we have restricted the use of local search for additional fitness evaluation to every 100 times a machine is ready to dispatch a new operation. We will gather data, and investigate potential improvements on how to best use computational time to apply local search. In the testing phase of each best-of-run DR from SOGP and every DR in the non-dominated front from MOGP we gather the following information:

- The queue length every time a dispatching decision is made. From this we report the mean, standard deviation and maximum recorded queue length of each problem instance.
- Every time a dispatching decision is made and local search is ap-

Table 4.8: Configuration of training, testing and extreme testing scenarios (processing time distribution, expected utilisation, due date tightness and operation setting) for extended local search based fitness experiments. Here "unif" is the discrete uniform(1,49) distribution, "geom" is the Geometric(0.04) distribution, F indicates "full" operations and V indicates "variable" operations.

	Dist	ρ	h	Ops		Dist	ρ	h	Ops
					Training				
TR1	unif	0.90	{2,3,4}	F	TR3	unif	0.90	{2,3,4}	V
TR2	unif	0.95	{2,3,4}	F	TR4	unif	0.95	{2,3,4}	V
		Testing					*Extreme Testing*		
T1	unif	0.80	4	F	XT1	geom	0.80	{2,3,4}	F
T2	unif	0.80	6	F	XT2	geom	0.80	{2,3,6}	F
T3	unif	0.80	8	F	XT3	geom	0.80	{2,2.5,3}	F
T4	unif	0.85	4	F	XT4	geom	0.85	{2,3,4}	F
T5	unif	0.85	6	F	XT5	geom	0.85	{2,3,6}	F
T6	unif	0.85	8	F	XT6	geom	0.85	{2,2.5,3}	F
T7	unif	0.90	4	F	XT7	geom	0.90	{2,3,4}	F
T8	unif	0.90	6	F	XT8	geom	0.90	{2,3,6}	F
T9	unif	0.90	8	F	XT9	geom	0.90	{2,2.5,3}	F
T10	unif	0.95	4	F	XT10	geom	0.95	{2,3,4}	F
T11	unif	0.95	6	F	XT11	geom	0.95	{2,3,6}	F
T12	unif	0.95	8	F	XT12	geom	0.95	{2,2.5,3}	F
T13	unif	0.80	4	V	XT13	geom	0.80	{2,3,4}	V
T14	unif	0.80	6	V	XT14	geom	0.80	{2,3,6}	V
T15	unif	0.80	8	V	XT15	geom	0.80	{2,2.5,3}	V
T16	unif	0.85	4	V	XT16	geom	0.85	{2,3,4}	V
T17	unif	0.85	6	V	XT17	geom	0.85	{2,3,6}	V
T18	unif	0.85	8	V	XT18	geom	0.85	{2,2.5,3}	V
T19	unif	0.90	4	V	XT19	geom	0.90	{2,3,4}	V
T20	unif	0.90	6	V	XT20	geom	0.90	{2,3,6}	V
T21	unif	0.90	8	V	XT21	geom	0.90	{2,2.5,3}	V
T22	unif	0.95	4	V	XT22	geom	0.95	{2,3,4}	V
T23	unif	0.95	6	V	XT23	geom	0.95	{2,3,6}	V
T24	unif	0.95	8	V	XT24	geom	0.95	{2,2.5,3}	V

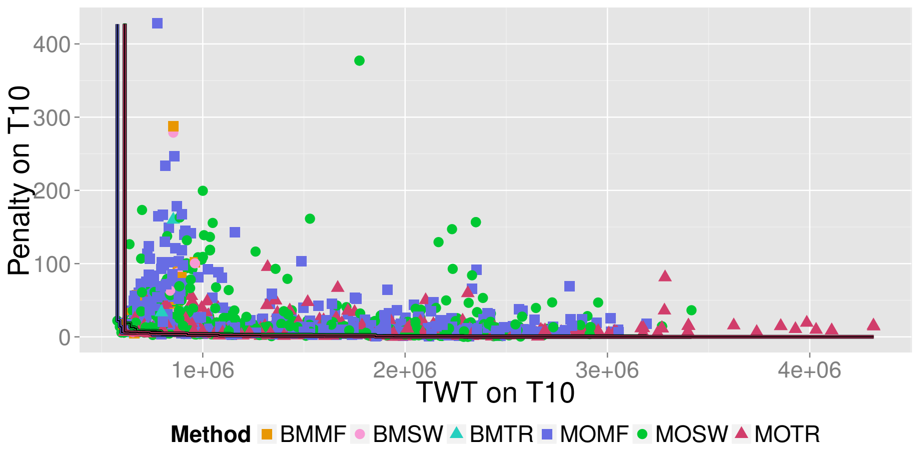
plied, we record the queue length when a positive penalty is assigned, as well as the penalty assigned. We also record the queue length when no penalty is assigned. From this we report the mean, standard deviation and maximum queue length for when positive penalty and no penalty is assigned. We also report the mean and standard deviation of penalty assigned for each queue length.

From this information we hope to be able to determine when the most important decisions are being made, i.e., whether it is more important to schedule a job correctly when there are few or many jobs in the queue. This information can enable us to use the application of local search more effectively, as it is such a time intensive process.

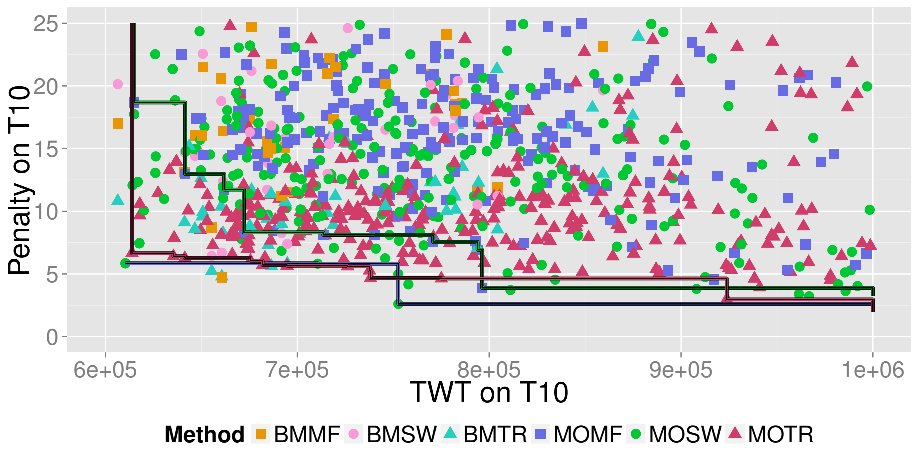
4.3.4 Experimental Results

To evaluate the three different local search operators, we performed 50 independent GP runs for each operator, with the same pseudo-random number generator seed. The front of non-dominated individuals from the final generation is tested on the test and extreme test instances. We also perform 50 GP runs of benchmark single objective GP, using TWT as the only objective. The best-of-run individuals are tested on the same test and extreme test instances, and the penalty incurred using each of the three local search operators is calculated during each simulation. Note that the best-of-run TWT values obtained from the benchmark GP evolved DRs are therefore the same for each LS operator, however the penalty values are different. We present the results of benchmark (BM) and LS-MO methods (MO) with MoveFront (MF), SwapFront (SW) and Transpose (TR) operators, and this gives six methods.

Figures 4.10 and 4.11 present the results of the scenarios with the same parameters as Figures 4.8 and 4.9 from the initial results. On these scenarios, it is still the case that the LS-MO methods achieve lower (better) TWT values than the benchmark methods. We also note that the longer warm up, greater number of jobs for which data is collected, and larger



(a) Full results of individuals from non-dominated fronts and aggregate non-dominated fronts from MOMF, MOSW and MOTR and best-of-run DRs from BMMF, BMSW and BMTR.



(b) Close up of aggregate non-dominated fronts and best DRs.

Figure 4.10: Aggregate non-dominated fronts from MOMF, MOSW and MOTR, and best-of-run individuals from BMMF, BMSW and BMTR on scenario T10.

population size through training, has led to an aggregate non-dominated front which attains lower TWT and lower penalty values than in the initial experiments, which is as expected.

At a quick glance, as in the full picture of Figures 4.10(a) and 4.11(a), all three operators have a similar aggregate non-dominated front, with slight differences in the distribution of evolved DRs. In Figures 4.10(b) and 4.11(b) we are able to see the differences between the local search operators. On scenario T10, MOSW has the best non-dominated front, with many DRs in the aggregate front having lower TWT values than both MOMF and MOTR. However, on scenario XT12 it is harder to differentiate between the three MO methods, and all three have DRs which attain lower TWT than the minimum attained by the benchmark GP.

Figure 4.12 presents a plot showing the distribution of the normalised TWT and normalised penalty values attained by the best-of-run DRs and DRs from the non-dominated fronts. These values were attained as follows. The fitness of a DR on scenario i is normalised by

$$f_{norm,i} = \frac{f_i - f_{min,i}}{f_{max,i} - f_{min,i}} \tag{4.7}$$

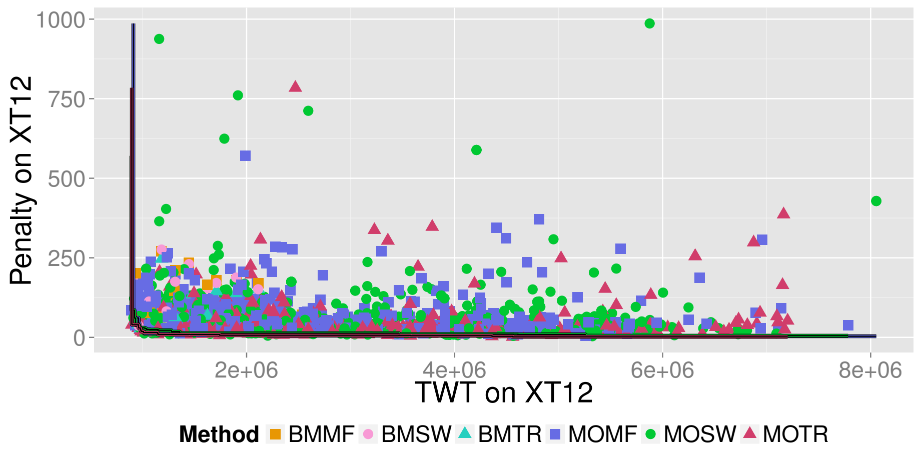
where $f_{min,i}$ is the minimum TWT value attained on scenario i across *all methods*, and $f_{max,i}$ is the maximum TWT value attained on scenario i across all methods. This gives two summary fitness values:

$$f_t = \tfrac{1}{24} \sum_{i=1}^{i=24} f_{norm,i} \qquad\qquad f_{xt} = \tfrac{1}{24} \sum_{i=25}^{i=48} f_{norm,i} \tag{4.8}$$

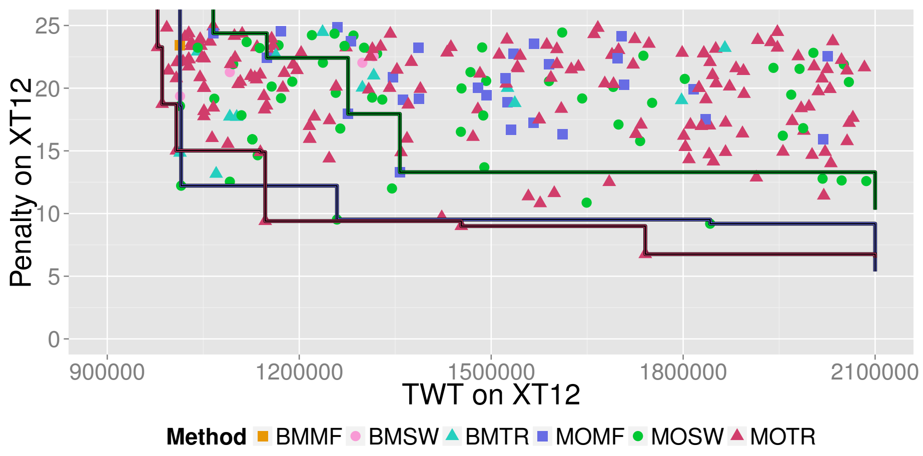
and one overall fitness value

$$f_{TWT} = \tfrac{1}{2} \left(f_t + f_{xt}\right) \tag{4.9}$$

Penalty values were normalised using the same method. Figure 4.12 shows that the best overall TWT values were attained by DRs evolved by MOSW, followed by MOTR; however the lowest penalty values were attained by DRs evolved by MOTR. The plot also looks similar to the plots from T10

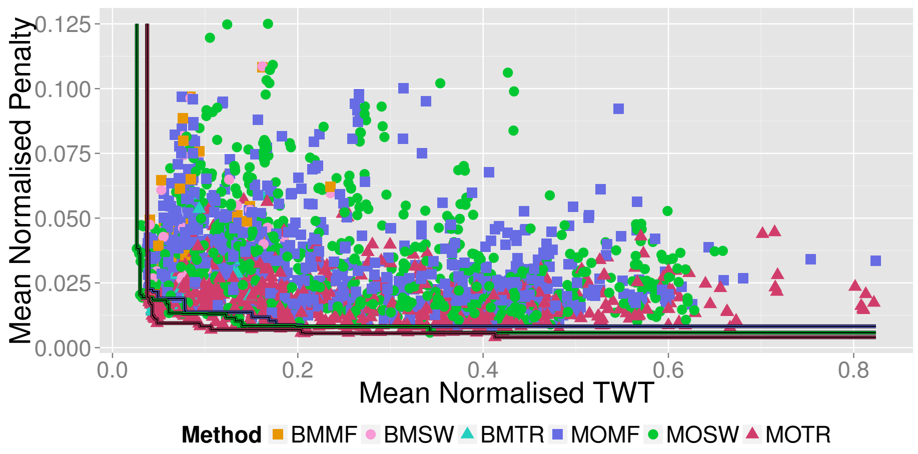


(a) Full results of individuals from non-dominated fronts and aggregate non-dominated fronts from MOMF, MOSW and MOTR and best-of-run DRs from BMMF, BMSW and BMTR.

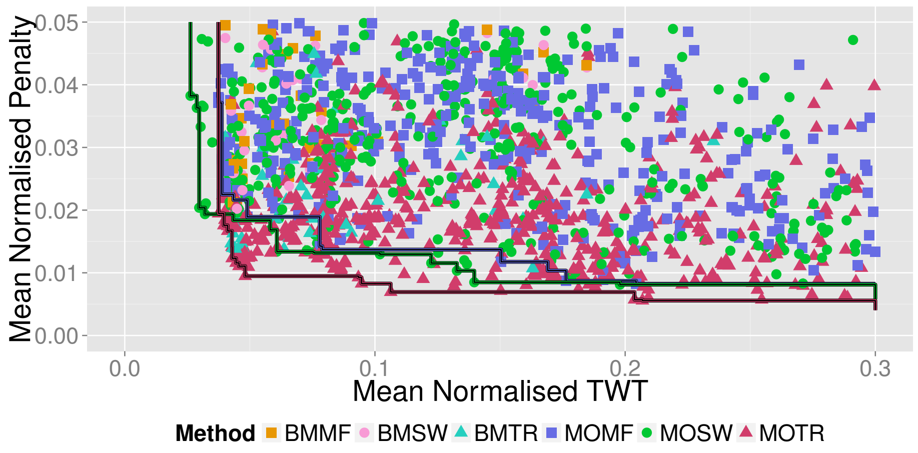


(b) Close up of aggregate non-dominated fronts and best DRs.

Figure 4.11: Aggregate non-dominated fronts from MOMF, MOSW and MOTR, and best-of-run individuals from BMMF, BMSW and BMTR on scenario XT12.



(a) Full results using normalised values.



(b) Close up on non-dominated fronts of normalised values.

Figure 4.12: Normalised TWT and penalty values across all test and extreme test instances.

and XT12, suggesting that relative performance across all instances is similar to T10 and XT12.

We want to identify the difference between SwapFront, Transpose and MoveFront to see what reasons there may be for the better performance of the DRs evolved by SwapFront. Better performance suggests that the feedback provided by SwapFront is better than the feedback provided using the other two operators, and therefore better identifies individual DRs which are fitter over this extended decision horizon and hence are less-myopic. SwapFront is likely to be considerably changing the expected completion time of two jobs in the queue, MoveFront will be considerably changing the value of one job in the queue, and Transpose only changes the expected completion time of two jobs in the queue. As the penalty value is assigned by $penalty = Total_0 - Total_{min}$, it only takes *one* possible queue order to have a better expected contribution to TWT for the penalty to be assigned. As the penalty function is the overall mean, we are unable to tell if a rule made several very bad (and high penalty) dispatching decisions, or was consistently scheduling jobs in a way that could be slightly improved across the extended decision horizon. If we had the computational time to be able to apply local search to *select* which job will be dispatched then Transpose would only change the dispatched job in one neighbourhood, whereas with both SwapFront and MoveFront each job appears at the front of the queue in one neighbourhood.

Queue and Penalty Properties. Figures 4.13 and 4.14 shows violin plots of the distribution of the mean queue length every time a dispatching decision is made beside the mean queue length when a positive penalty value is assigned and when there is no penalty assigned for evolved DRs from BMMF. There is one value for each evolved best-of-run DR and each DR in the non-dominated front. As local search was only used every 100 times the DR was called to make a dispatching decision, fewer values were used to calculate the mean for when penalty was and was not assigned. Local search is also not used when there is only one job in the queue, as there is no other possible queue order in this case. The violin plots show that for BMMF, on average the queues are longer when a penalty is assigned. Similar trends were shown for all other methods. The standard deviation on queue lengths are very large. In all figures, the Transpose method has the lowest mean penalty per queued job, followed by the MoveFront and SwapFront methods.

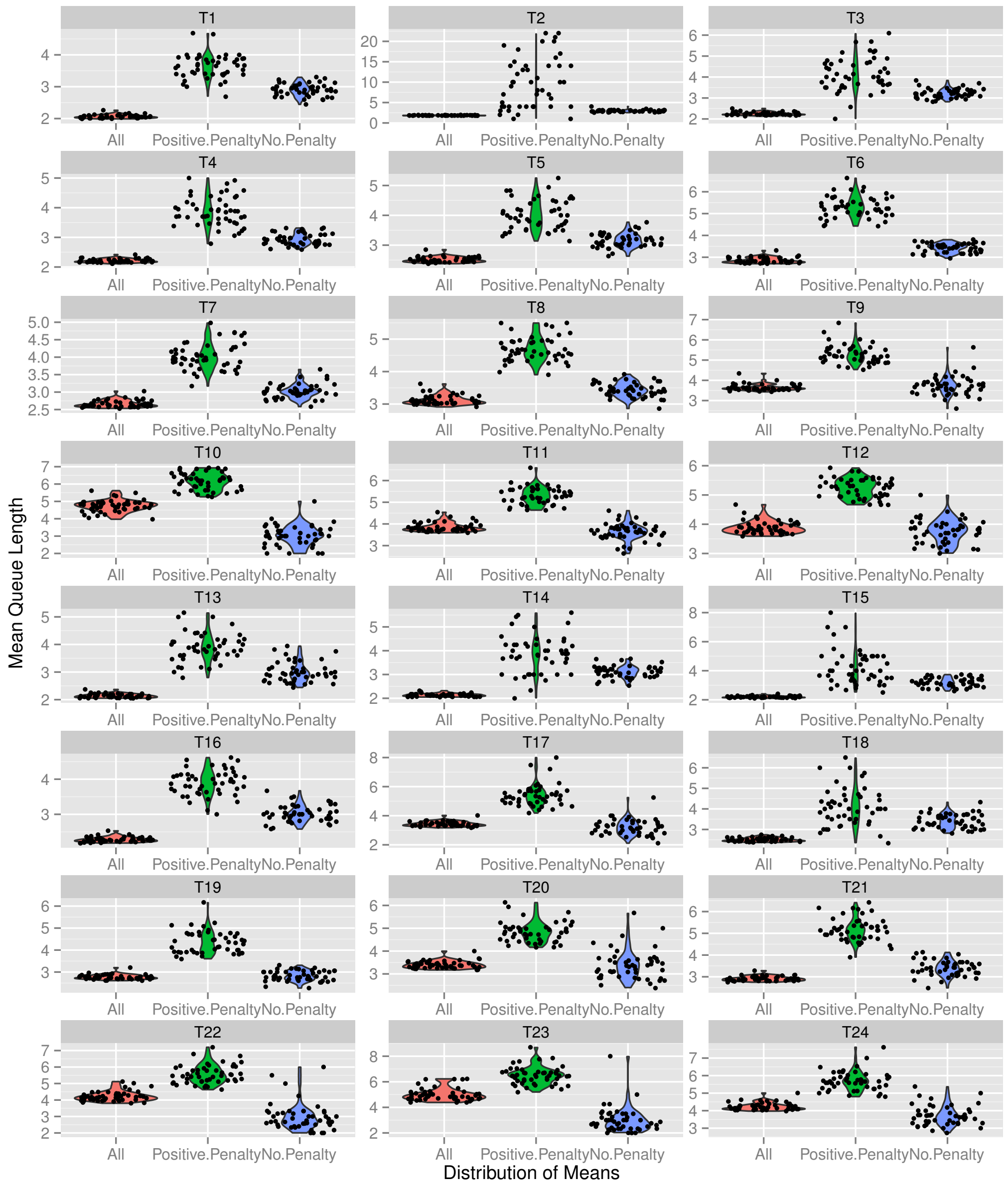


Figure 4.13: Violin plot of mean queue length over all dispatch points, mean queue length when local search is used and *positive penalty* is assigned, and mean queue length when local search is used and *no penalty* is assigned with BMMF on test instances.

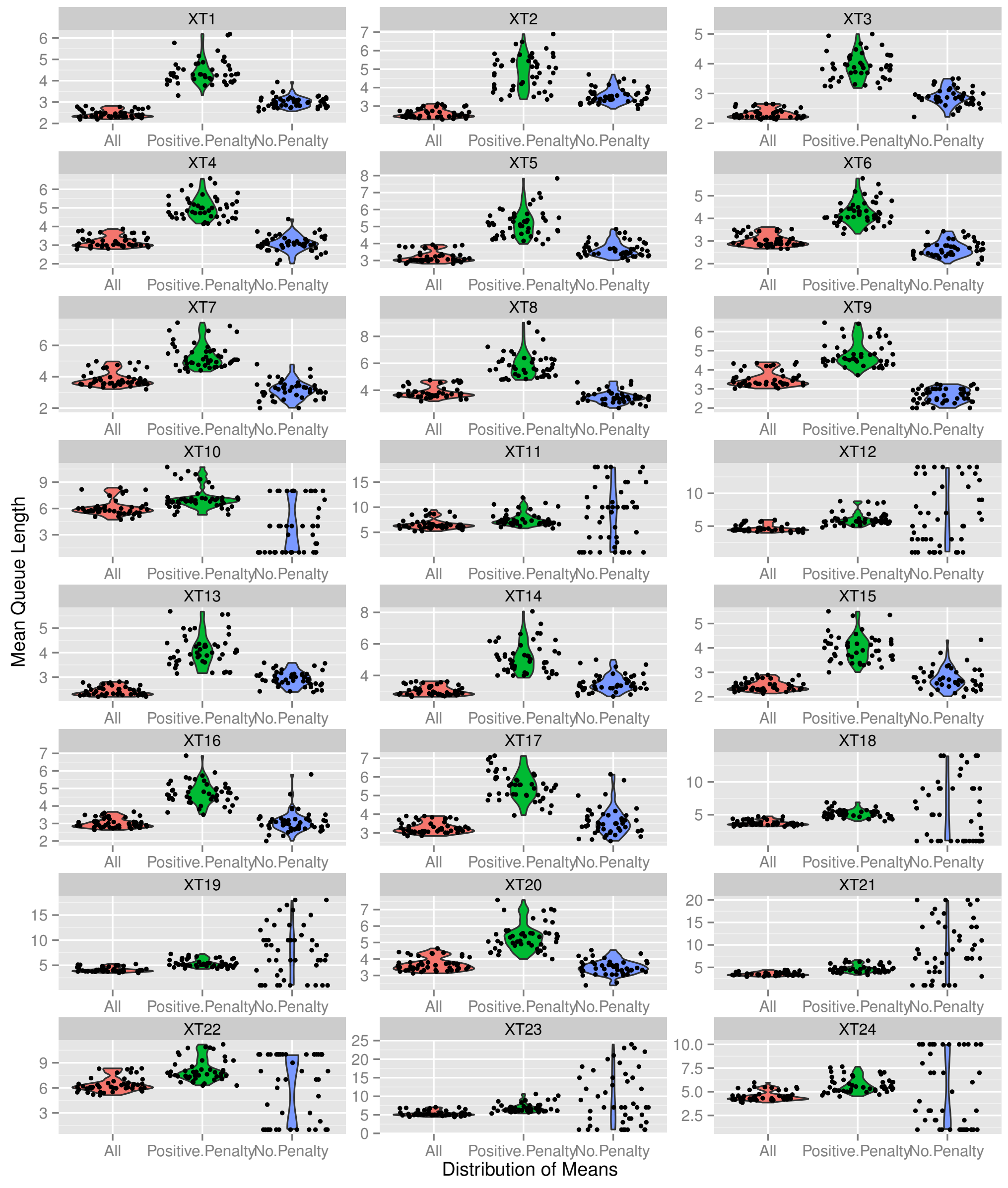
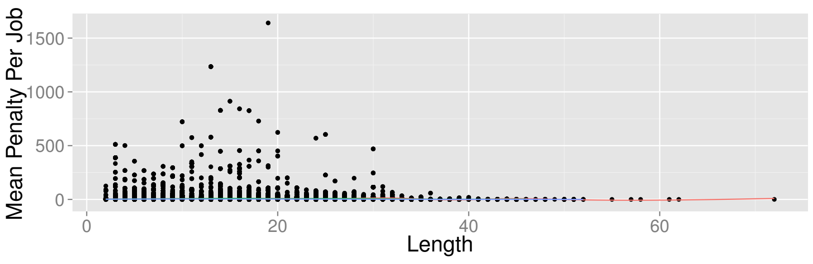
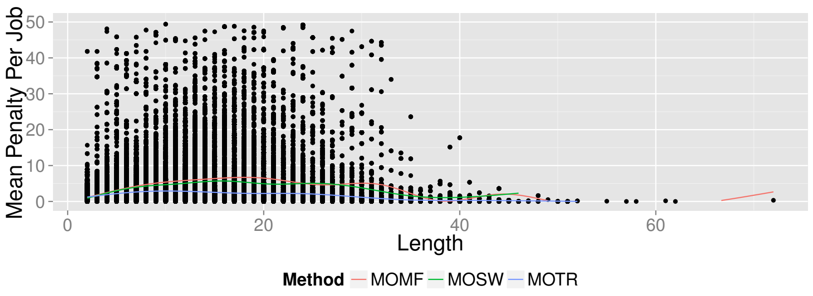


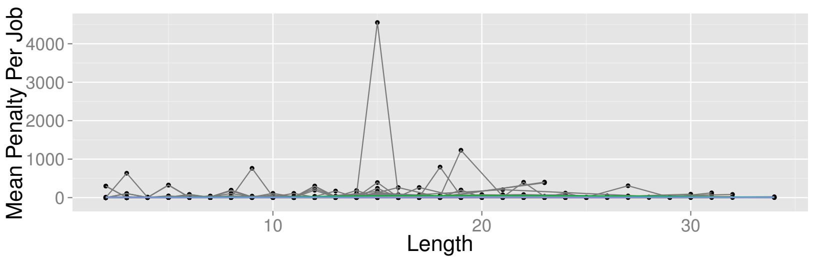
Figure 4.14: Violin plot of mean queue length over all dispatch points, mean queue length when local search is used and *positive penalty* is assigned, and mean queue length when local search is used and *no penalty* is assigned with BMMF on extreme test instances.



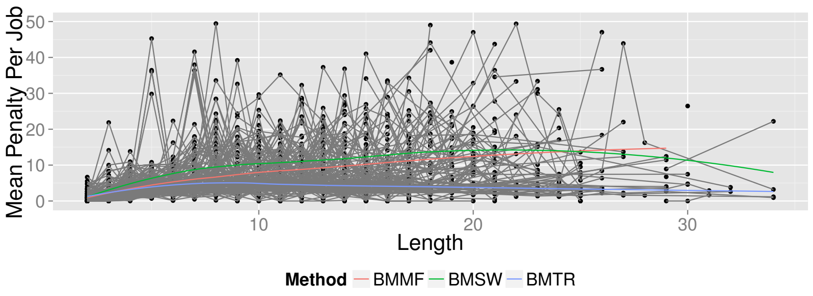
(a) Full results of LS-MO methods.



(b) Close up of LS-MO methods.



(c) Full results of BM methods.



(d) Close up of BM methods.

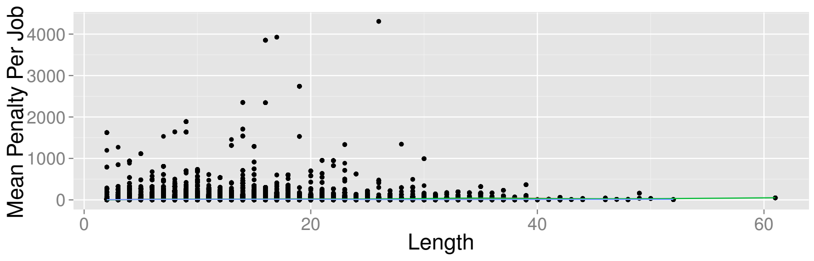
Figure 4.15: Mean penalty per queued job for all queue lengths on scenario T10.

Figure 4.15 shows the mean penalty for each queue length on test scenario T10 for all evolved DRs from BM methods (Figures 4.15(c) and 4.15(d)) and LS-MO methods (Figures 4.15(a) and 4.15(b)). For the BM methods we connect points from the same DR; for the LS-MO methods there are too many to plot clearly. Figure 4.16 shows the mean penalty for each queue length on extreme test scenario XT12 for all evolved DRs from BM methods (Figures 4.16(c) and 4.16(d)) and LS-MO methods (Figures 4.16(a) and 4.16(b)). The close up of both LS-MO and BM methods on both test scenario T10 and extreme test XT12 show the trend that the mean penalty per queued job increases from jobs 1 to 10, then plateaus. The plots also show that queue lengths get very long, over 60 in at least one instance on both scenarios.

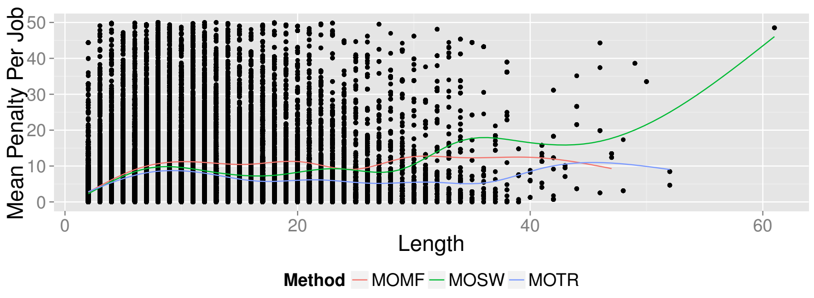
Best Evolved Dispatching Rules. Figure 4.17 shows the best evolved DR from all six methods. This DR was evolved by MOSW and has not been simplified. This DR has 22 terminal nodes, of which 14 are wider looking terminals. Although some of these less myopic terminals are in the branches of the `if>0` statements and therefore will not always be evaluated in assigning priorities, the majority of the less myopic terminals will be evaluated during priority assignment. This reinforces the conclusions of Section 4.2 that these terminals improve TWT performance and are indeed "less-myopic". Once again we also note the subtraction of values that we do not consider to be semantically valid, e.g., the number of jobs in the next queue is subtracted from the job's importance weighting. Although this DR is effective at dispatching jobs from the queue, it is difficult to provide insight into why it is effective, partly due to the large number of `min`, `max` and `if>0` statements.

4.3.5 Local Search Conclusions

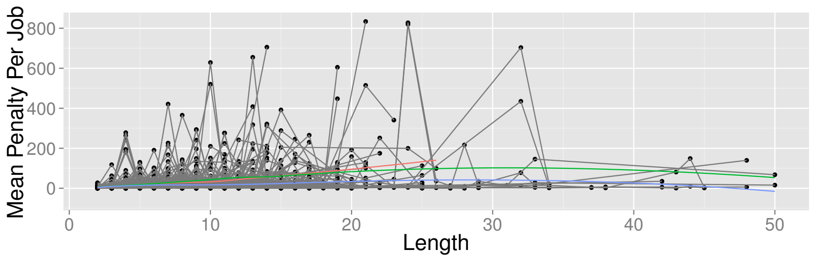
The goal of this section was to investigate the possible improvements to training of DRs for the dynamic ten-machine job shop in GP through the



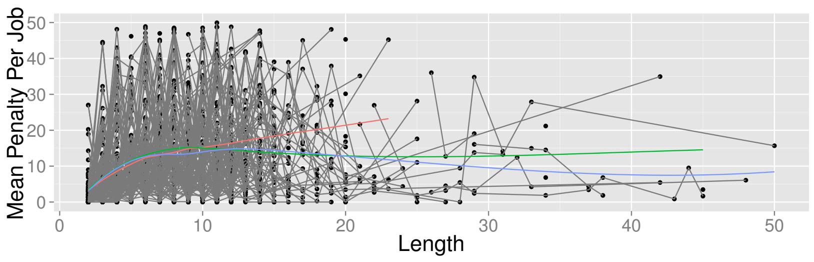
(a) Full results of LS-MO methods.



(b) Close up of LS-MO methods.



(c) Full results of BM methods.



(d) Close up of BM methods.

Figure 4.16: Mean penalty per queued job for all queue lengths on scenario XT12.

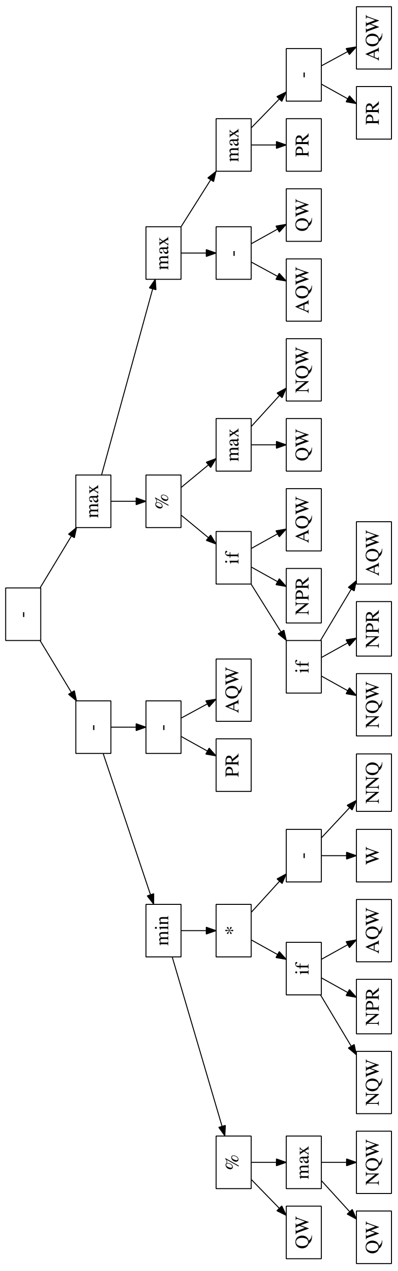


Figure 4.17: Evolved DR with best overall performance, evolved by MOSW.

use of local search. We implemented a local search based penalty, which punished DRs which were not scheduling jobs in the best order based on the current projected contribution to the TWT objective function. Initial results show that the inclusion of the local search penalty in evaluation has led to the evolution of DRs which have better local performance, and achieve some better TWT values. Our more in-depth investigation supports the findings of the initial investigation and shows that the SwapFront operator often leads to the non-dominated front of DRs which is closest to the best out of the three neighbourhood search operators investigated.

Analysis of the lengths of queues when penalties were assigned, and when there was no improvement to be made in terms of the contribution to TWT, showed that there is a huge overlap in mean$\pm$standard deviation. However the cluster of means shows that, as would be expected, the mean queue length is longer when the expected contribution to the TWT can be improved, than when it cannot. This suggests that in future work, using local search to evaluate the dispatching performance when queue lengths are longer could be more effective. This could be seen as treating machines as "bottlenecks" when they have a large queue of work awaiting processing.

4.4 Local Search to Investigate Tie Breaking

In this section, the local search based additional fitness evaluation described in Section 4.3.1 is used to investigate how often a dispatching rule assigns the same priority value to different jobs, and whether the default tie break, of using the shortest processing time (SPT) rule, is selecting the best job in those situations.

An effective DR should not assign the same priority to two jobs that do not have the same properties. This motivates us to explore the possibilities of encouraging the evolution of dispatching rules which *do not* often assign the same priority to multiple queued jobs, by using local search to penalise dispatching rules which assign the same priority to jobs with different properties.

The aim is to find out whether these methods can reduce the number of ties in test cases, and whether the TWT performance is improved.

4.4.1 Local Search Fitness for Tie Breaking

This investigation uses local search among the sub-schedule of jobs which have been assigned the *same* highest priority value in the queue. Sub-schedules are compared based on their expected contribution to the objective function. In order to evaluate how well the default tie break is working, two discrete-event simulations are run for each problem instance. In the first, we do not use local search, and ties are broken using SPT, which gives TWT_0. In the second, we use local search to alter which of the jobs that are tied for top priority is dispatched, dispatching the job which leads to the lowest expected contribution to TWT, and the resulting final value of the objective function is TWT_{LS}.

Local search is performed as described in Section 4.3.1, with local search operator SwapFront. The expected contribution to TWT is given by Equation (4.5). Due to computational restraints, we use local search only every second time a tie occurs. Two methods are investigated.

Tie Breaking Single Objective (TB-SO). This variant uses single objective GP, combining the TWT fitness with an added penalty if the performance of the DR was better with local search, i.e., if $TWT_{LS} < TWT_0$. The fitness of an individual for a problem instance is

$$TWT_0 + \max\{TWT_0 - TWT_{LS}, 0\}.$$

Tie Breaking Multiobjective (TB-MO). This variant uses a multiobjective approach. The first fitness objective is TWT_0, and the second fitness objective is the penalty for possible improvement on the schedule the DR produced, $\max\{TWT_0 - TWT_{LS}, 0\}$. We use the NSGA-II algorithm [32].

4.4.2 GP System Setup

We use the same GP system for evolving DRs, using ECJ20 [78], as for the initial experiments of Section 4.3.1 investigating local search for local decision making. This includes the use of the same terminal set as detailed in Table 4.1 and the same function set, $\{+, -, *, \%, \texttt{if>0}, \texttt{max}, \texttt{min}\}$.

Problem Instances

We randomly create problem instances of job shops with ten machines. We use four problem instances for training (see Table 4.9). We present results on two test and two extreme test scenarios. The extreme testing is to see how well the rules generalise to a job shop with different distribution and priority assignment.

We are dealing with dynamic JSS, so jobs arrive stochastically according to a Poisson process with rate λ for all problem instances. The settings for these are shown in Table 4.9. The desired expected utilisation of machines (ρ) is achieved by setting $\lambda = \frac{\rho}{(\mu \times p_M)}$, where p_M is the expected number of machines a job will visit (i.e. the expected number of operations in each job). Job due dates are set as in [5], $d_j = r_j + h \times \sum_{l=1}^{N_j} p(\sigma_{j,l})$, where h is a due date tightness parameter, randomly chosen with equal probability from the choices available for each job. Jobs are given weight 1, 2 or 4, with probability $(0.2, 0.6, 0.2)$ [111].

Four training scenarios are used. The processing times at each machine follow a discrete uniform distribution with mean μ, i.e., $U(1, 2\mu - 1)$. A warm up period of 100 jobs is used, and we collect data from the next 200 jobs to arrive ($N = 200$), however new jobs keep arriving in the system until the 300th job is completed. This is a very low number of jobs due to the increase in computational time required by local search.

In testing, we increase the warm up period to 500 jobs and increase the number of jobs we collect data from to $N = 1000$. In testing scenarios T1 and T2, all jobs have due date tightness of 4, and utilisation is 0.95, T1 has

Table 4.9: Configuration of training and testing scenarios (mean of processing time distribution, expected utilisation, due date tightness parameter and operation setting) for local search based tie breaking.

		μ	ρ	h	Operations
Training	TR1	25	0.90	{2,3,4}	full
	TR2	25	0.95	{2,3,4}	full
	TR3	25	0.90	{2,3,4}	variable
	TR4	25	0.95	{2,3,4}	variable
Testing	T1	25	0.95	4	full
	T2	25	0.95	4	variable
Extreme Testing	XT1	25	0.95	{2,2.5,3}	full
	XT2	25	0.95	{2,2.5,3}	variable

"full" operations, i.e., each job has 10 operations, and T2 has "variable" operations, i.e., the number of operations is between 2 and 10. In extreme testing, XT1 and XT2, the processing times follow a geometric distribution with mean $\mu = 25$ (parameter $p = 0.04$), and utilisation is 0.95. We also change the weights given to jobs, including an additional weight for *very* important jobs; jobs are now given weight 1, 2, 4 or 8, with probability $(0.2, 0.5, 0.2, 0.1)$. Due date tightness is equally likely from {2,2.5,3}; these are the same or tighter than in training. XT1 has "full" operations, i.e., each job has 10 operations, and XT2 has "variable" operations, i.e., the number of operations is between 2 and 10.

GP Parameters

The population size is 50, this small population size is a trade-off for the extra computational time required for local search. The initial population is generated using the ramped-half-and-half method [70] with initial minimum depth of two and maximum depth of six. This small population size is to restrain the computational time due to the incorporation of local search. Evolution is for 50 generations, a standard setting. GP trees have a maximum depth of six. Genetic operators crossover, mutation and elitism, use rates of 85%, 10% and 5% respectively. Tournament selection with a tournament size of seven is used to select individuals for genetic operators. This is a common setting that has been previously used [95].

4.4.3 Results

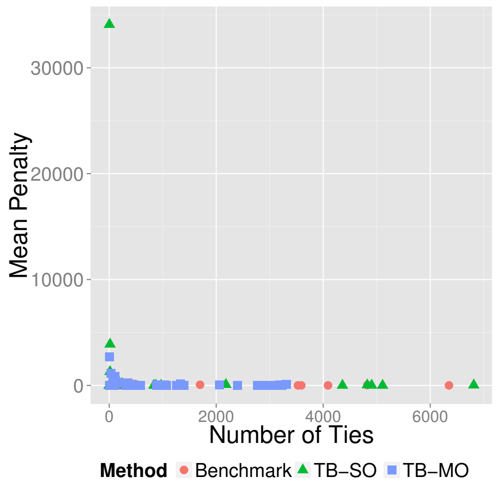
Figure 4.18 plots the mean penalty (penalty/(number of ties)) vs the number of ties for the test and the extreme test instances. These plots show that some DRs with a low number of ties have a very high penalty. These DRs encounter few ties, but when they do they frequently make a worse decision in terms of the expected contribution to TWT, i.e., the default tie break of SPT is not effective. There are also DRs with low mean penalty values for a range of number of ties that occur. For these rules the default tie break of SPT is effective. However, we must also consider how these values relate to the attained TWT values.

The TB-SO method produced the DRs which attained the lowest TWT value on three of the four scenarios. The exception is scenario T2, in which TB-MO and the benchmark attain the equal best TWT performance, which is clearly lower than the best TB-SO DR.

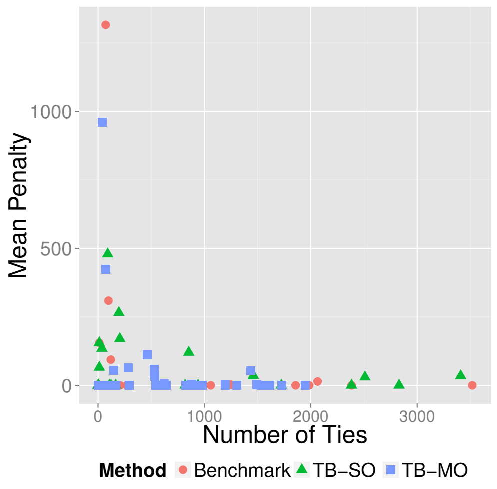
Most of the best evolved DRs have a penalty of 0; which supports our hypothesis that classifiers which are better at separating jobs by assigning distinct priorities are more effective. The spread of TWT and penalty results is similar across all four scenarios, which suggests that performing tie breaking with local search *does not* offer enough improvement to justify the additional computational cost incurred.

4.5 Chapter Summary

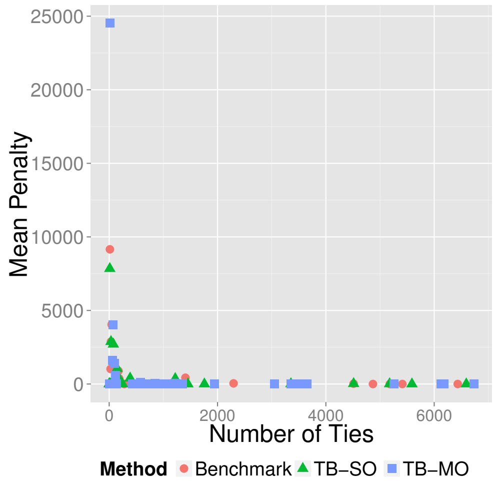
This chapter has explored the development of methods to automatically discover dispatching rules which are less-myopic using genetic program-



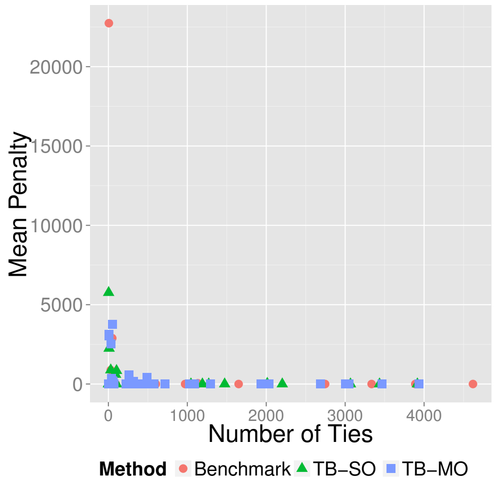
(a) T1



(b) T2

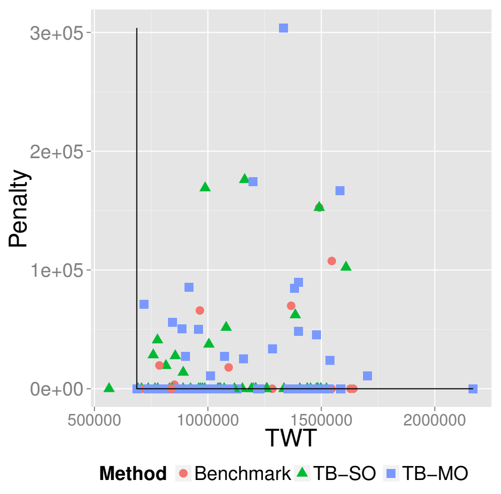


(c) XT1

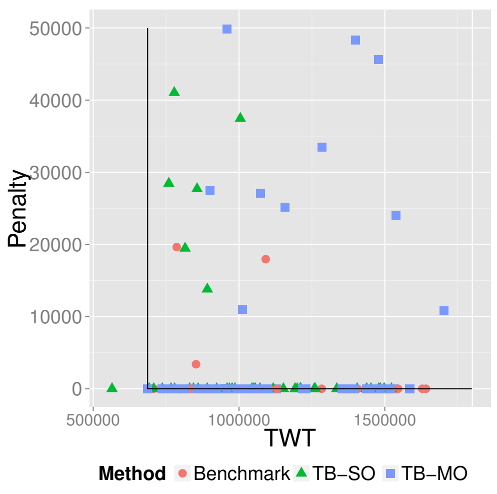


(d) XT2

Figure 4.18: Plots of Mean Penalty for each Tie frequency across test and extreme test instances.

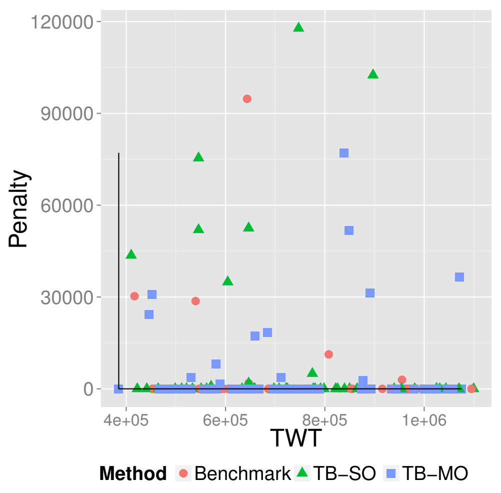


(a) Non-dominated fronts from TB-MO and best-of-run individuals from TB-SO.

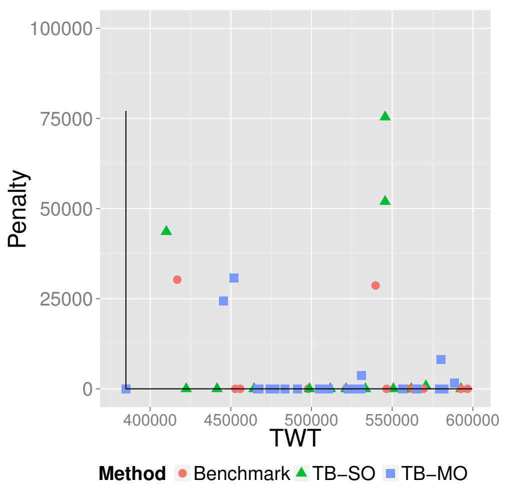


(b) Close up of lowest TWT attained.

Figure 4.19: Aggregate non-dominated fronts from TB-MO, and best-of-run individuals from TB-SO and Benchmark GP on scenario T1.

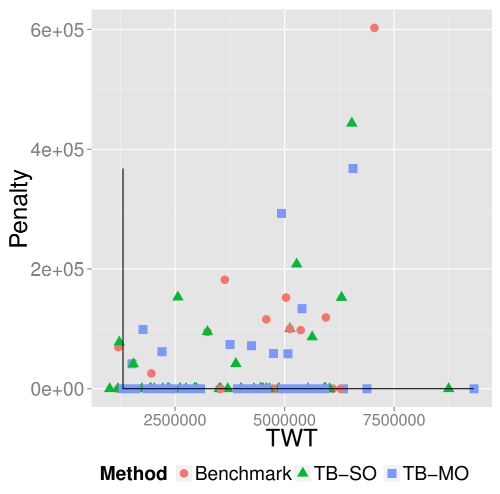


(a) Non-dominated fronts from TB-MO and best-of-run individuals from TB-SO.

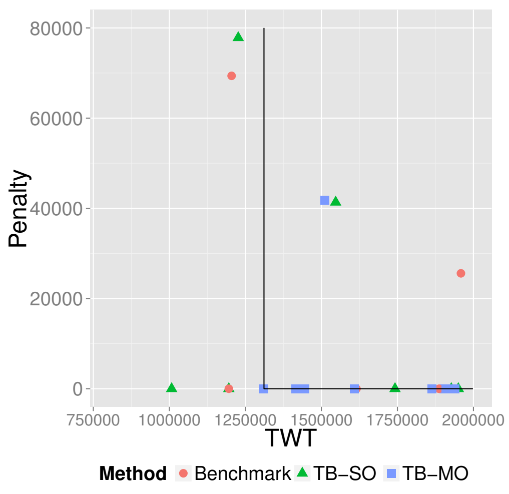


(b) Close up of lowest TWT attained.

Figure 4.20: Aggregate non-dominated fronts from TB-MO, and best-of-run individuals from TB-SO and Benchmark GP on scenario T2.

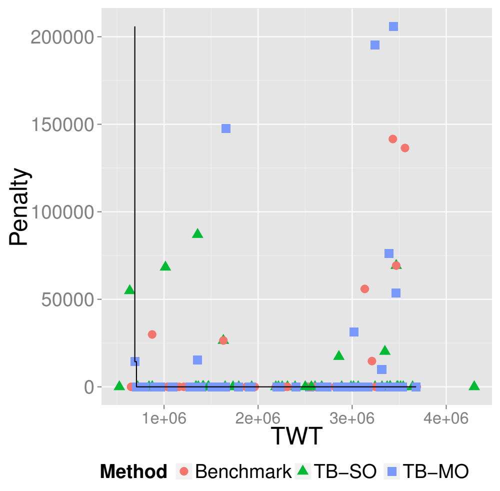


(a) Non-dominated fronts from TB-MO and best-of-run individuals from TB-SO.

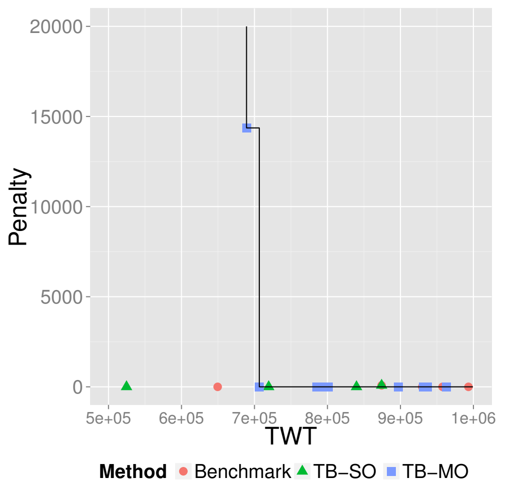


(b) Close up of lowest TWT attained.

Figure 4.21: Aggregate non-dominated fronts from TB-MO, and best-of-run individuals from TB-SO and Benchmark GP on scenario XT1.



(a) Non-dominated front from non-dominated fronts from TB-MO and best-of-run individuals from TB-SO.



(b) Close up of lowest TWT attained.

Figure 4.22: Aggregate non-dominated fronts from TB-MO, and best-of-run individuals from TB-SO and Benchmark GP on scenario XT2.

ming.

Results show that in the dynamic ten-machine job shop, incorporating features of the state of the wider shop, and the stage of a job's journey through the shop, improves the mean performance, and decreases the standard deviation of performance of the best evolved rules.

A direction for future work is to investigate whether the additional less-myopic terminals continue to offer improved performance as the scale of the job shop increases. Another suggestion is to investigate additional terminals which capture further properties of the job shop's current and potential future states, and incorporate a look-ahead element.

Results show that the inclusion of local search in evaluation during training led to the evolution of dispatching rules which make better decisions over the local time horizon, and attain lower total weighted tardiness. The advantages of using local search as a tie breaking mechanism are not so pronounced.

Results from the work of this chapter has also raised the issue of interpretability of evolved dispatching rules. In the next chapter this issue will be explored through the use of strongly typed GP to constrain the search space [80]. This will reduce the number of the comparisons which do not make semantic sense and enable better understanding of what information effective rules are capturing from the shop system.

Chapter 5

Evolving Dispatching Rules with Greater Understandability

5.1 Introduction

A major issue with dispatching rules evolved automatically by genetic programming is their interpretability. The interpretability of heuristics evolved by genetic programming, and other optimisers, has been identified as a *crucial* aspect to gain the trust of operators or managers [16]. How well, or even whether, these people are able to understand and interpret *how* and *why* the dispatching rule works is therefore important if the dispatching rules evolved using evolutionary computation techniques are to be used in real-world situations.

One particular limitation is that many of the comparisons made in evolved dispatching rules cannot be readily interpreted in terms of units, i.e., semantically most dispatching rules evolved using GP are incorrect. Knowing how well a given rule will generalise across different distributions of arrivals and processing times, and different scales, is also necessary if they are to be used in practice.

The aim is to improve the overall interpretability of evolved rules as a whole, thus improving the trust that can be placed in their performance.

Individual rules can be examined and modified manually, but this is a time consuming process requiring trial and error. If the use of strongly typed GP (STGP) [85] can, on average, improve the interpretability of evolved dispatching rules and keep human intervention to a minimum, then GP can be considered an effective tool for automatically generating dispatching rules.

Another factor in the interpretability of evolved dispatching rules is their size. It is clear that small, simple dispatching rules are easier to interpret. Likewise, it is intuitive that complex situations cannot be handled by a simple rule. Job shop scheduling is just one environment where this holds true. The allowable tree depth for most approaches using GP allows dispatching rules which are much larger than most dispatching rules from the literature. The increased size allows more complex dispatching rules, which may be able to discover more of the relationships and interactions amongst shop properties and therefore improve performance. However, the larger the DR, the more difficult it is to understand how it works and compare how a change, i.e., in processing time, would alter the priority of a given job.

Results in Chapter 4 suggested that the best-of-run dispatching rules evolved using standard GP are not easily interpretable. This is due partly to the way that features are combined with mathematical operators, e.g., it makes more sense to multiply or divide by the job weight than to add it to or subtract it from the current time.

Jakobović and Marasović [58], in their work evolving priority functions through GP within a meta-algorithm to form a scheduling heuristic, noted that "GP solutions are not 'analytically correct' ", and that solutions could be restricted so that only semantically correct expressions remain (e.g. not allowing processing time minus the number of unscheduled jobs). The authors state that preliminary results showed no statistically significant differences compared to standard tree-based GP. However, it is not stated whether comparisons were made only in terms of performance on test cases or also included consideration of the interpretability, reliability and generalisation ability (robustness) of the evolved rules, which we believe are important aspects that must be taken into account when considering how 'good' a DR is.

Intuitively we expect that a very simple DR may be easy to interpret but is likely to perform poorly in complex decision situations. However, it is desirable that a well performing (effective) DR be interpretable at least in commonly used components, and when these are selected (conditions).

In this chapter the job shop investigated is again the ten-machine dynamic job shop, with the objective of minimising the total weighted tardiness. The ATC and WCOVERT rules introduced in Section 2.2.3 (see page 42) are very good, small, compact and interpretable dispatching rules. These are major reasons that they are used in practice and are excellent examples of the rules we are trying to find automatically. Grammar-based GP has been used as a hyper-heuristic for the automatic generation of timetabling heuristics for the exam timetabling problem [4]. The advantages of grammar-based approaches are that it restricts the search space and encodes knowledge of the problem domain. STGP is one form of grammar-based GP. In STGP the grammar is used to prescribe the type system. STGP does not have all the benefits of other forms of grammar-guided GP; online grammar adaption is not possible in STGP. However, grammar-based GP approaches (including STGP) have not been used to improve the interpretability of DRs in dynamic job shop scheduling.

5.1.1 Chapter Goals

The aim of this chapter is to use a grammar-based approach [80] implemented through the STGP capabilities of ECJ20 [78] to evolve DRs for the multi-machine dynamic job shop environment. By using STGP to implement restrictions described by a grammar we constrain the search space, aiming to develop evolved rules with greater interpretability than standard GP, and therefore enabling better understanding of what information effective rules are capturing from the shop system. In particular we aim to address the following research objectives.

1. Develop terminal types and grammar(s) to specify the allowable interactions between these types which can be enforced with the use of STGP.

2. Investigate how the level of interpretability and performance compares between rules evolved using STGP, those evolved by the traditional GP based generation of dispatching rules, and manually designed rules from the literature.

3. Evaluate whether the use of STGP can be justified (if performance is worse) as an acceptable compromise for improvement in interpretability.

4. Investigate what insight can be gained towards developing a measure to evaluate and compare interpretability of automatically evolved rules.

5. Investigate whether the restriction of the search space through STGP can reveal good components or combinations of components that we can use to further restrict or alter the terminals included in the terminal set.

5.1.2 Chapter Organisation

The remainder of this chapter is organised as follows. Section 5.2 explains the proposed approach. Sections 5.3 and 5.4 present the results and analysis. Section 5.5 further discusses the results. Section 5.6 summarises the chapter.

5.2 Method and Experimental Design

This section describes the GP system used for the evolution of DRs and the JSS scenarios used for training and testing of the DRs.

5.2.1 GP System for Automatic Generation of DRs

The GP system is as described in Chapter 4 (see page 110). For each problem instance, the objective of interest, TWT, is calculated, and normalised (by dividing by the average expected utilisation across all machines), and the mean over all problem instances is set as the fitness of the DR. The best-of-run DR is tested on independent test and extreme test problem instances.

5.2.2 Job Shop Scenarios

Recall from Chapter 4 how we randomly generate scenarios using the properties given in Table 5.1 to determine the arrival rate using Equation (4.1) (see page 110). Note that Table 5.1 has different values in it from Table 4.8 (see page 137). Due dates are assigned using Equation (4.2) (see page 110).

A warm up period of 500 jobs is used ([52] finds this to be sufficient for the system to reach steady state), and we collect data from the next 2000 jobs to arrive ($N = 2000$), however new jobs keep arriving in the system until the 2500th job is completed.

Training

Four job shop problem instances are used for training the population of GP individuals (DRs), one from each scenario in Table 5.1. At each generation, each rule is used to dispatch jobs for the four scenarios. Each TWT value is normalised by the expected utilisation rate, and the average of these four normalised TWT values is used as the fitness for the GP individual. For

Table 5.1: Configuration of training, testing and extreme testing scenarios (processing time distribution, expected utilisation, due date tightness and operation setting). Here "unif" is the discrete uniform(1,49) distribution, "geom" is the Geometric(0.04) distribution, F indicates "full" operations and V indicates "variable" operations.

	Dist	ρ	h	Ops		Dist	ρ	h	Ops
Training									
TR1	unif	0.85	$\{3,5,7\}$	F	TR3	unif	0.85	$\{3,5,7\}$	V
TR2	unif	0.95	$\{3,5,7\}$	F	TR4	unif	0.95	$\{3,5,7\}$	V
Testing					*Extreme Testing*				
T1	unif	0.80	4	F	XT1	geom	0.80	$\{4,6,8\}$	F
T2	unif	0.80	6	F	XT2	geom	0.80	$\{3,5,7\}$	F
T3	unif	0.80	8	F	XT3	geom	0.80	$\{3,4,5\}$	F
T4	unif	0.85	4	F	XT4	geom	0.85	$\{4,6,8\}$	F
T5	unif	0.85	6	F	XT5	geom	0.85	$\{3,5,7\}$	F
T6	unif	0.85	8	F	XT6	geom	0.85	$\{3,4,5\}$	F
T7	unif	0.90	4	F	XT7	geom	0.90	$\{4,6,8\}$	F
T8	unif	0.90	6	F	XT8	geom	0.90	$\{3,5,7\}$	F
T9	unif	0.90	8	F	XT9	geom	0.90	$\{3,4,5\}$	F
T10	unif	0.95	4	F	XT10	geom	0.95	$\{4,6,8\}$	F
T11	unif	0.95	6	F	XT11	geom	0.95	$\{3,5,7\}$	F
T12	unif	0.95	8	F	XT12	geom	0.95	$\{3,4,5\}$	F
T13	unif	0.80	4	V	XT13	geom	0.80	$\{4,6,8\}$	V
T14	unif	0.80	6	V	XT14	geom	0.80	$\{3,5,7\}$	V
T15	unif	0.80	8	V	XT15	geom	0.80	$\{3,4,5\}$	V
T16	unif	0.85	4	V	XT16	geom	0.85	$\{4,6,8\}$	V
T17	unif	0.85	6	V	XT17	geom	0.85	$\{3,5,7\}$	V
T18	unif	0.85	8	V	XT18	geom	0.85	$\{3,4,5\}$	V
T19	unif	0.90	4	V	XT19	geom	0.90	$\{4,6,8\}$	V
T20	unif	0.90	6	V	XT20	geom	0.90	$\{3,5,7\}$	V
T21	unif	0.90	8	V	XT21	geom	0.90	$\{3,4,5\}$	V
T22	unif	0.95	4	V	XT22	geom	0.95	$\{4,6,8\}$	V
T23	unif	0.95	6	V	XT23	geom	0.95	$\{3,5,7\}$	V
T24	unif	0.95	8	V	XT24	geom	0.95	$\{3,4,5\}$	V

all four scenarios, the processing times follow a Uniform($1, 49$) distribution ($\mu = 25$). The four scenarios give two utilisation levels, 85% and 95%, and either "full" (one operation at each machine, giving ten operations per job) or "variable" operations (the number of operations per job is uniformly distributed on $\{2, \ldots, 10\}$). Jobs are given weight 1, 2 or 4, with probability $(0.2, 0.6, 0.2)$. This setting has been previously used [94] and originates from research by Pinedo and Singer [111] that showed that approximately 20% of jobs are of low importance, 60% are of average importance and the final 20% are very important, so weights of 1, 2 and 4 are assigned to jobs following these probabilities. The due date tightness parameter, h, for each job is chosen from $\{3, 5, 7\}$ with equal probability.

Testing

We use the ten-machine job shop of [117] for testing. As in training, the processing time distribution is Uniform($1, 49$) and jobs are given weight 1, 2 or 4, with probability $(0.2, 0.6, 0.2)$ [111]. There are three due date tightness parameters (4, 6, and 8), four machine utilisations (80%, 85%, 90%, and 95%) and either "full" or "variable" operations. This gives a total of 24 test scenarios which are given in Table 5.1.

Extreme Testing

In order to further test the generalisation ability of the best-of-run evolved dispatching rules, we have an additional set of extreme test instances. We change the distribution of processing times to a geometric distribution with mean $\mu = 25$ (parameter $p = 0.04$). The geometric distribution is chosen because it is a discrete distribution, different from the distribution which was used in training. We also change the weights given to jobs, including an additional weight for *extremely* important jobs; jobs are now given weight 1, 2, 4 or 8, with probability $(0.2, 0.5, 0.2, 0.1)$. If jobs with extremely high importance are not correctly scheduled, the penalty is much higher, so this checks the ability of rules to take job weight into account. The due date tightness parameter is equally likely from $\{4, 6, 8\}$, $\{3, 5, 7\}$, or $\{3, 4, 5\}$ depending on the scenario. This includes some tighter due dates than in training, making the problem instances more difficult as it will be even more important to schedule jobs efficiently. There are a total of 24 extreme test scenarios which are given in Table 5.1.

Within each training simulation a GP individual (dispatching rule) is used to evaluate queues and assign priorities many times, e.g., in a "full" scenario, 2000 jobs $\times$ 10 machines = 20,000 selections (some from queues of size one). The simulation process is time consuming, therefore a small number of training problem instances is used. When it comes to testing, we wish to see how well the evolved best-of-run dispatching rules perform across a wide range of problem instances, including problem instances with unseen distributions, thus testing the robustness of evolved dispatching rules.

5.2.3 Function and Terminal Sets

The attributes of the jobs, machines and the shop that are used as terminals in the GP system are the same as used by LMGP in Chapter 4 (see Table 4.1 on page 108).

The function set consists of a subset of the following functions $\{+, -, *, \%,$ `if>0`, `max`, `min`, `ifPS`, `ifOD`, `ifLO`$\}$, dependent on which grammar is being used. The arithmetic operators take two arguments. The first three arithmetic operators, $+$, $-$ and $*$, have their usual meanings. The $\%$ operator is protected division, returning one if dividing by zero. The `max` and `min` functions take two arguments and return the maximum and minimum of their arguments respectively. The `if>0` function takes three arguments; if the first argument is greater than 0 then it returns the second, else the third is returned. The remaining three `if` operators are introduced in Section 5.3.3 (see page 187).

We will consider using "standard" GP where there is no restriction on which terminals can be arguments for functions, and a version of GP which is restricted by a grammar, implemented using the Strongly Typed GP [85] capabilities of ECJ20 [78]. We refer to these methods as GP and STGP respectively. Both methods use the same set of terminals (presented in Table 4.1 on page 108) and functions.

This is the first investigation using a grammar to semantically constrain the search space in DR generation at the level of basic properties (as opposed to the approach of [95] which uses simple existing DRs). We have created different grammars based on partitioning the terminal set into four different type categories. One type represents counts: `NQ`, `RO`, `NNQ`. The second type consists of the job weight: `W`. The third type represents time durations: `PR`, `RT`, `QW`, `NPR`, `AQW`, `NQW`. The final type represents absolute (clock) times: `DD`, `RM`, `RJ` and `CT`.

The basis for creating these grammars has been to prohibit some interactions, and to begin by restricting the types to those of the input terminals only. This means that, e.g., there are no squared time units. We hope this will reveal information about what useful elements are in effective DRs. The allowable and restricted interactions are shown in Table 5.2. Black table cells show that the interaction is not allowed, e.g., `D*D`, and for interactions that are allowed the resulting type is given, e.g., `X*D` returns type `D`. Some of the interactions we restrict include adding or subtracting terminals that are a weight or a count to a time or a duration, which does not make sense (or certainly does not make *as much sense* as multiplying or dividing). We have been very harsh in limiting what time terminals can be divided and multiplied by. The `if>0` function can take any first argument, but the second and third arguments must be of the same type (and this is the type returned).

5.2.4 GP Parameter Settings

The initial population is generated using the ramped-half-and-half method [70] and has an initial maximum depth of five. The population size is

Table 5.2: Allowable interactions for each operator in $\{*, \%, +, -, \max, \min\}$ for Grammar 1. Where interaction is allowed the type of the result is given, and the cell is coloured black if the interaction is not allowed. Terminals are split into four different type categories: C represents counts, X represents job weight (and other ratios), D represents time durations and T represents absolute (clock) times.

$*$	X	C	D	T
X	X	X	D	■
C	X	C	D	■
D	D	D	■	■
T	■	■	■	■

%	X	C	D	T
X	X	X	■	■
C	X	X	■	■
D	D	D	X	X
T	■	■	X	X

$+$	X	C	D	T
X	X	■	■	■
C	■	■	■	■
D	■	■	D	T
T	■	■	T	■

$-$	X	C	D	T
X	X	■	■	■
C	■	■	■	■
D	■	■	D	T
T	■	■	T	D

min	X	C	D	T
X	X	■	■	■
C	■	C	■	■
D	■	■	D	■
T	■	■	■	T

max	X	C	D	T
X	X	■	■	■
C	■	C	■	■
D	■	■	D	■
T	■	■	■	T

1024 and evolution is for 50 generations. GP trees have a maximum tree depth of six. This is a smaller maximum tree depth than is often used, since with the increase in tree size interpretability is expected to decrease. For the genetic operators crossover, mutation and elitism, we use rates of 85%, 10% and 5% respectively. Tournament selection with a tournament size of seven is used to select individuals for genetic operators. These are common parameter settings that have been previously used [95].

5.3 Experimental Results

Here we present the results of using GP and STGP, working through iterations of additions to the function and terminal sets. We begin in Stage 1 with a base grammar, using only arithmetic functions. In Stage 2 we modify the base grammar to include `if>0`, `max` and `min` in the function set. In the third stage we remove `if>0` from the function set, replacing it with three specialised conditional statements based on analysis of the evolved rules from Stage 2. These modifications will be explained further in Sections 5.3.2 and 5.3.3. Additionally at each stage we investigate multiple grammars, introducing additional types to see if performance improves.

For all methods we perform 50 independent evolutionary runs, using 50 common pseudo-random number generator seeds, in order that we can perform statistical testing. Tables 5.5, 5.6, 5.9, 5.10, 5.11 and 5.12 give the mean and standard deviation of TWT for one problem instance per scenario. We use only one problem instance for each scenario as within a single problem instance a DR is called many times. When a method is statistically significantly better than all other methods the value is shown in bold in the tables. If two methods are not statistically significantly different from each other but both are statistically significantly better than the remaining methods, both values are shown in bold. For example, on problem scenario T15, WCOVERT attained a TWT value of 0.8 and GP1 (standard GP with a reduced function set of only four terminals, introduced at Stage 1) and GP2 (standard GP with `if>0`, `max` and `min` added to the function set of GP1, introduced at Stage 2) were not statistically different to each other, with mean performance across the 50 GP runs of 78.9±209.7 (from Table 5.5 see page 177) and 78.9±173.2 (from Table 5.9 see page 185) respectively.

The performance of the 50 best-of-run evolved DRs of each method is shown using violin plots. Each scenario's graph consists of a violin plot for each method, with individual values attained overlaid. The vertical axis is the TWT in 1000s, and the scale of the axis is different for each scenario's plot. These violin plots show us the *distribution* of TWT attained by evolved DRs from each method; this includes more information than can be gained by looking at the means and standard deviations in Tables 5.5, 5.6, 5.9, 5.10, 5.11 and 5.12.

We are interested in quantifying any performance loss by the use of STGP. To do this we obtain the average normalised TWT fitness value across the test and extreme test scenarios of each best-of-run dispatching rule (as we do to obtain a single fitness value in training). We then average across all best-of-run rules from each method, obtaining values m_{GP1} for method GP1, m_{ST1} for method ST1 (STGP with a reduced function set of only four terminals, and initial grammar introduced at Stage 1), etc. We then calculate values comparing each GP method to its STGP counterparts:

$$m_{GP,ST} = \frac{m_{ST} - m_{GP}}{m_{GP}}.$$

5.3.1 Stage One: Arithmetic GP and STGP

First we use the grammar in Table 5.3 to restrict STGP. This uses a reduced function set of only the four arithmetic operators $\{+, -, *, \%\}$. We compare rules evolved under this grammar to GP with the same arithmetic function set. We will call these two methods GP1 and ST1. We start with this arithmetic function set as DRs with these operators are most straightforward to understand. This grammar expresses the allowable interactions presented

Table 5.3: Grammar 1 for STGP. This is the grammar based representation of the allowable interactions for the arithmetic operators and four basic types presented in Table 5.2. The four types representing the initial terminal set, X, C, D and T are used. Any interaction which returns type I is not allowable in this basic grammar.

S =	<A>		
N =	{A, X, C, T, D}		
Σ =	{∗, %, −, +, PR, RT, RO, RJ, DD, W, RM, NQ,		
	QW, CT, NPR, NNQ, AQW, NQW}		
P =	{<A>	::=	<X> | <C> | <D> | <T>
	<X>	::=	(W) | (+ <X> <X>) | (− <X> <X>)
			| (∗ <X> <X>) | (% <X> <X>)
			| (% <C> <C>) | (% <C> <X>)
			| (% <X> <C>)| (∗ <C> <X>)
			| (∗ < X > <C>) | (% <D> <D>)
			| (% <T> <D>) | (% <T> <T>)
			| (% <D> <T>)
	<C>	::=	(RO) | (NQ) | (NNQ) | (∗ <C> <C>)
	<D>	::=	(PR) | (RT) | (QW) | (NPR) | (NQW) | (AQW)
			| (+ <D> <D>) | (− <D> <D>)
			| (− <T> <T>)
			| (∗ <D> <X>) | (∗ <X> <D>)
			| (∗ <C> <D>) | (∗ <D> <C>)
			| (% <D> <C>) | (% <D> <X>)
	<T>	::=	(DD) | (CT) | (RJ) | (RM)
			| (+ <D> <T>) | (+ <T> <D>)
			| (− <T> <D>) | (− <D> <T>) }

earlier in Table 5.2. This grammar is a basic grammar, restricting interactions that definitely do not make sense, as well as those that only make dubious sense. There are many other grammars that could be defined, with varying degrees of harshness of restriction; the time taken to explore these would be extensive. Therefore this is a basic starting point for exploring what are useful elements within a semantically correct search space.

Initial results of GP1 and ST1 are shown in the first two violin plots: Figure 5.1 (test scenarios T1 to T24) and Figure 5.2 (extreme test scenarios XT1 to XT24). Tables 5.5 and 5.6 show that DRs evolved under ST1 do not perform as well as those evolved under GP1. We performed a paired Mann Whitney U Test which showed that the difference of mean TWT is statistically significant (GP1 with lower mean TWT at the 5% significance level) on 35/48 test and extreme test scenarios. We do not expect ST1 to attain the same level of performance as it is searching a heavily restricted search space. In general the results in Figures 5.1 and 5.2 show approximately the same shape for lower TWT values, but ST1 has a much longer tail (worse TWT performance) particularly on extreme test scenarios. We wonder if this could be due to the grammar not allowing interactions which give inverse durations, e.g., $1/PR$ or W/PR, and squared durations, e.g., $RT \times PR$.

Inclusion of Inverse-Duration Type

To investigate this premise we introduce a fifth type, I, to represent inverse-durations. The allowable interactions are shown in Table 5.4. Due to the implementation of strong typing in ECJ, to allow this additional type we need to define a terminal of type I. Firstly we add in a new terminal "invPR" which is $1/PR$ (we know PR is always > 0), and call this ST1A. Secondly, to the original terminal set we add in a new terminal "W/PR", which returns W/PR, since this, by itself, is a common rule, and is also a component of many of the successful dispatching rules from the literature for the dynamic JSS problem with TWT objective, in particular ATC and

Table 5.4: Allowable interactions for each operator in $\{*, \%, +, -, \max, \min\}$ for Grammar 1A. Where interaction is allowed the type of the result is given, and the cell is coloured black if the interaction is not allowed. Terminals are split into five different type categories: C, X, D, T, and new type I which represents inverse time durations.

$*$	X	C	D	T	I
X	X	X	D	■	I
C	X	C	D	■	I
D	D	D	■	■	X
T	■	■	■	■	X
I	I	I	X	X	■

%	X	C	D	T	I
X	X	X	I	■	D
C	X	X	I	■	D
D	D	D	X	X	■
T	■	■	X	X	■
I	I	I	■	■	X

$+$	X	C	D	T	I
X	X	■	■	■	■
C	■	■	■	■	■
D	■	■	D	T	■
T	■	■	T	■	■
I	■	■	■	■	I

$-$	X	C	D	T	I
X	X	■	■	■	■
C	■	■	■	■	■
D	■	■	D	T	■
T	■	■	T	D	■
I	■	■	■	■	I

min	X	C	D	T	I
X	X	■	■	■	■
C	■	C	■	■	■
D	■	■	D	■	■
T	■	■	■	T	■
I	■	■	■	■	I

max	X	C	D	T	I
X	X	■	■	■	■
C	■	C	■	■	■
D	■	■	D	■	■
T	■	■	■	T	■
I	■	■	■	■	I

WCOVERT described in Section 2.2.3 (see page 42). We call this ST1B. Of ST1, ST1A and ST1B, the best results are given by ST1A with statistically significantly lower means than ST1 on 3/24 test instances and 6/24 extreme test instances. GP1 has statistically significantly lower means than ST1A on 25/48, only two less than compared to ST1. There was no statistically significant difference of means at 5% between ST1 and ST1A, or between ST1 and ST1B. The performance of these two methods in comparison to ST1 did not convince us that it made a difference either in improving or hindering the ability of GP to find effective dispatching rules. Table 5.5 shows that the mean performance of GP1, ST1, ST1A and ST1B all have large standard deviations.

The performance decrease using the measure described above, gives values of $m_{GP1,ST1} = 0.4967$, $m_{GP1,ST1A} = 0.5371$, and $m_{GP1,ST1B} = 0.5891$. These values show that the three ST methods (ST1, ST1A and ST1B) all have approximately 50% greater TWT than GP1.

Inclusion of Square-Duration Type

The performance of the ST methods still do not quite match the performance of GP. To investigate this premise we introduce a sixth type, S, to represent squared-durations. The allowable interactions are shown in Table 5.7. Again, due to the implementation of strong typing in ECJ, we need to define a terminal of type S. We add in a new terminal "sqPR" which is $[PR * PR]$ (we know PR is always > 0), and call this ST1S.

Figure 5.1 shows that ST1S often has a shorter tail of worse TWT values than the ST1A and ST1B methods, and is more similar in distribution to ST1. Figure 5.2, on extreme test scenarios, shows less difference in the distribution of attained TWT.

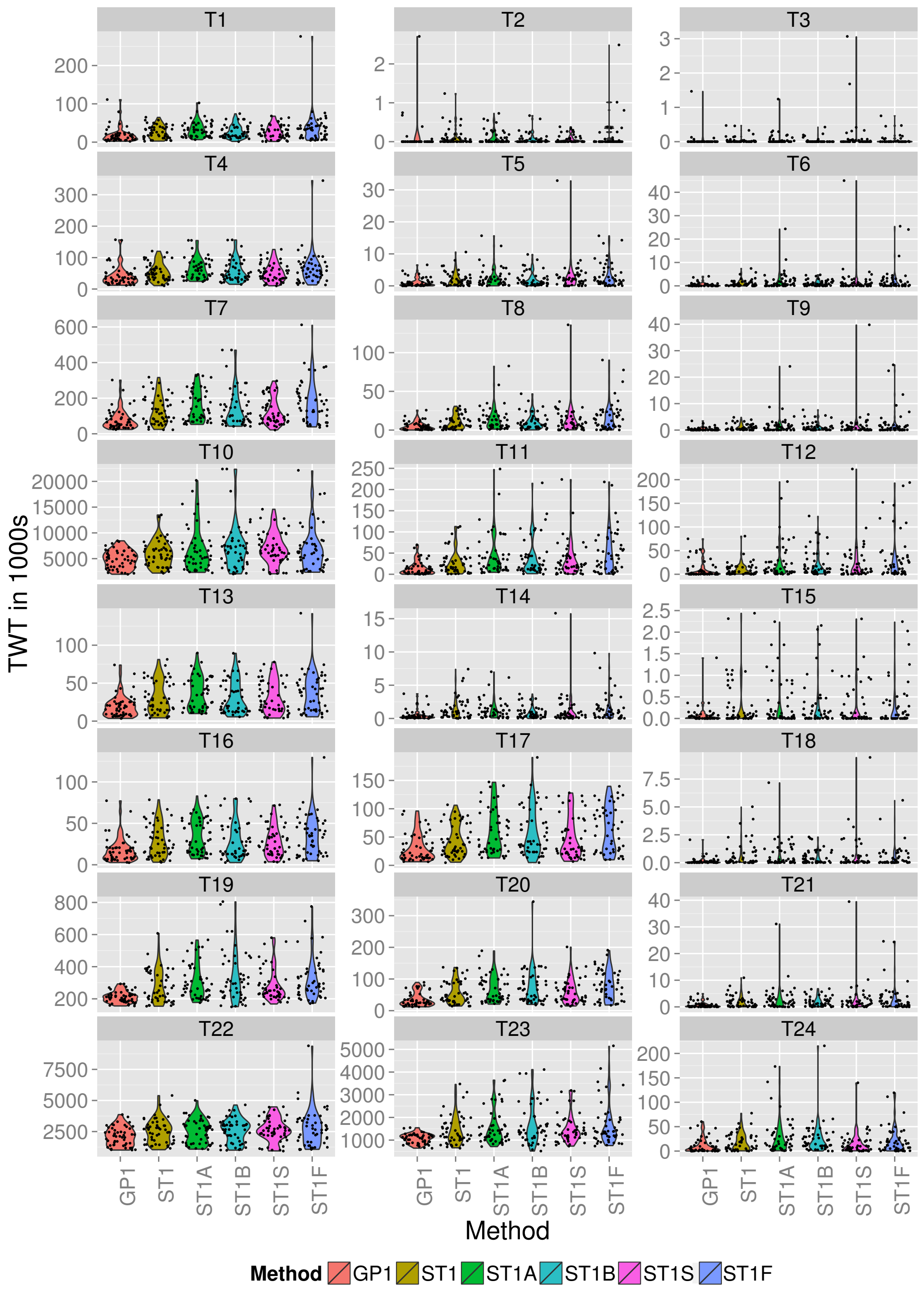


Figure 5.1: Violin plots presenting results (TWT in 1000s) of test scenarios for Stage 1 GP and STGP methods. Note that the vertical axis scales are different in each subfigure. These figures shows the distribution of TWT values attained from the 50 independent GP runs of each method.

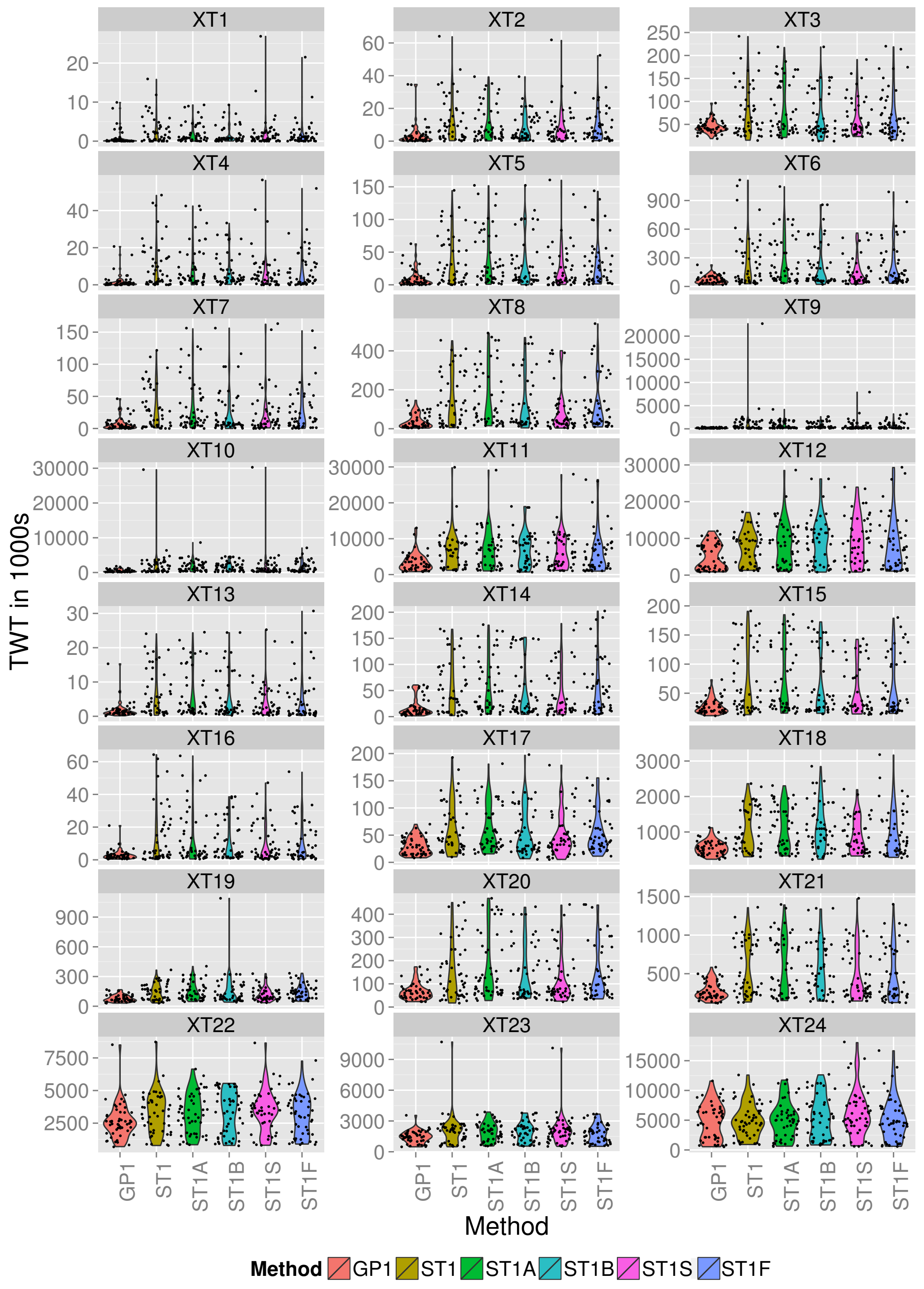


Figure 5.2: Violin plots presenting results (TWT in 1000s) of extreme test scenarios for Stage 1 GP and STGP methods. Note that the vertical axis scales are different in each subfigure. These figures shows the distribution of TWT values attained from the 50 independent GP runs of each method.

Table 5.5: Mean and standard deviation of total weighted tardiness on test problem instances of Stage 1 (lower values are better). Values that are statistically significantly better than all other GP and STGP methods across the same row across Tables 5.5, 5.9 and 5.11 are in **bold**. Where there are two values in a row that are bold, they are not statistically significantly different from each other, but are statistically significantly better than all remaining methods.

	Benchmark DRs		GP1	ST1	ST1A	ST1B	ST1S	ST1I
	WCOVERT	WATC	Mean±Stdev	Mean±Stdev	Mean±Stdev	Mean±Stdev	Mean±Stdev	Mean±Stdev
T1	74607.8	22613.2	**18620.4 ± 19703.0**	26531.7 ± 16482.1	32871.4 ± 20068.6	26830.4 ± 16839.7	36506.9 ± 39874.6	26690.1 ± 17540.9
T2	0.0	0.0	94.5 ± 406	105.3 ± 223.6	117.1 ± 185.6	62.0 ± 137.5	122.9 ± 395.2	56.5 ± 101.4
T3	0.0	0.0	35.6 ± 208.1	45.5 ± 107.2	64.9 ± 188.4	25.1 ± 77.0	49.1 ± 140.5	130.8 ± 491.8
T4	121131.0	38719.9	**40592.2 ± 31286.7**	53044.0 ± 28063.6	63303.5 ± 31973.0	57686.6 ± 34853.3	65232.4 ± 49876.8	51771.8 ± 28360.7
T5	10343.9	0.0	1027.1 ± 1399.6	2038.0 ± 2241.2	2571.8 ± 3035.3	1932.6 ± 2226.8	3201.8 ± 3800.4	2854.7 ± 4737.6
T6	939.0	0.0	483.0 ± 872.1	1158.1 ± 1545.2	2098.4 ± 3859.9	1095.9 ± 1242.2	2399.6 ± 5095.5	2116.4 ± 6394.3
T7	259278.7	155171.7	**77953.9 ± 56282.9**	128149.8 ± 77893.8	154828.3 ± 86266.8	143618.3 ± 104276.2	172173.2 ± 118410.0	123008.1 ± 73710.0
T8	72425.9	7782.9	4911.5 ± 5414.0	10150.5 ± 8825.8	14127.7 ± 14980.0	10617.5 ± 10107.6	17088.1 ± 18991.7	13205.1 ± 20079.6
T9	903.6	0.0	379.4 ± 679.3	1086.5 ± 1175.0	1869.6 ± 3701.1	1218.5 ± 1698.4	2432.8 ± 5027.3	1910.1 ± 5641.8
T10	1840851.3	1981855.2	4858750.6 ± 1757378.7	5966859.9 ± 2514085.4	6756229.2 ± 4247084.1	7188453.1 ± 4444147.6	6949611.5 ± 4294712.5	6938116 ± 2984721.6
T11	208240.8	123375.3	13927.2 ± 16400.7	27067.2 ± 27683.3	44833.5 ± 49257.9	40448.3 ± 44902.6	50845.8 ± 50247.8	33863.3 ± 39724.8
T12	189379.0	86525.7	9898.2 ± 17421.7	10008.1 ± 13762.9	25239.0 ± 39673.6	21625.3 ± 27940.5	36445.3 ± 49706.2	22181.0 ± 36523.7
T13	76278.6	20238.7	**19170.5 ± 14379.1**	30442.6 ± 20745.2	35971.2 ± 21358.5	30854.2 ± 21790.8	36901.3 ± 25574.3	30355.6 ± 20855.3
T14	1159.5	54.5	499.9 ± 780.9	1393.9 ± 1604.7	1352.3 ± 1497.7	863.6 ± 899.2	1419.1 ± 1998.6	1213.1 ± 2327.6
T15	0.8	3.6	**78.9 ± 209.7**	319.5 ± 555.9	345.3 ± 539.7	304.2 ± 526.8	298.6 ± 530.2	267.8 ± 474.4
T16	77436.5	20684.0	**17528.0 ± 14731.9**	28248.2 ± 19595.9	34451.5 ± 20482.5	28905.9 ± 21859.0	34714.2 ± 23907.8	27289.7 ± 18308.5
T17	100027.6	78622.3	30245.6 ± 23444.2	44064.9 ± 27911.2	60446.8 ± 40782.4	58269.9 ± 41473.5	62031.8 ± 39980.4	47212.1 ± 35941.0
T18	82.8	0.0	144.0 ± 321.1	698.5 ± 1112.2	698.5 ± 1185.0	455.5 ± 605.4	555.0 ± 1003.3	686.6 ± 1542.2
T19	306873.7	264216.6	**211110.5 ± 34468.8**	280645.5 ± 103866.7	306162.1 ± 108975.8	330419.2 ± 153391.4	322272.2 ± 131365.8	282839.1 ± 99445.2
T20	148573.6	102920.4	34813.2 ± 21809.5	55209.7 ± 35560.7	67531.3 ± 41671.7	72998.3 ± 56150.0	77991.8 ± 45783.2	59348.5 ± 38620.3
T21	148.1	26.7	**702.4 ± 1070.4**	1956.4 ± 2279.6	2877.8 ± 4722.3	1856.1 ± 1836.0	3128.1 ± 5144.7	2587.0 ± 5780.1
T22	970487.9	1022018.9	2131306.6 ± 742620.5	2498801.9 ± 943851.3	2504233.1 ± 951527.7	2554825.1 ± 937928.6	2758445.0 ± 1483709.4	2629431.6 ± 916464.1
T23	1019714.7	901866.7	1067595.6 ± 211617.1	1432053.8 ± 629594.9	1551080.4 ± 734545.9	1673677.0 ± 891556.3	1647614.7 ± 932138.0	1486955.5 ± 613816.6
T24	47578.3	12591.1	**10328.3 ± 15270.5**	18193 ± 18026.2	28139.5 ± 35835.6	25178.8 ± 32389.9	20679.2 ± 26343.4	14016.5 ± 22313.1

Table 5.6: Mean and standard deviation of total weighted tardiness on extreme test problem instances of Stage 1 (lower values are better). Values that are statistically significantly better than all others across the same row across Tables 5.6, 5.10 and 5.12 are in **bold**. Where there are two values in a row that are bold, they are not statistically significantly different from each other, but are statistically significantly better than all remaining methods.

	Benchmark DRs		GP1	ST1	ST1A	ST1B	ST1S	ST1I
	WCOVERT	ATC	Mean±Stdev	Mean±Stdev	Mean±Stdev	Mean±Stdev	Mean±Stdev	Mean±Stdev
XT1	1495.1	0.0	**839.6 ± 1992.2**	2626.7 ± 3402.2	2020.3 ± 2519.2	1533.3 ± 2187.1	1788.9 ± 3464.3	2073.2 ± 4199.4
XT2	41436.2	15270.9	**3845.9 ± 6936.7**	12454.5 ± 13604.4	10579.7 ± 11039.8	8109.6 ± 8335.9	9296.9 ± 10294.3	8654.8 ± 10945.9
XT3	1625.5	0.0	44208.1 ± 13438.2	77340.6 ± 55876.0	83068.1 ± 56806.9	65983.0 ± 48828.0	70484.0 ± 52047.6	62144.5 ± 40200.3
XT4	48998.5	5117.2	**1837.4 ± 3792.7**	10500.8 ± 12406.8	10482.9 ± 11787.8	7894.8 ± 8925.6	7823.1 ± 10233.4	7308.7 ± 10708.7
XT5	127840.5	60816.7	**9236.6 ± 12572.0**	37711.0 ± 38678.0	39982.9 ± 42297.6	31824.1 ± 38007.7	33288.1 ± 36635.9	25180.5 ± 33427.7
XT6	5302.1	42.6	**70475.7 ± 41438.1**	260283.5 ± 271169.8	257269.7 ± 240524.6	216565.6 ± 224909.7	216518.8 ± 224621.4	152497.0 ± 146436.9
XT7	113330.7	40819.4	**6802.7 ± 9533.5**	30570.1 ± 33198.0	34326.3 ± 39982.9	25104.5 ± 34560.9	26191.6 ± 32153.9	22439.0 ± 33993.7
XT8	298810.2	215212.5	**37050.8 ± 35976.1**	146640.0 ± 135460.9	157319.1 ± 149736.7	127298.0 ± 140196.5	123544.9 ± 130467.8	90845.2 ± 111127.7
XT9	778883.6	894735.1	**186669.1 ± 94106.5**	1324549.1 ± 3209179.8	852515.4 ± 816559.7	719777.2 ± 682262.0	699526.5 ± 789504.2	671247.2 ± 1226609.5
XT10	916183.6	1035644.4	**457585.2 ± 387358.4**	2278006.9 ± 4217681.4	1815475.2 ± 1610225.2	1810345.5 ± 1357920.8	1578081.6 ± 1580341.1	1711576.4 ± 4295866.0
XT11	865992.1	745914.2	3425155.5 ± 2498451.7	6597318.4 ± 5262220.7	6744636.5 ± 5211048.4	6764407.2 ± 4727205.3	6136085.6 ± 5964578.2	6268085.5 ± 5016671.8
XT12	680.7	460.0	4846596.6 ± 3537491.8	7327962.9 ± 4750244.5	8249627.7 ± 5954000.6	9139143.9 ± 6334936.3	8057444.2 ± 7252974.9	9211463.1 ± 6280280.9
XT13	26151.0	15806.0	**1727.7 ± 2285.2**	6550.4 ± 6701.5	6679.5 ± 7141.9	5614.5 ± 6556.0	5126.5 ± 6962.6	4246.5 ± 5110.1
XT14	38192.1	13676.2	**17684.9 ± 17318.6**	56405.3 ± 50496.3	56424.3 ± 51620.8	48782.4 ± 51662.4	53084.2 ± 52821.3	41137.4 ± 45668.4
XT15	1824.9	1247.7	**27490.9 ± 12945.8**	69746.4 ± 56469.6	69237.7 ± 53841.9	61891.5 ± 53771.9	58820.7 ± 49359.4	51252.4 ± 42013.0
XT16	27222.1	14307.9	**2929.1 ± 3178.0**	13445.0 ± 16971.1	12571.4 ± 16056.8	10507.8 ± 11786.6	9710.1 ± 11711.5	8854.4 ± 11256.5
XT17	271583.7	263100.7	**29152.5 ± 15765.6**	61071.6 ± 45194.0	60208.0 ± 40680.3	53975.2 ± 42656.7	54117.5 ± 37102.5	50093.2 ± 38313.2
XT18	163517.5	80359.2	556743.8 ± 191343.2	1079218.6 ± 606650.9	1089129.9 ± 602294.1	1117038.2 ± 646134.5	978932.4 ± 650993.9	890752.8 ± 496159.0
XT19	91101.9	72758.9	**79232.9 ± 33045.5**	140976.3 ± 81261.6	151145.7 ± 88299.6	155978.2 ± 160220.8	143692.1 ± 73654.0	118789.2 ± 65294.2
XT20	133643.9	120456.0	**66293.2 ± 34462.6**	150761.6 ± 122068.2	166120.9 ± 136636.5	140002.9 ± 119349.1	140127.3 ± 116693.5	117101.0 ± 100408.0
XT21	570388.5	581547.7	275727.1 ± 112163.7	576309.2 ± 357192.5	614901.0 ± 376547.7	566211.1 ± 343913.8	480764.4 ± 343732.9	458997.0 ± 324733.4
XT22	454536.9	420731.2	2638620.4 ± 1292992.8	3278079.5 ± 1591076.1	3240274.4 ± 1510740.7	3286780.7 ± 1631151.2	3087629.8 ± 1557426.0	3404463.1 ± 1589795.9
XT23	428607.7	432883.8	1436336.5 ± 583534.5	2047652.0 ± 1519999.8	1955078.8 ± 891569.5	1909128.7 ± 945196.0	1773266.2 ± 888548.6	2030606.0 ± 1423460.5
XT24	227787.8	205629.2	4854042.7 ± 2987829.3	4871705.4 ± 2764775.9	5016157.9 ± 3030008.2	5223200.3 ± 3561380.0	5120407.2 ± 3588513.3	6047566.3 ± 3918527.4

Inclusion of Inverse-Square-Duration Type

The final type we introduce is inverse square durations, type F, and we call this ST1F. We add new terminal $[1/[PR * PR]]$. The allowable interactions are shown in Table 5.8. Table 5.5 shows that ST1F has lower means than the other ST methods on XT13 to XT21, but similar on the other scenarios.

The performance decrease using the measure described above, gives values of $m_{GP1,ST1S} = 0.5716$ and $m_{GP1,ST1F} = 0.4901$. This shows that the best performance of STGP relative to GP is from ST1F, followed by ST1 and then ST1A. This is interesting that the difference between ST1F and ST1 is small (0.0036 difference in performance measures) despite the addition of two additional types. It is also interesting that the grammars in between (ST1A, ST1B and ST1S) are not as effective. Based on the ordering generated by the performance measure the best of the Stage 1 grammars that we recommend to use are ST1 and ST1F.

5.3.2 Stage Two: Inclusion of If, Max and Min

We introduce `if>0`, `max` and `min` into the function set of the second grammar. The allowable interactions are shown in Table 5.2. The first argument, the condition, of the `if>0` function can potentially be a large or complex expression for which it is difficult to understand why different behaviour should happen based on the outcome of the given expression. Again we compare this to standard GP with the same function set. We will call these two methods GP2 and ST2. Following Stage 1, the additional types I, S and F are also added incrementally, giving us ST2A, ST2S and ST2F respectively.

Results from the best-of-run DRs from GP2 and ST2 are shown in the violin plots of Figures 5.3 and 5.4 and in Tables 5.9 and 5.10. These show that DRs evolved under ST2 do not perform as well as those evolved under GP2, with a greater number of DRs attaining high TWT values.

In the evolved best-of-run DRs from GP2 there are 81 `if>0` statements

Table 5.7: Allowable interactions for each operator in $\{\times, \%, +, -, \max, \min\}$ for Grammar 1S. Where interaction is allowed the type of the result is given, and the cell is coloured black if the interaction is not allowed. Terminals are split into six different type categories: C, X, D, T, I, and new type S which represents squared time durations.

$\times$	X	C	D	T	I	S
X	X	X	D	■	I	S
C	X	C	D	■	I	S
D	D	D	S	■	X	■
T	■	■	■	■	X	■
I	I	I	X	X	■	D
S	S	S	■	■	D	■

%	X	C	D	T	I	S
X	X	X	I	■	D	■
C	X	X	I	■	D	■
D	D	D	X	X	S	I
T	■	■	X	X	■	I
I	I	I	■	■	X	■
S	S	S	D	D	■	X

$+$	X	C	D	T	I	S
X	X	■	■	■	■	■
C	■	C	■	■	■	■
D	■	■	D	T	■	■
T	■	■	T	T	■	■
I	■	■	■	■	I	■
S	■	■	■	■	■	S

$-$	X	C	D	T	I	S
X	X	■	■	■	■	■
C	■	C	■	■	■	■
D	■	■	D	T	■	■
T	■	■	T	D	■	■
I	■	■	■	■	I	■
S	■	■	■	■	■	S

min	X	C	D	T	I	S
X	X	■	■	■	■	■
C	■	C	■	■	■	■
D	■	■	D	■	■	■
T	■	■	■	T	■	■
I	■	■	■	■	I	■
S	■	■	■	■	■	S

max	X	C	D	T	I	S
X	X	■	■	■	■	■
C	■	C	■	■	■	■
D	■	■	D	■	■	■
T	■	■	■	T	■	■
I	■	■	■	■	I	■
S	■	■	■	■	■	S

Table 5.8: Allowable interactions for each operator in $\{\times, \%, +, -, \max, \min\}$ for Grammar 1F. Where interaction is allowed the type of the result is given, and the cell is coloured black if the interaction is not allowed. Terminals are split into seven different type categories: C, X, D, T, I, S and new type F which represents inverse-square-durations.

×	X	C	D	T	I	S	F
X	X	X	D	■	I	S	F
C	X	C	D	■	I	S	F
D	D	D	S	■	X	■	I
T	■	■	■	■	X	■	I
I	I	I	X	X	F	D	■
S	F	F	■	■	D	■	X
F	F	F	I	I	■	X	■

%	X	C	D	T	I	S	F
X	X	X	I	■	D	F	S
C	X	X	I	■	D	F	S
D	D	D	X	X	S	I	■
T	■	■	X	X	■	I	■
I	I	I	F	F	X	■	D
S	F	F	D	D	■	X	■
F	F	F	■	■	I	■	X

+	X	C	D	T	I	F	F
X	X	■	■	■	■	■	■
C	■	C	■	■	■	■	■
D	■	■	D	T	■	■	■
T	■	■	T	T	■	■	■
I	■	■	■	■	I	■	■
S	■	■	■	■	■	F	■
F	■	■	■	■	■	■	F

−	X	C	D	T	I	F	F
X	X	■	■	■	■	■	■
C	■	C	■	■	■	■	■
D	■	■	D	T	■	■	■
T	■	■	T	D	■	■	■
I	■	■	■	■	I	■	■
S	■	■	■	■	■	F	■
F	■	■	■	■	■	■	F

min	X	C	D	T	I	F	F
X	X	■	■	■	■	■	■
C	■	C	■	■	■	■	■
D	■	■	D	■	■	■	■
T	■	■	■	T	■	■	■
I	■	■	■	■	I	■	■
S	■	■	■	■	■	F	■
F	■	■	■	■	■	■	F

max	X	C	D	T	I	F	F
X	X	■	■	■	■	■	■
C	■	C	■	■	■	■	■
D	■	■	D	■	■	■	■
T	■	■	■	T	■	■	■
I	■	■	■	■	I	■	■
S	■	■	■	■	■	F	■
F	■	■	■	■	■	■	F

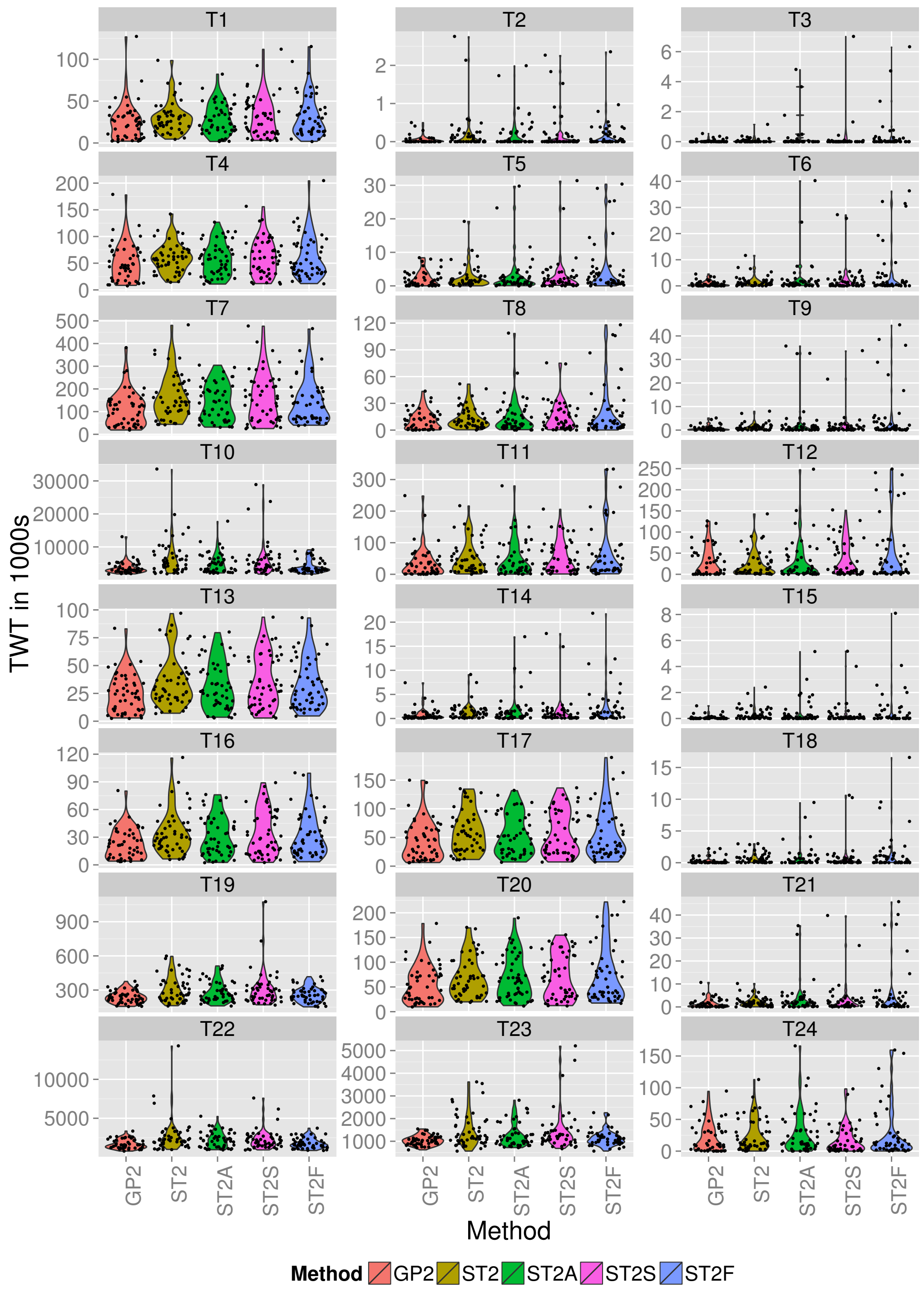


Figure 5.3: Violin plots presenting results (TWT in 1000s) of test scenarios for Stage 2 GP and STGP methods. Note that the vertical axis scales are different in each subfigure. These figures shows the distribution of TWT values attained from the 50 independent GP runs of each method.

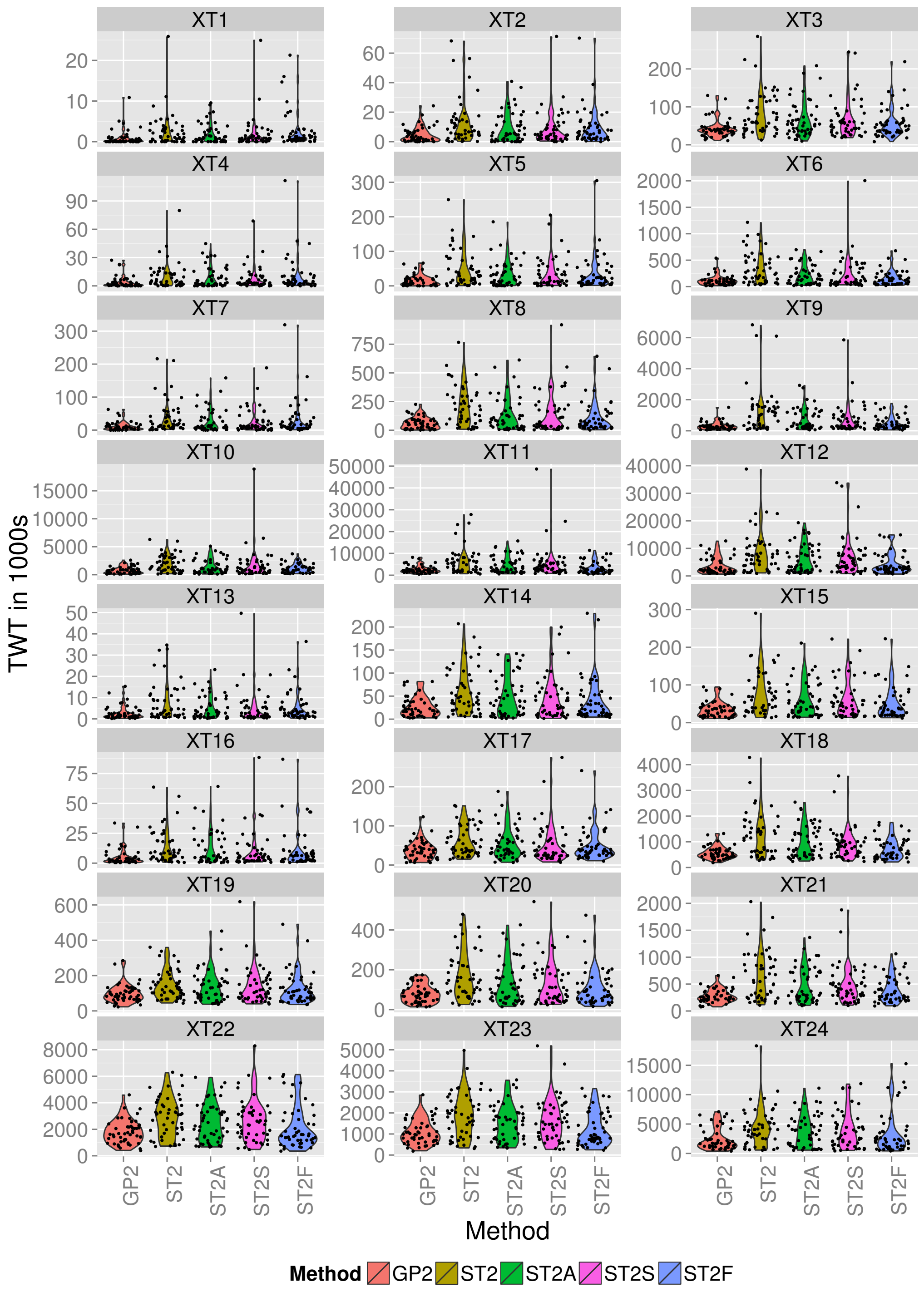


Figure 5.4: Violin plots presenting results (TWT in 1000s) of extreme test scenarios for Stage 2 GP and STGP methods. Note that the vertical axis scales are different in each subfigure. These figures shows the distribution of TWT values attained from the 50 independent GP runs of each method.

in the 50 best-of-run evolved DRs (see Table 5.14 on page 194). The evolved conditions (i.e. the first argument) of GP2 are frequently long, difficult to interpret, and/or it does not make sense to branch behaviour dependent on the condition value. Of the 81, 52 remain after simplification: 10 have complex first arguments, 21 have `NQW`, `NPR` or `NNQ`, 11 have `QW` or `AQW` (only very rarely equal to zero, and never negative), five `NQ−NNQ`, and five which are comparisons between the current time and the due date. This is a very high proportion of `if>0` statements which are not playing an active part in the DR, and only 34/50 of the best-of-run evolved DRs contain `if>0` statements. This caused us to question how useful the three argument `if>0` is as a function, and whether it could be better replaced by several `if` statements with predetermined conditions.

When we compare GP2 and ST2, there is a statistically significant difference of means at a significance level of 5% in favour of GP2 in 35/48 test instances. This decreased to 32/48 when we compared GP2 to ST2A (with the fifth inverse-duration type, and terminal `1/PR`), and when comparing ST2 to ST2A, the mean TWT was improved by ST2A on 13 test instances. In this case the inclusion of the inverse-duration type seems to improve mean performance more than it did with arithmetic operators only. Figures 5.3 and 5.4 show that the shape of all Stage 2 STGP methods are reasonably similar, although the height of the tail end (higher TWT values) often increases from ST2A to ST2S to ST2F.

At Stage 2 the performance decreases are $m_{GP2,ST2} = 1.0247$, $m_{GP2,ST2A} = 0.5765$, $m_{GP2,ST2S} = 0.7337$ and $m_{GP2,ST2F} = 0.2165$. This shows that ST2A has much better performance relative to GP2 than ST2. ST2 is *twice* as bad as GP2. The performance decrease from ST2 and ST2A to GP2 is worse than in our Stage 1 results. The best performance clearly from ST2F, which has TWT only 20% greater than GP2. However the performance decrease from GP2 to ST2F is the again the least, and there is less of a decrease in performance than at Stage 1.

Table 5.9: Mean and standard deviation of total weighted tardiness on test problem instances of Stage 2 (lower values are better). Values that are statistically significantly better than all others across the same row across Tables 5.5, 5.9 and 5.11 are in **bold**. Where there are two values in a row that are bold, they are not statistically significantly different from each other, but are statistically significantly better than all remaining methods.

	Benchmark DRs		GP2	ST2	ST2A	ST2S	ST2F
	WCOVERT	ATC	Mean±Stdev	Mean±Stdev	Mean±Stdev	Mean±Stdev	Mean±Stdev
T1	74607.8	22613.3	24773.8 ± 22258.5	29872.0 ± 18329.5	28579.0 ± 18402.5	32074.8 ± 25073.2	31646.2 ± 23876.6
T2	0.0	0.0	38.3 ± 102.2	201.1 ± 492.0	197.5 ± 405.7	168.2 ± 380.5	184.8 ± 471.7
T3	0.0	0.0	40.4 ± 107.4	68.3 ± 180.5	238.0 ± 873.7	367.3 ± 1155.3	184.3 ± 1004.8
T4	121131.0	38719.9	51095.2 ± 34739.9	59488.9 ± 26846.0	56587.0 ± 30169.7	59048.4 ± 38302.7	60219.3 ± 33905.6
T5	10343.8	0.0	2029.8 ± 2354.8	2411.9 ± 3349.8	2719.7 ± 5392.1	4753.6 ± 7602.9	2853.5 ± 5433.0
T6	939.0	0.0	793.2 ± 1054.5	1594.9 ± 2124.6	2602.2 ± 6596.4	4530.2 ± 9264.1	2343.2 ± 5248.8
T7	259278.7	155171.7	110939.2 ± 75835.9	162299.4 ± 97624.0	140111.1 ± 80244.7	143177.6 ± 97114.0	159580.1 ± 107440.2
T8	72425.9	7782.9	10550.6 ± 10608.4	13518.7 ± 12091.7	14481.7 ± 18783.8	21949.0 ± 29651.2	14886.3 ± 16348.5
T9	903.6	0.0	722.8 ± 1120.0	1291.7 ± 1611.2	3118.3 ± 7939.0	5017.1 ± 10519.3	2103.6 ± 5543.5
T10	1840851.3	1981855.2	3575100.3 ± 1823703.0	6728154.2 ± 5571493.1	5103776.2 ± 3043094.0	3937961.9 ± 1925720.7	5876119.7 ± 5321231.9
T11	208240.8	123375.3	35386.5 ± 47386.6	51122.1 ± 51552.4	48157.5 ± 56952.9	66809.1 ± 85957.8	49736.6 ± 50665.8
T12	189379.0	86525.7	28618.4 ± 35851.2	23050.3 ± 30567.2	29146.6 ± 46864.0	50253.1 ± 73503.8	34937.6 ± 41872.8
T13	76278.6	20238.7	**24442.2 ± 17160.1**	35856.3 ± 22507.5	31579.9 ± 21052.7	32708.1 ± 21839.4	34938.8 ± 23630.6
T14	1159.5	54.5	797.5 ± 1299.6	1455.8 ± 1694.3	1711.5 ± 3055.3	2340.8 ± 3980.1	1500.2 ± 3175.9
T15	0.8	3.6	**78.9 ± 173.2**	325.7 ± 490.1	435.4 ± 935.9	584.4 ± 1355.8	329.9 ± 925.9
T16	77436.5	20684.0	22614.7 ± 16238.4	34346.9 ± 22957.9	30470.5 ± 20548.8	31679.9 ± 23618.1	33685.5 ± 24295.6
T17	100027.6	78622.3	43970.6 ± 34054.4	60880.7 ± 35844.5	53878.1 ± 34819.1	56634.0 ± 43440.1	56385.5 ± 37901.9
T18	82.8	0.0	296.8 ± 549.1	672.9 ± 826.7	837.5 ± 1779.9	1377.6 ± 2983.8	717.4 ± 2051.1
T19	306873.7	264216.6	237696.1 ± 51878.6	303171.7 ± 110099.7	278882.5 ± 88635.0	258984.7 ± 63341.5	306000.9 ± 149731.6
T20	148573.6	102920.4	54789.4 ± 37518.0	68184.1 ± 39831.8	67862.9 ± 43310.8	72731.9 ± 55330.8	68903.3 ± 45098.8
T21	148.1	26.7	1413.8 ± 2022.3	2014.1 ± 2115.7	3410.4 ± 6617.1	5619.6 ± 10945.7	3073 ± 6687.3
T22	970487.0	1022018.9	**1670375.3 ± 583730.5**	2737968.8 ± 2141355.9	2207079.3 ± 995533.5	1783142.6 ± 660436.3	2266314.4 ± 1237565.2
T23	1019714.7	901866.7	1031699.8 ± 211353.9	1463156.2 ± 730838.7	1270494.5 ± 514338.2	1126598.8 ± 325922.7	1448254.8 ± 893233.5
T24	47578.2	12591.1	20647.9 ± 23605.4	24297.4 ± 25785.5	27585.0 ± 33560.2	25672.5 ± 39349.2	16880.2 ± 21626.2

Table 5.10: Mean and standard deviation of total weighted tardiness on extreme test problem instances of Stage 2 (lower values are better). Values that are statistically significantly better than all others across the same row across Tables 5.6, 5.10 and 5.12 are in **bold**. Where there are two values in a row that are bold, they are not statistically significantly different from each other, but are statistically significantly better than all remaining methods.

	Benchmark DRs		GP2	ST2	ST2A	ST2S	ST2F
	WCOVERT	ATC	Mean±Stdev	Mean±Stdev	Mean±Stdev	Mean±Stdev	Mean±Stdev
XT1	1495.1	0.0	**866.6 ± 1860.9**	2567.5 ± 4138.5	1871.3 ± 2484.3	2550.3 ± 4340.8	1790.2 ± 3815.4
XT2	41436.2	15270.9	4460.5 ± 5256.9	12898.6 ± 15092.6	9062.9 ± 10201.2	9277.3 ± 11779.9	8312.5 ± 11555.1
XT3	1625.5	0.0	**41921.0 ± 23271.2**	81944.7 ± 56086.2	65256.8 ± 46639.2	55825.5 ± 38905.3	65345.0 ± 49400.3
XT4	48998.5	5117.2	**3600.2 ± 5955.2**	10513.6 ± 13761.6	8771.0 ± 11069.4	9938.5 ± 18100.5	7481.7 ± 11732.0
XT5	127840.5	60816.7	15212.9 ± 15976.4	47192.4 ± 51427.8	32438.1 ± 37828.2	32577.5 ± 49230.2	34688.4 ± 44219.1
XT6	5302.1	42.6	109770.3 ± 94304.6	343919.3 ± 298896.8	212700.2 ± 185478.4	151913.6 ± 140049.3	231489.8 ± 316219.8
XT7	113330.7	40819.4	10380.9 ± 12995.6	38605.5 ± 48648.2	26378.9 ± 35020.0	29314.1 ± 51933.0	24216.6 ± 35905.6
XT8	298810.2	215212.5	59740.0 ± 51070.6	207873.4 ± 172959.0	133625.9 ± 148854.0	107664.5 ± 131314.8	138035.4 ± 176690.8
XT9	778883.6	894735.1	296941.0 ± 254735.9	1258959.8 ± 1473799.4	701621.1 ± 700386.6	460036.7 ± 413855.2	709782.0 ± 951839.0
XT10	916183.6	1035644.4	751358.5 ± 607389.0	1941517.7 ± 1547620.4	1353805.7 ± 1271681.8	1062420.1 ± 817602.0	1714809.2 ± 2833795.4
XT11	865992.1	745914.2	2600405.8 ± 1716181.6	6457207.6 ± 6029758.6	4721236.1 ± 3877225.7	3329157.4 ± 2749643.6	5639069.2 ± 7594930.9
XT12	680.7	460.0	3398849.1 ± 2996424.2	8201893.4 ± 7358834.2	6346130.1 ± 5067654.4	4239981.1 ± 3876590	6848188.2 ± 7017809.9
XT13	26151.0	15806.0	**2633.1 ± 3104.9**	7730.2 ± 9021.4	4956.0 ± 5486.3	5057.2 ± 6684.3	4864.4 ± 8117.4
XT14	38192.1	13676.2	**26547.0 ± 22032.3**	60088.8 ± 49727.9	45894.4 ± 43660.4	39725.9 ± 46791.7	44376.8 ± 46364.3
XT15	1824.9	1247.7	**33873.4 ± 20238.7**	77919.8 ± 57787.8	57418.5 ± 45119.5	50114.2 ± 42609.2	59084.8 ± 50059.6
XT16	27222.1	14307.9	**5107.1 ± 6867.4**	13400.9 ± 14203.6	11833.4 ± 14164.7	10834.5 ± 16141.6	9851.6 ± 14982.8
XT17	271583.7	263100.7	37592.2 ± 23203.0	61804.8 ± 38134.3	53070.5 ± 39649.6	49583.2 ± 41366.7	50658.0 ± 49799.0
XT18	163517.5	80359.2	566907.1 ± 241945.3	1218757.6 ± 838939.1	969576.2 ± 608931.7	708523.9 ± 400193.8	907566.7 ± 604111.3
XT19	91101.9	72758.9	97910.1 ± 50899.1	147089.5 ± 78345.2	132515.6 ± 84911.4	123054.2 ± 87883.8	136925.0 ± 100894.9
XT20	133643.9	120460.0	**79617.6 ± 44960.0**	170415.4 ± 117774.0	129630.6 ± 103966.3	101689.4 ± 85080.5	129091.8 ± 108976.2
XT21	570388.5	581547.7	283775.9 ± 127774.4	646679.5 ± 447791.3	470231.2 ± 321177.3	388479.5 ± 232209.4	467550.9 ± 332758.6
XT22	454536.9	420731.2	1805194.1 ± 878782.8	2918387.4 ± 1475730.7	2474629.4 ± 1419902.0	2117093.0 ± 1546182.9	2664044.8 ± 1592865.6
XT23	428607.7	432883.8	1079405.5 ± 564285.5	1872140.1 ± 1034353.0	1480497.6 ± 860919.3	1203965.7 ± 779382.5	1611240.4 ± 987963.4
XT24	227787.8	205629.2	2415320.4 ± 1896610.8	4383654.7 ± 3239783.9	3784575.3 ± 2717007.3	3283960.7 ± 3462732.4	4000878.5 ± 3123689.1

5.3.3 Stage Three: Specialised Conditionals

The third grammar uses the arithmetic operators, the `max` and `min` operators and three new conditional operators: `ifPS`, `ifOD` and `ifLO`. The new conditional operators each have a fixed condition, and take two arguments. The `ifPS` operator returns the first argument if the slack is non-negative, i.e., it is possible for the job to be completed on or before its due date ($\texttt{DD} - \texttt{CT} - \texttt{RT} \geq 0$) and returns the second argument otherwise. The `ifOD` operator tests whether the job is already overdue; if the job is overdue ($\texttt{DD} \leq \texttt{CT}$) the first argument is returned, else the second argument is returned. The final new conditional operator, `ifLO`, returns the first argument if it is the last operation of the job, or the second argument if it is not. These methods will be referred to as GP3 and ST3.

We observe that GP3 and ST3 best-of-run DRs have much longer tails of high TWT than GP1/ST1 and GP2/ST2. Figures 5.5 and 5.6 show that on a number of test scenarios the ST3 violin plot shows lower TWT is achieved than under GP3. The lower best values of ST3 are particularly obvious on, e.g., T7 and T13, and occurs less on extreme test instances.

Figures 5.5 and 5.6 show that the relative performance of ST3A, ST3S and ST3F compared to ST3 is very different across test instances and extreme test instances. Across the test instances, as shown in Figure 5.5, ST3A, ST3S and ST3F have worse performance, shown by the violin plot having most of its bulk higher than that of GP3 and ST3. However, on test instances T10, T22, T23 and the extreme test instances, the opposite is true, with these three methods generally outperforming both GP3 and ST3, as well as results being much more concentrated. The distribution of TWT from best-of-run classifiers of ST3A, ST3S and ST3F are very similar.

Figures 5.5 and 5.6 show that the overall shape of the violin plots are similar for GP and STGP methods with the same function set. This shows that we are achieving a similar distribution of TWT from the best-of-run DRs with grammars restricting the allowable interactions. From Tables 5.5, 5.6, 5.9, 5.10, 5.11 and 5.12 it is clear that all methods have large standard

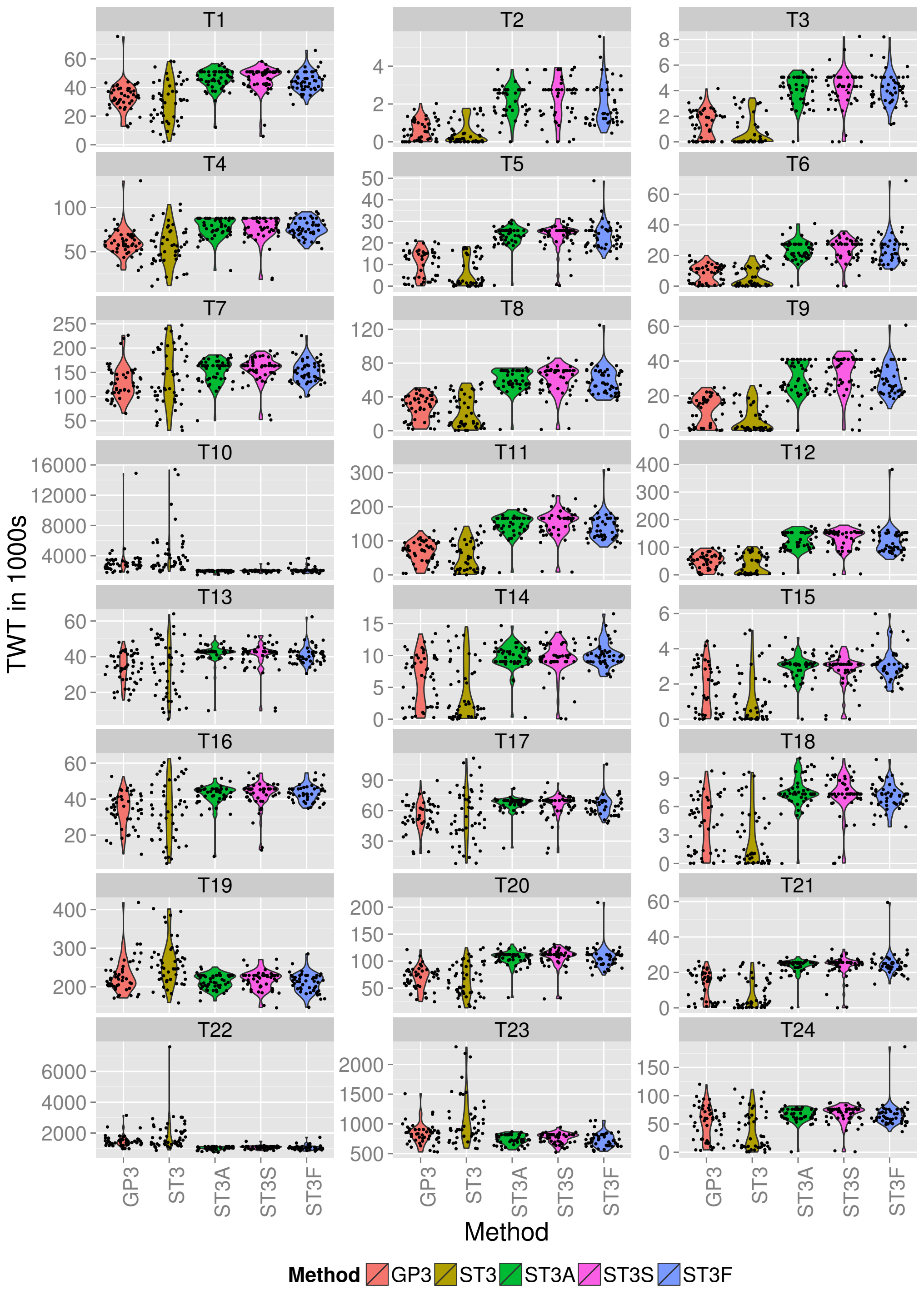


Figure 5.5: Violin plots presenting results (TWT in 1000s) of test scenarios for Stage 3 GP and STGP methods. Note that the vertical axis scales are different in each subfigure. These figures shows the distribution of TWT values attained from the 50 independent GP runs of each method.

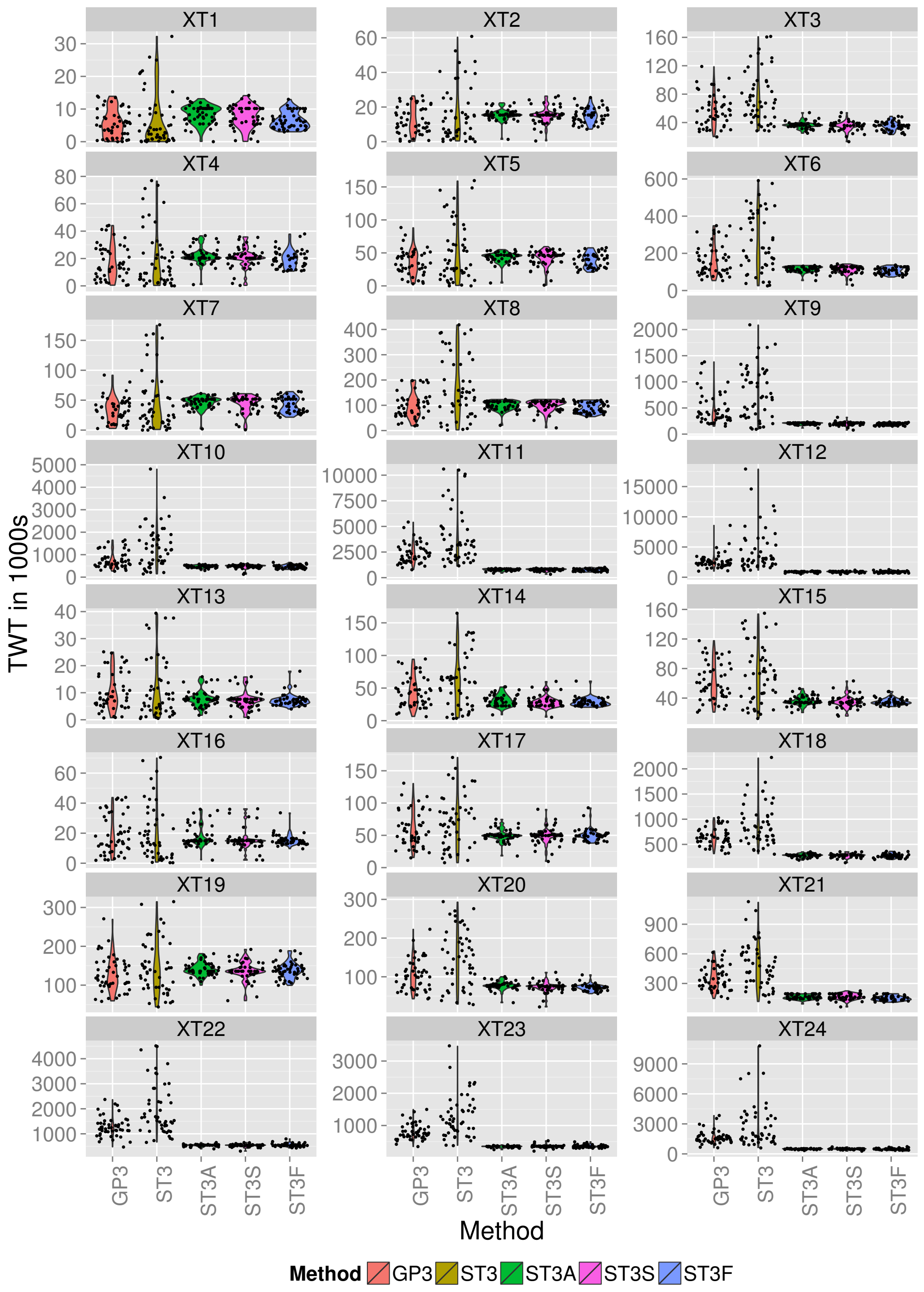


Figure 5.6: Violin plots presenting results (TWT in 1000s) of extreme test scenarios for Stage 3 GP and STGP methods. Note that the vertical axis scales are different in each subfigure. These figures shows the distribution of TWT values attained from the 50 independent GP runs of each method.

Table 5.11: Mean and standard deviation of total weighted tardiness on test problem instances of Stage 3 (lower values are better). Values that are statistically significantly better than all others across the same row across Tables 5.5, 5.9 and 5.11 are in **bold**. Where there are two values in a row that are bold, they are not statistically significantly different from each other, but are statistically significantly better than all remaining methods.

	GP3	ST3	ST3A	ST3S	ST3F
	Mean±Stdev	Mean±Stdev	Mean±Stdev	Mean±Stdev	Mean±Stdev
T1	33612.2 ± 9796.4	30563.1 ± 14272.8	45479.8 ± 7446.7	44183.9 ± 6897.4	45569.2 ± 9139.7
T2	639.4 ± 544.0	431.5 ± 573.2	2162.6 ± 799.6	2002.0 ± 1095.7	2317.4 ± 1126.2
T3	1350.5 ± 1140.8	740.2 ± 1131.3	4060.9 ± 1093.9	4157.3 ± 1256.7	4096.0 ± 1599.1
T4	60509.5 ± 15321.1	56746.7 ± 21450.8	77464.1 ± 11327.7	76371.8 ± 9807.1	76199.7 ± 14697.3
T5	10720.6 ± 6507.5	6238.7 ± 6728.5	23513.1 ± 4482.9	23413.2 ± 6007.9	23029.4 ± 6392.3
T6	8360.9 ± 5643.2	5393.1 ± 6191.3	23571.8 ± 6569.9	23570.2 ± 9554.6	23881.1 ± 7878.7
T7	129015.0 ± 33633.5	140445.8 ± 58753.7	151488.6 ± 25184.5	151674.3 ± 23324.5	156854.7 ± 26922.6
T8	27949.5 ± 14274.9	22329.7 ± 18269.4	60277.5 ± 12609.4	58441.4 ± 15757.1	60353.1 ± 17108.5
T9	12002.9 ± 8223.6	6457.9 ± 7847.9	30256.1 ± 9286.1	29170.1 ± 8787.8	31539.3 ± 11100.2
T10	3144528.1 ± 1827626.3	4174526.0 ± 2860867.0	**1956263.7 ± 128272.7**	2061974.5 ± 334956.0	**2002595.4 ± 165348.8**
T11	67712.0 ± 30844.9	54664.5 ± 42606.2	146677.4 ± 30283.5	139828.1 ± 37510.8	150886.8 ± 40748.8
T12	49615.5 ± 27694.0	36433.9 ± 34880.3	123663.3 ± 32431.0	121075.1 ± 48938.5	128450.3 ± 38061.1
T13	33668.5 ± 8475.4	33196.3 ± 15442.0	41698.9 ± 5832.6	40946.9 ± 5224.2	40721.0 ± 7769.7
T14	6293.6 ± 4010.8	4245.2 ± 4537.8	9821.9 ± 2054.9	10085.1 ± 1920.5	9499.1 ± 2752.9
T15	1788.6 ± 1390.1	1153.4 ± 1418.0	2921.3 ± 702.4	3066.6 ± 872.0	2792.5 ± 918.9
T16	34090.4 ± 9549.4	33463.1 ± 16224.0	42223.4 ± 6380.3	43255.2 ± 4458.1	42391.2 ± 7915.0
T17	54939.4 ± 14678.5	56006.5 ± 25696.4	66534.0 ± 7583.9	64863.9 ± 9417.9	65180.5 ± 11723.1
T18	4203.6 ± 2942.6	2547.7 ± 3022.8	7469.9 ± 1702.6	7061.1 ± 1308.6	7226.8 ± 2237.1
T19	228178.2 ± 43843.1	260078.5 ± 57227.4	213408.8 ± 18380.7	210524.0 ± 24715.1	216645.6 ± 24928.6
T20	69460.9 ± 20946.9	64913.1 ± 30238.7	105711.9 ± 14524.2	106499.4 ± 19651.9	107177.5 ± 18914.2
T21	12729.4 ± 8132.9	7981.6 ± 8478.1	23643.2 ± 4222.4	24798.1 ± 6260.4	23457.0 ± 6287.2
T22	**1468687.6 ± 376933.1**	1859420.2 ± 1023172.5	**1019826.1 ± 99915.8**	**1047586.6 ± 146840.7**	**1058920.5 ± 95949.4**
T23	**832372.5 ± 170750.4**	1064701.1 ± 406980.4	**737251.1 ± 80890.9**	**728326.8 ± 116495.3**	**761548.7 ± 84838.1**
T24	52003.2 ± 29316.8	38457.3 ± 33837.3	66278.4 ± 12901.0	66199.8 ± 20488.8	67082.3 ± 17431.1

Table 5.12: Mean and standard deviation of total weighted tardiness on extreme test problem instances of Stage 3 (lower values are better). Values that are statistically significantly better than all others across the same row across Tables 5.6, 5.10 and 5.12 are in **bold**. Where there are two values in a row that are bold, they are not statistically significantly different from each other, but are statistically significantly better than all remaining methods.

	GP3	ST3	ST3A	ST3S	ST3F
	Mean±Stdev	Mean±Stdev	Mean±Stdev	Mean±Stdev	Mean±Stdev
XT1	5585.8 ± 4176.8	6623.5 ± 8072.2	8316.0 ± 2702.9	6500.2 ± 2537.5	7701.8 ± 3305.6
XT2	12515.5 ± 7717.6	16122.3 ± 15636.4	15490.9 ± 3227.0	15315.3 ± 4327.2	15381.6 ± 4661.3
XT3	57053.4 ± 22683.8	75824.3 ± 35778.7	**36230.7 ± 4769.5**	**34938.1 ± 6176.0**	**34918.1 ± 6615.8**
XT4	17589.0 ± 12271.2	20002.2 ± 21374.1	22074.7 ± 5957.9	19661.8 ± 6336.9	20752.3 ± 7052.7
XT5	35661.2 ± 18954.8	50651.8 ± 44239.7	43102.1 ± 8094.1	38713.2 ± 10704.4	42122.4 ± 11812.1
XT6	162760.3 ± 83161.7	263476.9 ± 158978.0	115372.2 ± 14601.8	106044.1 ± 17483.7	112833.2 ± 20640.9
XT7	33639.8 ± 19850.7	46775.3 ± 48206.6	46043.4 ± 9898.2	42294.6 ± 12421.8	44372.0 ± 14095.7
XT8	97922.0 ± 49995.3	171796.5 ± 122011.0	99086.4 ± 18371.5	89794.3 ± 20126.7	98683.6 ± 22156.6
XT9	476989.7 ± 298486.3	789444.2 ± 495545.6	201208.6 ± 19540.5	191861.3 ± 22890.2	200067.2 ± 34353.3
XT10	751588.3 ± 354646.6	1373188.6 ± 936216.4	476012.0 ± 54207.5	467524.7 ± 80922.0	478666.1 ± 85252.2
XT11	2349556.2 ± 987480.4	3886735.7 ± 2590992.0	**774733.3 ± 93028.6**	**745848.3 ± 118624.5**	**770420.2 ± 110764.0**
XT12	2627514.8 ± 1218935.1	4408031.4 ± 3533272.3	**883510.6 ± 104592.0**	**858908.0 ± 143357.0**	**897888.5 ± 111472.2**
XT13	10151.6 ± 6776.2	10941.6 ± 10834.0	7751.7 ± 2954.8	7170.5 ± 2192.5	7201.6 ± 3138.4
XT14	43724.5 ± 22594.9	60368.3 ± 41481.6	30390.5 ± 8795.7	28499.1 ± 6457.3	29032.6 ± 9350.0
XT15	61714.7 ± 25193.8	71612.8 ± 38921.8	36251.9 ± 6075.0	35294.9 ± 4333.7	34992.2 ± 7936.3
XT16	18404.1 ± 12746.0	19899.3 ± 18441.9	17797.1 ± 6937.9	15245.3 ± 3509.4	15688.4 ± 6919.9
XT17	59779.3 ± 26957.9	69654.8 ± 40704.7	49882.3 ± 9112.7	50261.4 ± 9491.5	49263.1 ± 12473.4
XT18	645855.6 ± 181925.0	893531.9 ± 409308.1	**279551.1 ± 32253.5**	**270025.2 ± 39403.6**	**276997.4 ± 36923.3**
XT19	127506.5 ± 43260.7	143898.7 ± 67458.7	140799.8 ± 16836.6	138573.5 ± 20037.1	137249.7 ± 23973.4
XT20	104074.0 ± 39349.7	147327.0 ± 73759.1	77318.2 ± 10348.1	72297.6 ± 8259.6	73397.7 ± 13065.1
XT21	355661.8 ± 120027.8	481870.7 ± 227198.4	**165612.6 ± 24302.0**	**150802.2 ± 24078.8**	**165855.8 ± 29637.2**
XT22	1277834.8 ± 392185.2	1983138.2 ± 1032717.2	**537522.1 ± 40470.4**	**547286.8 ± 63431.7**	**535963.6 ± 52492.3**
XT23	799482.9 ± 245972.9	1270466.8 ± 644221.9	**344325.1 ± 24959.7**	**355971.6 ± 43697.7**	**353005.8 ± 44938.6**
XT24	1642738.2 ± 626403.4	2798365.1 ± 2117136.1	**487811.9 ± 45229.2**	**477324.4 ± 81627.9**	**488801.1 ± 60224.9**

deviations relative to the means, and that ST3A, ST3S and ST3F are the best methods on a number of extreme test instances.

The performance decrease value is $m_{GP3,ST3} = 0.4881$, $m_{GP3,ST3A} = -0.4552$, $m_{GP3,ST3S} = -0.4482$ and $m_{GP3,ST3F} = -0.4562$. The performance decrease between between GP3 and ST3 is very similar to the performance decrease between ST1 and GP1. It is interesting that ST3A, ST3S and ST3F *improve* on the performance of GP3. This is shown by the negative performance measure values. These three methods attain TWT which is approximately 60% of the TWT attained by GP3. This is a considerable improvement. These are the only three STGP methods which improve on their corresponding GP method. Although GP1 has a lower mean than GP3 on many scenarios, GP3 outperforms GP1 and GP2 using this performance measure. This shows that the performance of individual DRs evolved by GP3 is more consistent than the performance of individual DRs evolved by both GP1 and GP2.

5.4 Analysis of Evolved Rules

Tables 5.13, 5.14 and 5.15 show the number of occurrences of the terminals and functions in the 50 best-of-run DRs, as well as the number of DRs the terminal/function appears in, for GP and STGP. This is constructed from unsimplified DRs. If we were to manually simplify the rules then the values in this table would change, as there are often fragments which are redundant or cancel out. However, simplification is sometimes subjective and is beyond the scope of this chapter.

The most frequently used terminals across all methods are `PR`, `DD`, `CT` and `NNQ`. The calculation of TWT requires both the job's weight and due date, so we expect both `W` and `DD` to appear in DRs for this objective function for effective performance. The least used terminals are `NQW` and `NPR`, i.e., the expected queue wait at the next machine the job visits and the processing time at that machine. `RJ` is also one of the least frequently used

Table 5.13: Number of occurrences of functions and terminals in rules and number of rules functions and terminals occur in for Stage 1 methods.

	GP1		ST1		ST1A		ST1B		ST1S		ST1F	
	Total	Rules	Total	Rules	Total	Rules	Total	Rules	Total	Rules	Total	Rules
$+$	194	48	94	41	96	39	76	38	99	40	85	34
$-$	230	50	139	50	169	49	154	50	154	50	130	49
$*$	177	49	194	46	221	48	170	48	183	50	168	47
%	190	48	175	49	145	44	127	43	157	48	163	49
DD	146	50	83	50	88	50	71	50	73	49	75	48
PR	126	50	71	37	34	19	36	21	37	23	23	16
W	51	24	59	28	65	31	87	35	45	23	22	18
RT	93	42	56	34	52	31	53	29	65	34	62	32
RO	56	28	72	38	71	30	61	30	41	29	37	27
RJ	28	16	12	9	17	11	3	3	20	14	15	10
RM	50	29	46	28	45	32	39	28	37	29	39	29
NQ	46	25	72	33	74	30	76	32	70	33	62	31
QW	18	12	14	9	17	13	12	9	25	19	16	11
CT	90	38	53	36	51	33	44	29	51	33	42	33
NPR	11	8	14	11	24	20	23	17	13	10	14	11
NNQ	85	47	73	42	55	39	53	37	42	27	42	33
NQW	10	9	11	10	4	4	4	4	18	10	9	9
AQW	41	24	27	22	15	11	19	11	17	15	26	23
1/PR	-	-	-	-	73	30	-	-	65	33	48	26
W/PR	-	-	-	-	-	-	63	31	-	-	-	-
[PR*PR]	-	-	-	-	-	-	-	-	42	25	42	24
[1/[PR*PR]]	-	-	-	-	-	-	-	-	-	-	31	19
Total	1642	-	1265	-	1316	-	1171	-	1254	-	1151	-

Table 5.14: Number of occurrences of functions and terminals in rules and number of rules functions and terminals occur in for Stage 2 methods.

	GP2		ST2		ST2A		ST2S		ST2F	
	Total	Rules	Total	Rules	Total	Rules	Total	Rules	Total	Rules
+	143	42	70	30	76	34	69	24	55	26
−	195	50	153	50	144	50	149	50	120	49
*	145	46	163	46	154	47	166	46	171	48
%	136	48	125	45	144	47	178	46	187	48
max	79	36	67	29	73	32	63	31	68	30
min	93	33	53	30	53	29	62	25	55	26
if>0	81	34	73	35	66	32	87	35	71	31
PR	141	50	70	32	39	22	54	26	29	15
RT	88	33	63	36	61	33	48	24	37	20
RO	49	29	61	28	67	31	72	30	77	34
RJ	50	23	31	17	26	18	33	15	37	21
DD	146	50	92	50	103	49	95	50	102	49
W	54	30	54	23	62	30	69	35	64	31
RM	85	36	61	34	53	33	67	35	61	33
NQ	79	32	141	38	104	37	107	37	66	34
QW	47	21	38	23	27	18	12	9	24	15
CT	103	40	58	35	62	37	68	37	59	35
NPR	36	18	36	20	22	19	14	11	24	13
NNQ	101	41	99	45	86	42	91	46	77	37
NQW	34	18	14	12	12	10	21	14	17	11
AQW	24	15	23	15	28	21	45	24	23	15
1/PR	-	-	-	-	86	37	80	33	65	35
[PR*PR]	-	-	-	-	-	-	56	19	60	24
[1/[PR*PR]]	-	-	-	-	-	-	-	-	43	20
Total	1909		1545		1548		1706		1592	

terminals; often `CT` will be approximately equal.

The introduction of three `if` statements with predetermined conditions has given an increase in the overall number of conditional statements used (see Table 5.15 on page 196). Further, the most useful of the three, based upon overall frequency and number of rules it appears in, is `ifPS`, where the condition is the slack of the job. Every best-of-run rule (50/50) in GP3 uses the `ifPS` function, which is far more often than the `if>0` function is used in GP2 (34/50 rules). With the introduction of the specialised conditionals, the number of rules where the arithmetic operators are present has decreased. Also, although the terminal `DD` appears less frequently, in only 29 GP3 rules and 42 ST3 rules, it is present in the condition of `ifPS`.

5.4.1 Fragment Analysis

We have also analysed which are the most common two-level fragments, i.e., one function with terminals as all its arguments. The occurrences of terminals and fragments reveal what knowledge GP and STGP are discovering, e.g., (− `DD CT`) which could be included in future terminal sets. These also give us insight into potential grammars, by looking at the most popular fragments, and the type interactions within them.

The top 10 most common for each method are shown in Tables 5.16, 5.17 and 5.18. The most common fragments in GP1, ST1, ST1S, GP2, ST2A, ST2S and ST3 are `(− CT DD)` or `(− DD CT)`, the time left until the job is overdue. The most common fragment in ST1A, ST1B, ST1F, ST2 and ST2S is similar, `(− RM DD)` which is also comparing the time to the due date. It is interesting that the top nine most common fragments in GP2 are all allowable under the introduced grammar. Two of the top ten fragments under GP1 are not allowable: adding two counts and multiplying durations of time. It is interesting to compare the number of occurrences of these fragments in total to the number of rules the fragment appears in. For example, the fragments `(− CT DD)` or `(− DD CT)` occur 36 times

Table 5.15: Number of occurrences of functions and terminals in rules and number of rules functions and terminals occur in for Stage 3 methods.

	GP3		ST3		ST3A		ST3S		ST3F	
	Total	Rules	Total	Rules	Total	Rules	Total	Rules	Total	Rules
$+$	81	38	72	30	88	49	83	47	68	36
$-$	90	37	103	41	19	13	27	17	19	10
$*$	75	36	141	48	204	50	176	49	256	50
%	130	40	144	41	91	40	87	32	141	40
max	61	31	51	30	52	28	52	27	39	25
min	64	29	39	21	48	28	41	26	27	17
if>0	-	-	-	-	-	-	-	-	-	-
ifPS	133	50	87	44	117	50	100	50	107	50
ifOD	54	28	39	25	44	29	39	23	36	23
ifLO	58	34	55	33	50	29	44	23	26	15
PR	145	49	61	32	7	5	7	6	14	9
RT	28	17	28	14	5	4	13	7	24	15
RO	62	30	73	30	32	16	20	16	40	24
RJ	39	24	24	14	4	4	14	10	32	21
DD	51	29	74	42	2	2	16	10	27	18
W	84	36	106	32	203	50	163	50	169	50
RM	54	28	54	27	11	7	17	10	15	11
NQ	87	38	113	43	55	23	38	19	41	26
QW	49	25	28	16	4	3	10	7	15	12
CT	43	26	68	32	15	11	17	10	21	13
NPR	17	10	20	11	6	4	6	3	6	3
NNQ	91	48	103	48	27	18	16	8	22	13
NQW	23	14	15	8	0	0	3	2	2	2
AQW	46	28	29	17	6	5	9	6	21	15
1/PR	-	-	-	-	386	50	325	50	138	40
[PR*PR]	-	-	-	-	-	-	28	16	57	29
[1/[PR*PR]]	-	-	-	-	-	-	-	-	127	39
Total	1565		1527		1265		1168		1321	

Table 5.16: Number of occurrences of fragments of depth two for Stage 1 methods. We have combined (op A B) and (op B A) and present the top 10 most common fragments for each method.

GP1	Total	Rules	ST1	Total	Rules	ST1A	Total	Rules
(− DD CT)	36	23	(− DD CT)	19	15	(− DD RM)	21	16
(− DD RM)	27	18	(− DD RM)	17	11	(− RT DD)	14	12
(− RT DD)	14	8	(∗ RO RO)	12	11	(− DD CT)	15	10
(% PR CT)	11	8	(% W NNQ)	11	10	(∗ NQ NNQ)	12	9
(+ NQ NNQ)	7	6	(⋆ RO NQ)	12	9	(⋆ RO NQ)	9	7
(∗ PR AQW)	7	6	(⋆ NQ NNQ)	12	9	(% W NNQ)	7	6
(% PR RT)	11	6	(⋆ RT RO)	6	6	(+ PR DD)	6	5
(% W NNQ)	7	5	(⋆ NNQ NNQ)	7	6	(∗ RO RO)	9	5
(+ PR PR)	5	4	(⋆ PR NQ)	6	5	(⋆ RO W)	7	5
(− PR RT)	4	4	(⋆ PR NNQ)	6	5	(⋆ NQ NQ)	8	5

ST1B	Total	Rules	ST1S	Total	Rules	ST1F	Total	Rules
(- DD RM)	17	15	(- DD CT)	18	15	(- DD RM)	13	11
(- DD CT)	17	13	(- DD RM)	12	11	(- DD CT)	12	10
(⋆ RO NQ)	14	9	(- PR CT)	6	5	(⋆ RO NQ)	8	7
(+ RT RM)	10	8	(⋆ NQ NNQ)	7	5	(⋆ NQ NNQ)	9	7
(⋆ NQ NQ)	9	7	(+ RT CT)	5	4	(+ RT CT)	5	5
(⋆ NQ NNQ)	7	7	(⋆ W 1/PR)	4	4	(- RT DD)	6	5
(⋆ RO RO)	6	6	(% W NNQ)	4	4	(% W NNQ)	4	4
(% NQ W/PR)	7	6	(% NNQ 1/PR)	4	4	(% NQ 1/PR)	4	4
(⋆ NNQ NNQ)	5	5	(+ RT DD)	3	3	(+ DD QW)	4	3
(- RT DD)	4	4	(+ RT RM)	3	3	(+ RT RM)	2	2

Table 5.17: Number of occurrences of fragments of depth two for Stage 2 methods. We have combined (op A B) and (op B A) and present the top 10 most common fragments for each method.

GP2	Total	Rules	ST2	Total	Rules	ST2A	Total	Rules
(− DD CT)	31	18	(− DD RM)	17	15	(− DD CT)	18	13
(− DD RM)	30	14	(* PR NQ)	10	8	(− RT DD)	13	10
(% PR DD)	11	8	(− DD CT)	10	7	(− DD RM)	19	9
(* RT W)	6	5	(% W NNQ)	8	7	(% W NNQ)	7	7
(% PR RM)	9	5	(+ RT CT)	6	6	(% NNQ 1/PR)	9	6
(% PR CT)	6	5	(* RO NQ)	9	6	(% NQ 1/PR)	9	6
(+ RT CT)	6	4	(− RT DD)	6	5	(max NQ NNQ)	8	6
(* PR NQ)	7	4	(min NQ NNQ)	6	5	(* NQ NNQ)	6	5
(% PR RJ)	7	4	(* NQ NNQ)	8	4	(* RT W)	6	4
(+ PR W)	7	3	(max RO NNQ)	5	4	(* RT 1/PR)	6	4

ST2S	Total	Rules	ST2F	Total	Rules
(− DD RM)	18	11	(− DD CT)	13	9
(− DD CT)	11	10	(− DD RM)	13	8
(max NQ NNQ)	10	8	(% W NNQ)	7	6
(* W 1/PR)	7	6	(max NQ NNQ)	7	5
(% W NNQ)	6	6	(− RT DD)	6	4
(+ RT CT)	5	5	(max DD CT)	5	4
(− CT AQW)	7	5	(% PR RO)	5	3
(+ PR DD)	5	4	(min RO NNQ)	3	3
(− RT DD)	4	4	(min DD RM)	4	3
(* RO NQ)	3	3	(+ RM AQW)	5	2

but in only 23 distinct rules under GP1.

The most common fragments of GP3, ST3A, ST3S and ST3F are the most different from the other methods. Here the most frequent is the WSPT rule for both ST3A and ST3S, and WSPT multiplied by PR for ST3F. The most common across all methods are differences between the current time or machine ready time and the job due date. Ratios of current time or machine ready time to job processing time appear in the top 10 of GP1, GP2 and GP3, yet not in the top 10 of any STGP methods (even though allowed).

5.4.2 Best Evolved DRs

We have selected some of the best evolved DRs from each method, based on comparing their performance across test and extreme test instances with the performance of ATC and WCOVERT. The DRs shown have not been simplified. The DRs evolved by STGP methods are shorter and easier to interpret.

Stage One

Figures 5.7 and 5.8 show one of the best rules evolved under GP1 and ST1 respectively. In Figure 5.7, the function nodes which take argument pairs that would not be allowed under the grammar are shaded. This affects nodes higher up the tree. This DR cannot easily be interpreted, but we can see that a fragment that occurs twice in the tree is $\frac{\mathtt{PR}}{\mathtt{RT}}(\mathtt{DD} - \mathtt{CT})$. This is the ratio of the processing time of the current operation to the total remaining time, multiplied by the time until the job is due.

The DR of Figure 5.8 performed similarly to the DR of Figure 5.7 in terms of performance relative to ATC and WCOVERT, and also contains the fragment $\frac{\mathtt{PR}}{\mathtt{RT}}(\mathtt{DD} - \mathtt{CT})$. The presence of this fragment in two DRs with very good performance suggests that this could make a useful higher-level terminal.

Table 5.18: Number of occurrences of fragments of depth two for Stage 3 methods. We have combined (op A B) and (op B A) and present the top 10 most common fragments for each method.

GP3	Total	Rules	ST3	Total	Rules	ST3A	Total	Rules
(% NQ NNQ)	8	6	(- DD CT)	15	11	(* W 1/PR)	38	22
(% W NNQ)	6	5	(% W NNQ)	17	11	(+ W W)	13	8
(- PR W)	6	4	(- DD RM)	8	7	(% W W)	4	4
(- DD RM)	7	4	(* PR NNQ)	8	6	(max NQ NNQ)	5	4
(- DD CT)	5	4	(ifOD RO NQ)	7	5	(min W W)	6	4
(% PR W)	7	4	(- PR CT)	6	4	(% W NNQ)	3	3
(min RT NNQ)	4	4	(* RO NQ)	5	4	(max W W)	3	3
(* NNQ AQW)	3	3	(% DD CT)	7	4	(* RO 1/PR)	2	2
(% PR RO)	9	3	(ifPS RO NQ)	6	4	(* W W)	3	2
(% PR RJ)	3	3	(+ W W)	4	3	(+ RT RM)	1	1

ST3S	Total	Rules	ST3F	Total	Rules
(* W 1/PR)	32	21	(% W [PR*PR])	17	11
(+ W W)	5	4	(* W 1/PR)	15	10
(* NQ 1/PR)	4	4	(* W [1/[PR*PR]])	11	9
(% RT [PR*PR])	4	3	(+ W W)	5	4
(% RO W)	3	3	(% W NNQ)	8	4
(% QW [PR*PR])	4	3	(* RO [1/[PR*PR]])	3	3
(- RM NPR)	2	2	(* CT [1/[PR*PR]])	3	3
(* W NQ)	2	2	(+ RT QW)	3	2
(* NNQ [PR*PR])	2	2	(* RT 1/PR)	2	2
(% W W)	3	2	(* RO W)	3	2

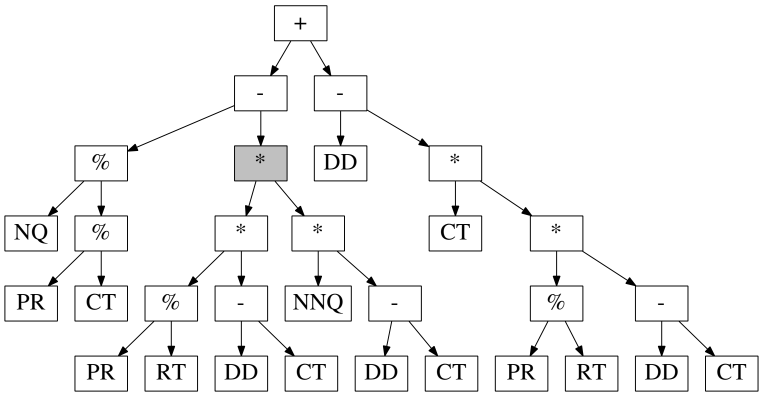


Figure 5.7: An evolved rule from *GP1* which outperformed ATC on 34/48 test instances and WCOVERT on 36/48 test instances.

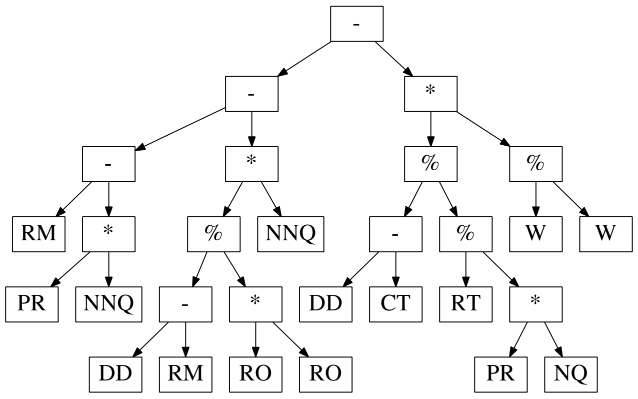


Figure 5.8: An evolved rule from *ST1* which outperformed ATC on 33/48 test instances and WCOVERT on 39/48 test instances.

Figures 5.9 and 5.10 show one of the best rules evolved under ST1A and ST1B respectively. The DR of Figure 5.9 shows that wherever the terminal `1/PR` appeared, it was multiplied by `W`. This is one reason `W/PR` was used as the only I type terminal for ST1B. With some simplification this rule becomes

$$\frac{\mathtt{PR}}{\mathtt{RT}}(\mathtt{RM}-\mathtt{DD})\left(\frac{\mathtt{RO}/\mathtt{W}+1}{\mathtt{RO}*\mathtt{RJ}}\right)$$

Note that this has a very similar fragment to that mentioned above, with $(\mathtt{RM}-\mathtt{DD})$ instead of $(\mathtt{DD}-\mathtt{CT})$. The ready machine time will often be

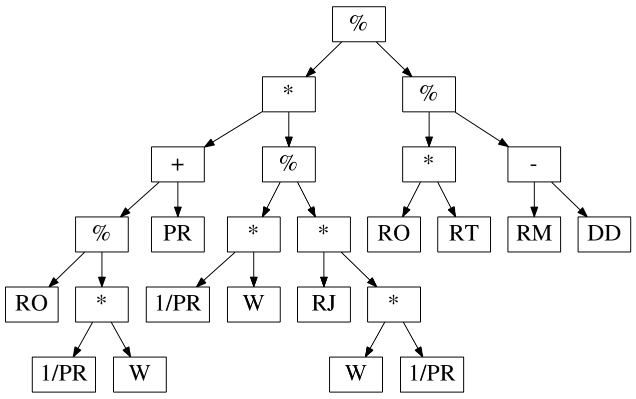


Figure 5.9: An evolved rule from *ST1A* which outperformed ATC on 25/48 test instances and WCOVERT on 32/48 test instances.

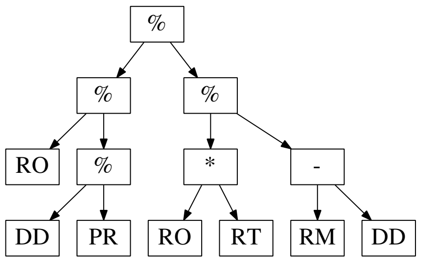


Figure 5.10: An evolved rule from *ST1B* which outperformed ATC on 34/48 test instances and WCOVERT on 38/48 test instances.

the same as the current time, unless the machine has been idle and multiple jobs arrive at the machine queue at the same time. If the machine has a queue of jobs waiting, the scheduling decision is made at the time the machine becomes available, which is the current time, hence in this situation `CT=RM`. However, if when an operation finishes there is *not* a waiting job, the machine becomes idle until a job arrives and it is immediately dispatched (there is no need to use the DR when there is only one job to choose from). Hence the only time that `RM` and `CT` will take different values in the DR when priority values are assigned is when multiple jobs arrive at an idle machine at exactly the same time. The DR of Figure 5.10

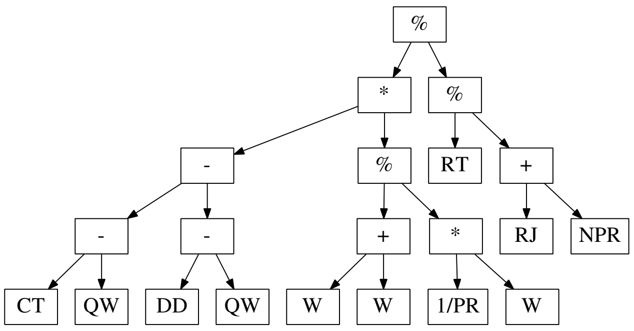


Figure 5.11: An evolved rule from *ST1S* which outperformed ATC on 26/48 test instances and WCOVERT on 34/48 test instances.

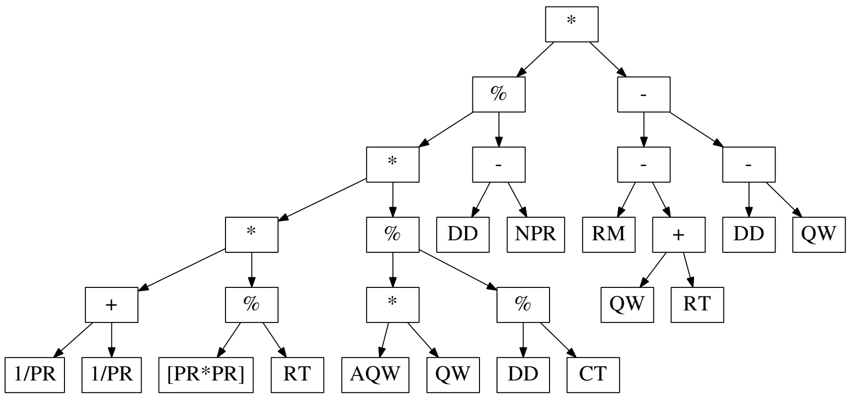


Figure 5.12: An evolved rule from *ST1F* which outperformed ATC on 25/48 test instances and WCOVERT on 37/48 test instances.

simplifies to an expression similar to that of Figure 5.9:

$$\frac{\mathtt{PR}}{\mathtt{RT}}\frac{(\mathtt{RM}-\mathtt{DD})}{\mathtt{DD}}$$

Figures 5.11 and 5.12 show one of the best rules evolved under ST1S and ST1F respectively. The DR of Figure 5.11 simplifies to

$$\frac{2\mathtt{PR}}{\mathtt{RT}}(\mathtt{CT}-\mathtt{DD})(\mathtt{RJ}+\mathtt{NPR}).$$

Interestingly, the job weight does not appear in the simplified rule, however there is the comparison of the current time to the job's due date. The

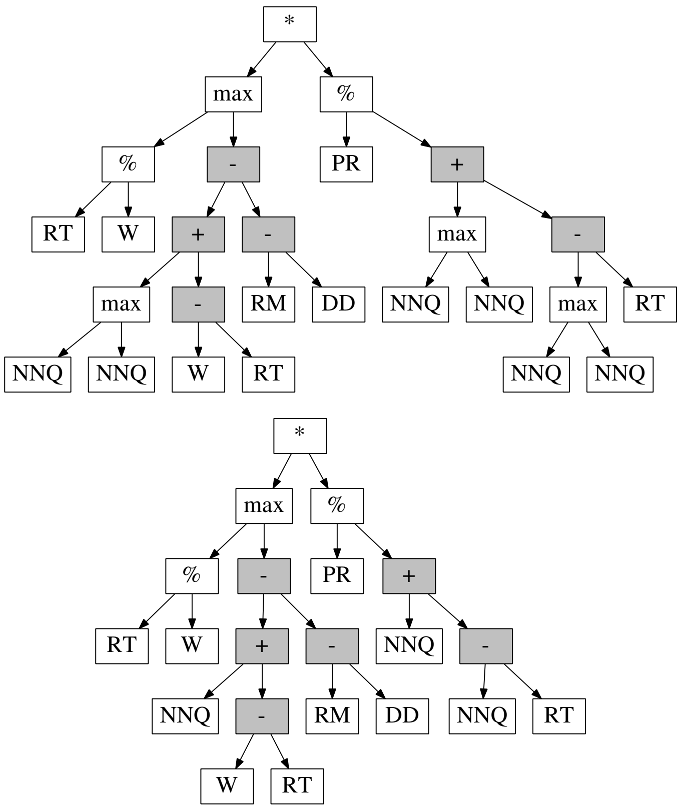


Figure 5.13: An evolved rule from *GP2*, and in simplified form below, wh-ich outperformed ATC on 44/48 test instances and WCOVERT on 47/48 test instances.

DR of Figure 5.12 simplifies to

$$\frac{2 * \mathtt{PR} * \mathtt{AQW} * \mathtt{QW} * \mathtt{CT} * (\mathtt{RM} - \mathtt{RT} - \mathtt{DD})}{\mathtt{RT} * \mathtt{DD} * (\mathtt{DD} - \mathtt{NPR})}.$$

Once again, the job weight does not appear in the simplified rule.

Stage Two

Figures 5.13 and 5.14 show one of the best rules evolved under GP2 and ST2 respectively. This GP2 rule simplifies to the DR in the lower figure in Figure 5.13. This DR outperformed or equalled WCOVERT on 47/48 test

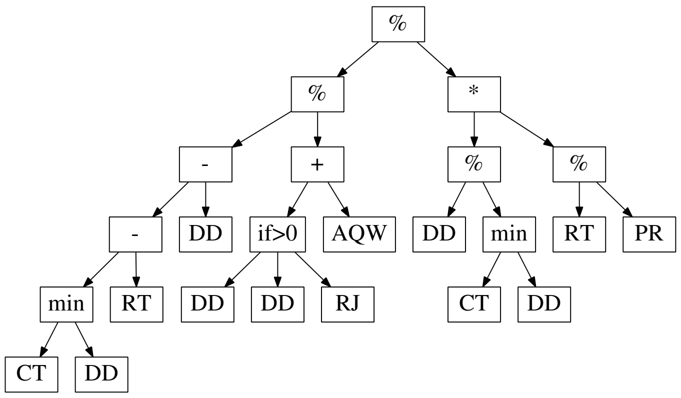


Figure 5.14: An evolved rule from *ST2* which outperformed ATC on 35/48 test instances and WCOVERT on 43/48 test instances.

and extreme test instances and equalled or outperformed ATC on 44/48 test and extreme test instances. However, the shaded nodes show that this rule has an expression $\mathtt{NNQ} + \mathtt{W} - \mathtt{RT} - \mathtt{RM} + \mathtt{DD}$ which adds or subtracts a weight, a count, a time duration and fixed points in time, i.e., the very interactions we are trying to prevent.

The DR from ST2 shown in Figure 5.14, shows an example of how the `if>0` operator can have a first argument that is always positive, namely `if>0 DD`, since `DD` is always positive. This rule assigns priority

$$\frac{-\mathtt{PR}}{\mathtt{DD} + \mathtt{AQW}}$$

if the job is overdue, and

$$\left(\frac{-\mathtt{PR}}{\mathtt{DD} + \mathtt{AQW}}\right)\left(\frac{\mathtt{CT} * (\mathtt{DD} - \mathtt{CT} + \mathtt{RT})}{\mathtt{DD} * \mathtt{RT}}\right)$$

if the job is not overdue.

Figure 5.15 shows one of the best performing DRs evolved by ST2A. It simplifies to

$$\frac{(\min\{\mathtt{RM}, \max\{\mathtt{DD}, \mathtt{CT}\} - \mathtt{RT}\} - \max\{\mathtt{DD}, \mathtt{CT}\})\mathtt{PR}}{\mathtt{RT}}$$

which is relatively easy to interpret, and allows us to understand why it performs so well. There are three different cases: (1) if the job is overdue

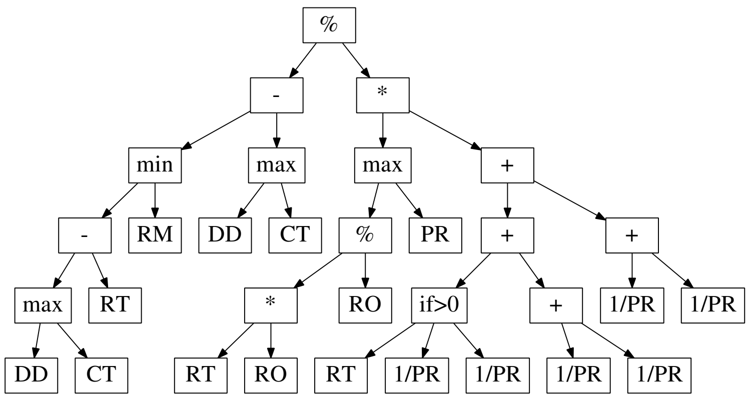


Figure 5.15: An evolved rule from *ST2A* which outperformed ATC on 39/48 test instances and WCOVERT on 43/48 test instances.

and has been waiting in the queue for the machine to become available, then the job is assigned priority 0; (2) if the job is not overdue but cannot complete on time ($\texttt{DD}-\texttt{RT}<\texttt{RM}$), then the job is assigned priority proportional to $(-\texttt{PR})$, which is always negative as processing time is always positive; and (3) if the job is not overdue ($\texttt{DD}>\texttt{CT}$) and has positive slack ($\texttt{DD}-\texttt{RT}>\texttt{CT}$), then ($\texttt{DD}-\texttt{RT}>\texttt{RM}$) and the job is assigned priority proportional to

$$\left(\frac{\texttt{DD}-\texttt{RT}}{\texttt{RT}}\right) * (-\texttt{PR}).$$

This is the priority of situation (2) multiplied by the ratio of remaining time until overdue to remaining processing time, which, as we know the job has positive slack, must be greater than 1. The priority assigned in this case is also always negative, therefore for two jobs with the same PR, a job which had positive slack will be assigned a lower priority than one which does not have positive slack.

Figures 5.16 and 5.17 show one of the best rules evolved under ST2S and ST2F respectively. Figure 5.16 can be rewritten as

$$\frac{(\texttt{CT}-\texttt{DD}) * (\texttt{RJ}/\texttt{RT}) * \max\{\texttt{NQ},\texttt{NNQ}\} * ((\texttt{DD}-\texttt{RT})/(\texttt{DD}-\texttt{AQW}))}{\texttt{NQ} * \min\{\texttt{[1/PR]} * \texttt{W}, \texttt{[1/PR]}\}}.$$

Once again, this rule has comparisons between the current time and the

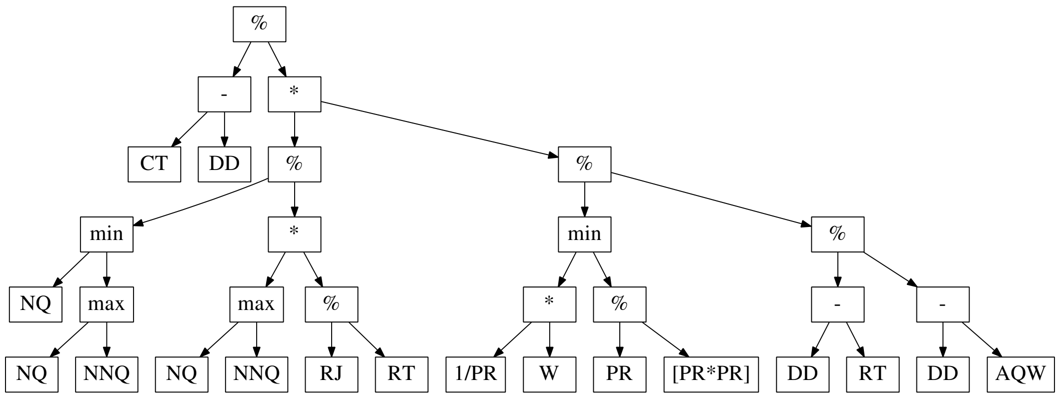


Figure 5.16: An evolved rule from *ST2S* which outperformed ATC on 23/48 test instances and WCOVERT on 33/48 test instances.

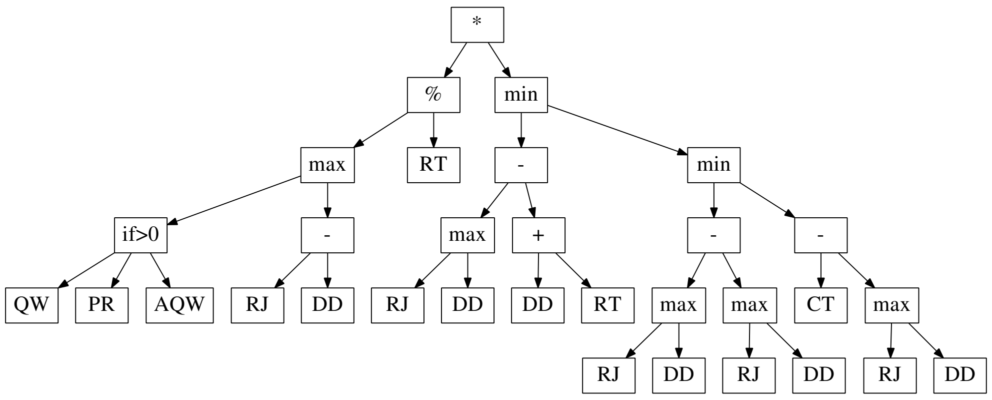


Figure 5.17: An evolved rule from *ST2F* which outperformed ATC on 29/48 test instances and WCOVERT on 38/48 test instances.

job's due date, as does the rule in Figure 5.17. It is interesting that this rule takes the maximum between `RJ`, the ready time of the job's current operation, and `DD`, the job's due date. This means the rule schedules jobs that were already overdue when the current operation became available differently from those that were not already overdue.

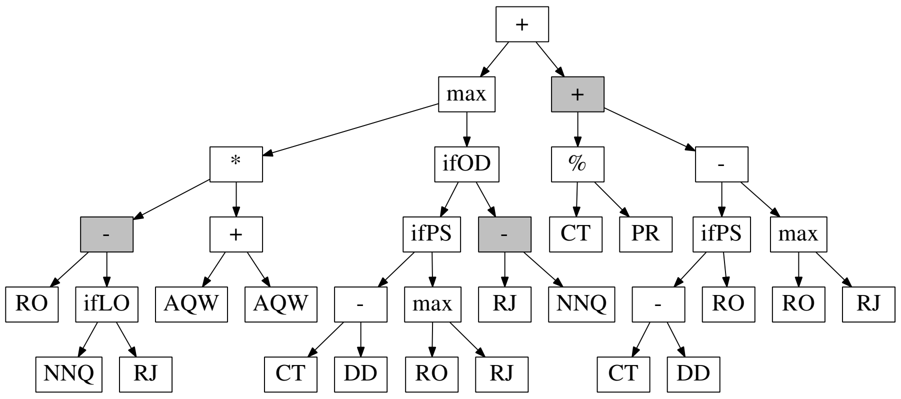


Figure 5.18: An evolved rule from *GP3* which outperformed ATC on 22/48 test instances and WCOVERT on 34/48 test instances.

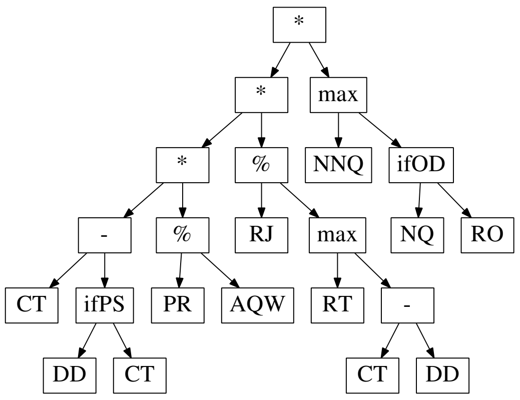


Figure 5.19: An evolved rule from *ST3* which outperformed ATC on 36/48 test instances and WCOVERT on 38/48 test instances.

Stage Three

Figures 5.18 and 5.19 show one of the best rules evolved under GP3 and ST3 respectively. The shaded nodes in the DR in Figure 5.18 show that this rule has a fragment which subtracts `RJ` (a clock time) and `NNQ` (a count). The DR from GP3 shown in Figure 5.18 has each of the new conditional operators in it, however the arguments returned are almost always of different return types. Therefore even though the branching condition makes sense, the different outcomes dependent on the condition do not make so much sense. With so many conditional operators with these arguments, it is difficult to understand what the DR is doing. One feature of the grammar used is that all count terminals are of the same type, allowing fragments like $\mathtt{max}\{\mathtt{NNQ}, \mathtt{RO}\}$, which will occur in this DR if the job is overdue. Taking the maximum of the number of jobs in the queue and the number of remaining operations in the job is not the most sensible, this could be an area of potential future refinement in the grammar. There is a great difference in performance, in favour of ST3, compared to benchmark ATC and WCOVERT rules of the DRs in Figures 5.18 and 5.19, the biggest difference observed between methods with the same function and terminal sets.

Figure 5.20 shows one of the best evolved DRs under ST3A. This DR has five `max` and `min` statements, as well as one `ifPS` statement. This means that depending on the values of the given job, priority may be assigned using different functions, and it is difficult to simplify. Figure 5.21 shows one of the best evolved DRs under ST3S. This DR uses the `ifPS` terminal, but also compares the job due date to the earliest time the job can complete in the `min` statement in the leftmost branch. This DR is better than the best from ST2S and ST1S in terms of how frequently it outperformed ATC and WCOVERT. Figure 5.22 shows one of the best evolved DRs by ST3F. This rule outperformed ATC and WCOVERT the least number of times of all DRs shown in Figures 5.7–5.22. This is also a rather large DR, in terms of the number of nodes, compared to many of the other best of method DRs.

5.5 Further Discussions

Here we present further discussions of our results, examining the impact of tree depth, how performance of the best evolved rules compare on test and extreme test scenarios, and how the evolution and training time compares across all methods.

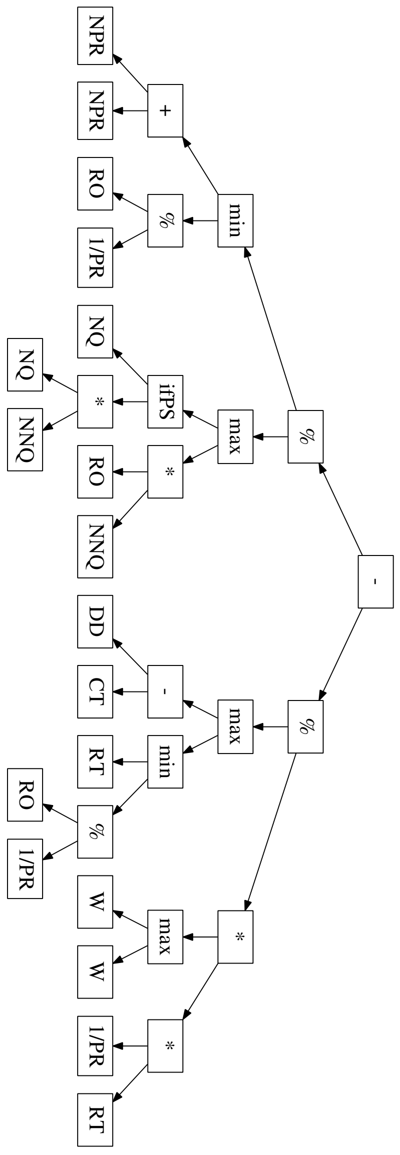


Figure 5.20: An evolved rule from *ST3A* which outperformed ATC on 31/48 test instances and WCOVERT on 43/48 test instances.

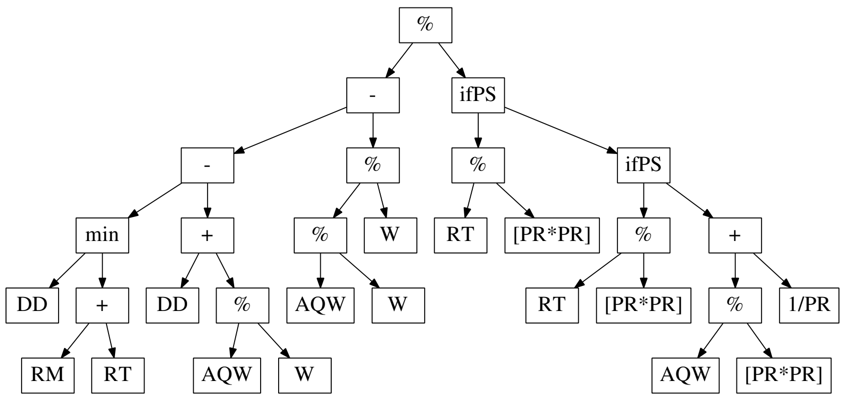


Figure 5.21: An evolved rule from *ST3S* which outperformed ATC on 33/48 test instances and WCOVERT on 42/48 test instances.

5.5.1 DR Size

In general, the larger a DR is, the more difficult it is to interpret; intuitively larger dispatching rules are more complex, and therefore expected to be harder to interpret. This is evident when looking at existing DRs in the literature. The SPT and EDD rules are clearly interpretable, and they are small in size. The ATC and WCOVERT rules are much larger, and require much more complicated calculation to determine the priority value of waiting jobs; they are still interpretable but it is more difficult to work out from the formula the clear interpretation which the name implies. As quantifying interpretability is an *extremely* difficult task, it is also difficult to illustrate how the difficulty of interpretation increases with DR size.

There is a clear trend that the best-of-run rules evolved under a grammar are smaller than their standard GP counterparts, and they also generally have smaller standard deviations. The mean number of nodes in DRs from GP1 is 32.8 compared to 25.3 from ST1, 26.3 from ST1A and 23.4 from ST1B. The difference between GP1 and the strongly typed ST1, ST1A and ST1B are all statistically significant at the 5% significance level. ST1S has mean number of nodes 25.1 and ST1F has mean 23.0. GP2 has mean number of nodes 44.2 compared to 30.96 from ST2, which is statistically signif-

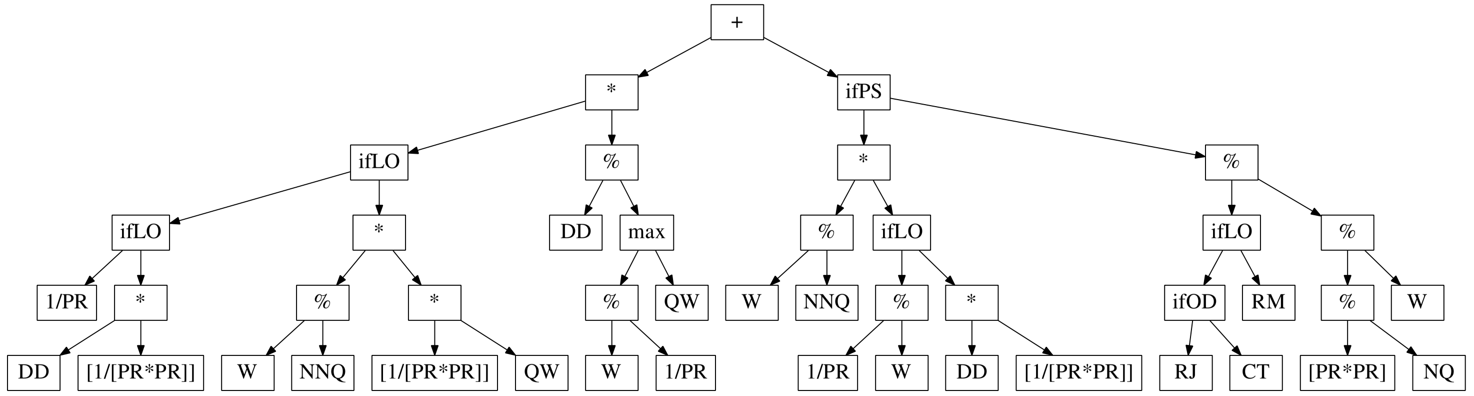


Figure 5.22: An evolved rule from *ST3F* which outperformed ATC on 18/48 test instances and WCOVERT on 28/48 test instances.

icant at 5%, and GP3 has mean number of nodes 31.3 and ST3 has 30.5, and this difference is not statistically significant. ST2A has mean number of nodes 31.0, ST2S has 34.1 and ST2F has 31.8. ST3A has a mean of 25.3, ST3S has 23.4 and ST3F has a mean of 26.4. These are smaller than those evolved at Stage 2, and very similar to those evolved at Stage 1. The maximum number of nodes that a DR can evolve with a maximum depth of six is 63 (without `if>0` operator which takes three arguments), so the number of nodes in evolved rules is clearly smaller than the potential largest size.

We have also performed a second set of 50 GP runs under GP2 and ST2 with a maximum depth of eight. Increasing the maximum depth from six to eight increases the maximum number of nodes to 255, a huge increase in the size of the search space. Although the mean TWT attained on each test and extreme test instance was decreased (improved), it was not a statistically significant difference at the 5% significance level. The average size of the best-of-run DR increased to 71 nodes, twice the size of those evolved with a maximum depth of six. Examining rules evolved with this increase in maximum depth shows the ease of interpretability is definitely decreased. This is not to say that six is the optimal depth for DRs, especially as for the ATC and WCOVERT rules to be expressed in this form greater depth would be needed. However, perhaps this shows that including the size of DRs as an objective to be minimised alongside a more traditional measure of shop performance is a sensible direction to pursue. Parsimony pressure [141] is one frequently used method of constraining the size of GP individuals to prevent bloat.

STGP by nature may need a deeper maximum depth to compensate for the restriction imposed by the strong typing, since for the same maximum depth STGP is searching a much smaller search space. However, smaller DR size makes the DRs easier to analyse and understand.

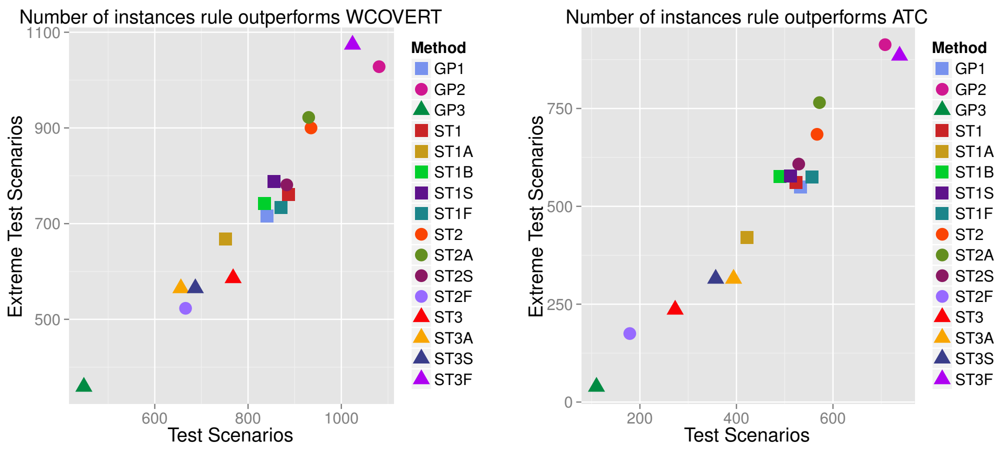


Figure 5.23: Number of instances DRs in Figures 5.7 to 5.22 that outperform WCOVERT (left) and ATC (right) on 50 problem instances from each test and extreme test scenario.

5.5.2 Comparison of Performance in Testing vs Extreme Testing

We evaluated the best-of-method rules shown in Figures 5.7 to 5.22 across 50 additional instances for each of the 24 test and 24 extreme test problem scenarios. Figure 5.23 shows how the rules performed in comparison to WCOVERT and ATC respectively, by plotting how many times the DR outperformed the benchmark rule out of 1200 ($= 50 \times 24$) possible test instances against how many times the DR outperformed the benchmark rule out of 1200 possible extreme test instances. Better performance is towards the top right corner of the plot. Figure 5.23 shows that the best DRs from GP2 and ST3S have the best performance, followed by ST2 and ST2A, then ST2F and all Stage 1 methods, the remaining Stage 3 ST methods, and lastly ST2S and GP3. It is interesting that the performance of the corresponding ST methods is worse for GP1, similar for GP2, and much better for GP3. The best performing rules are the ones evolved by GP2

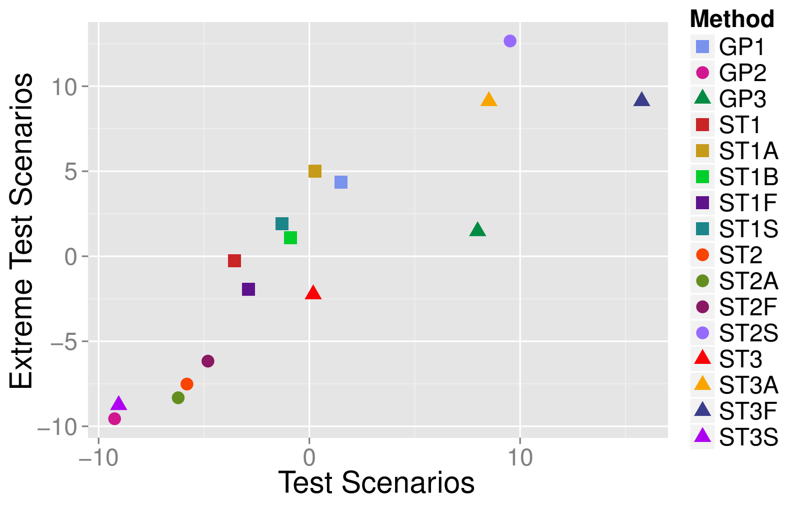


Figure 5.24: Mean Z-score of DRs in Figures 5.7 to 5.22 across 50 problem instances from each test and extreme test scenario.

and ST3S (Figures 5.13 and 5.21 respectively).

Further, we calculate Z-scores, $\frac{actual-mean}{standard\ deviation}$, where $actual$ is the performance of the given rule on a particular problem scenario, and $mean$ and $standard\ deviation$ are from all rules on the particular problem scenario, to measure the relative performance of each rule across all solutions for the 50 problem instances. We average these Z-scores across the 48 problem scenarios, and plot these values for test instances vs extreme test instances in Figure 5.24. *Better* performance is now *lower* Z-score values, i.e., towards the bottom left corner of the plots. Figure 5.24 is less linear than those in Figure 5.23, and GP2 and ST3S are still clearly better performers. However, the results of GP1, ST1, ST1A and ST1B appear more similar to those of GP3 and ST3 than in the comparisons of these rules with ATC and WCOVERT. The best rules from ST1F, ST1, ST1B, ST1S and ST3 have better mean Z-scores on both test and extreme test scenarios than GP1, ST1A and GP3. The worst Z-scores are attained by the best performing DRs from

ST2S, ST3F and ST3A. The Z-scores have separated out the performance of the best rules more than the comparisons with ATC and WCOVERT. We can see that although GP1 and GP3 attain the same performance on the extreme testing scenarios, GP3 is much worse than GP1 on test scenarios.

5.5.3 Training and Testing Times

Table 5.19 shows the mean±standard deviation of evolution time (in minutes) and testing and extreme testing time (in milliseconds) for each method. Table 5.19 shows that ST1, ST1A, ST1B, ST1S and ST1F methods have approximately half the mean evolution time and between three and four times shorter mean testing time than GP1, as well as much shorter evolution time. Table 5.19 shows that ST2 and ST2A methods have shorter mean evolution time and longer mean testing time than GP2. ST2S and ST2F have very similar testing and extreme testing time to GP2. ST3 has shorter mean evolution and mean testing times than GP3 as well as smaller standard deviations. However, ST3A, ST3S and ST3F have much longer evolution time, but shorter testing times. This may be due to the introduction of the specialised conditional operators, which lead to nearly triple number of conditional statements appearing in dispatching rules.

5.5.4 Exploration of Performance Decrease in STGP

To further explore the difference in performance between STGP and GP methods, we compare randomly generated trees, using the same comparison measure as used in Section 5.3. We obtain the average normalised TWT fitness value across the test and extreme test scenarios of each of 1024 randomly generated trees. We then average across all randomly generated trees from each method, obtaining values r_{GP1} for GP1, r_{ST1} for ST1, etc. We then calculate values comparing each GP method to its STGP counterparts:

$$r_{GP,ST} = \frac{r_{ST} - r_{GP}}{r_{GP}}$$

Table 5.19: Mean and standard deviation of evolution time and test time for GP and STGP.

	Mean±Stdev	Mean±Stdev	Mean±Stdev
	GP1	ST1	ST1A
Evolution (min)	2195.1 ± 1413.4	754.4 ± 398.4	588.0 ± 103.9
Testing (ms)	17753.5 ± 45558.3	7169.8 ± 3774.9	5515.5 ± 350.6
Extreme testing (ms)	12446.0 ± 8837.3	7595.46 ± 3875.9	5855.26 ± 382.3
	ST1B	ST1S	ST1F
Evolution (min)	692.8 ± 465.0	564.3 ± 176.1	501.7 ± 94.5
Testing (ms)	7010.1 ± 2884.3	5903.62± 1106.0	5970.8 ± 670.9
Extreme testing (ms)	7361.9 ± 3119.7	6395.4 ± 1483.7	6360.08 ± 675.1
	GP2	ST2	ST2A
Evolution (min)	1170.8 ± 46.5	842.8 ± 550.5	719.1 ± 409.6
Testing (ms)	5522.3 ± 344.6	8418.2 ± 3864.9	7143.6 ± 3305.8
Extreme testing (ms)	5866.0 ± 347.2	9018.42 ± 4181.2	7685.48 ± 3649.2
	ST2S	ST2F	
Evolution (min)	553.7 ± 97.1	526.84 ± 83.1	
Testing (ms)	5725.2 ± 577.8	5662.4 ± 497.5	
Extreme testing (ms)	6126.5 ± 586.1	6118.22 ± 492.2	
	GP3	ST3	ST3A
Evolution (min)	1408.8 ± 768.2	675.3 ± 280.6	9038.4 ± 1749.1
Testing (ms)	7206.6 ± 3513.3	5747.4 ± 2175.7	5183.2 ± 366.2
Extreme testing (ms)	7503.7 ± 3583.0	6284.2 ± 2341.1	5815.3 ± 1290.0
	ST3S	ST3F	
Evolution (min)	9354.0 ± 2311.3	7134.3 ± 2937.9	
Testing (ms)	5421.9 ± 525.9	6154.22 ± 1914.8	
Extreme testing (ms)	6120.7 ± 590.7	6625.9 ± 1970.8	

This gives us the following values, $r_{GP1,ST1} = 0.0051$, $r_{GP1,ST1A} = 0.0428$, $r_{GP1,ST1B} = 0.0388$, $r_{GP1,ST1S} = 0.2495$, $r_{GP1,ST1F} = 0.0708$. At Stage 2, $r_{GP2,ST2} = -0.0279$, $r_{GP2,ST2A} = -0.0699$, $r_{GP2,ST2S} = 0.2374$ and $r_{GP2,ST2F} = 0.2060$. At Stage 3, $r_{GP3,ST3} = 0.0694$, $r_{GP3,ST3A} = 0.0211$, $r_{GP3,ST3S} = 0.3457$ and $r_{GP3,ST3F} = 0.2019$. This shows that the initial populations are reasonably similar in terms of TWT performance across the test and extreme test scenarios. It also shows the validity of constraining the search space to se-

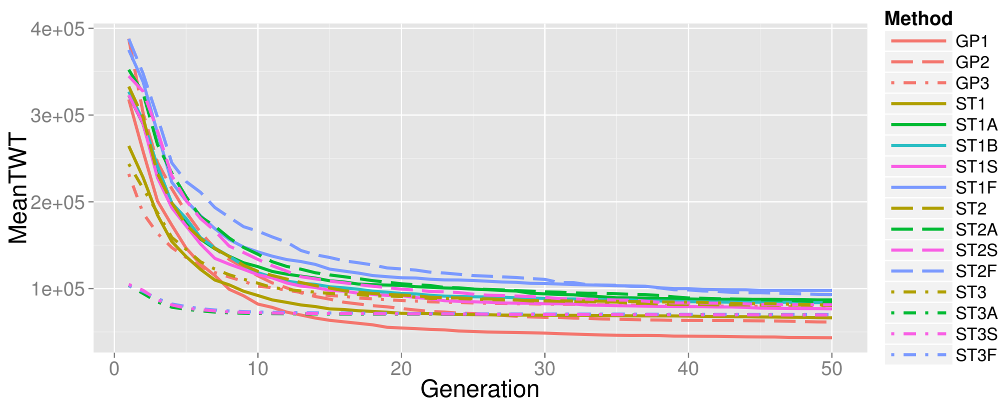


Figure 5.25: Mean best-of-generation fitness (mean normalised TWT) across all methods through 50 generations of evolution.

mantic correctness in that the randomly generated trees have very similar performance, and potentially the difficulty lies in the evolutionary process.

Figure 5.25 shows the mean best-of-generation fitness across all runs of each method, through the 50 generations of evolution. This shows that ST3A, ST3S and ST3F start with the lowest (best) mean fitness at generation 1, clearly lower than all other methods. However, around generation 12 GP1 overtakes, and then maintains the lowest mean fitness until the end of the evolutionary process at generation 50. The performance of all methods apart from GP1 form a band ranging from 60,000 to 100,000 at generation 50. It would be of interest to investigate why the performance of the ST3A, ST3S and ST3F methods plateaus so early compared to the other methods.

5.6 Chapter Summary

The goal of this chapter has been to use a grammar-based GP approach, using strongly typed GP, to semantically constrain the search space of possible heuristic dispatching rules, to evolve DRs with greater interpretability. This is the first work using GP for the automatic discovery of DRs that has explored the interpretability to this level. We investigated using three different function sets, and compared the best-of-run DRs evolved with and without semantic constraint.

The mean performance of DRs that were evolved under the type constraints of STGP was (as expected) not as good as the mean performance of DRs evolved without semantic constraint. However, there were still effective rules evolved under STGP. In fact, from the best performing rules of each method, the most effective DRs were evolved under semantic constraint. This highlights the need for multiple GP runs to establish performance of methods.

Our investigations led us to introduce new terminals, `1/PR` and `W/PR`, expanding the grammar for STGP to include a fifth type, I, and new functions `ifOD`, `ifPS` and `ifLO`. The use of conditional operators with predetermined conditions made the rules easier to interpret, due to the condition being tested being known. The conditions evolved, particularly without the use of a grammar, are often very long and are unable to be interpreted easily, if at all. Of these three new conditionals `ifPS` was the most frequently used. However, the performance of evolved DRs was not as good as that of DRs evolved with the `if>0` function in the function set. This is another trade-off between interpretability and TWT performance. We then introduce new terminals, `[PR*PR]` and `[1/[PR*PR]]`, expanding the grammar for STGP to include sixth and seventh types, S and F respectively.

We have found the interpretability of DRs evolved under STGP, and particularly the most effective one from the 50 best-of-run DRs, to be more easily understandable. Several of the rules are able to be understood as they break available jobs into categories based on if the job is overdue or not, and then different priority assignments are made based on this.

Interpretability is a difficult concept to quantify. The best we are able to do is manually inspect rules and see if, as human operators, we are able to explain a rule's effective or ineffective performance. Despite the small deterioration of mean TWT performance when the search space was semantically constrained, the best performing rules were better able to be understood. We believe that this improvement justifies the use of STGP to semantically constrain the evolved DRs, even at the cost of the observed slightly poorer (on average) performance (loss of effectiveness) from *insisting* on semantically valid models.

Further analysis of fragments appearing in the best-of-run rules revealed that the most common fragments are acceptable semantically even without semantic constraint. However, most of the evolved DRs contained at least one interaction which was not allowed under the corresponding grammar. This is an example of how GP is able to discover which terminals are more useful than others, and discover new useful terminals and combinations of terminals that could improve future work. A larger study with more independent GP runs will be made in Chapter 6 to discover building blocks that are usable as higher-level terminals.

The use of a grammar decreases the mean program size, as well as the standard deviation of evolved program size. The disadvantage of the use of a grammar is that there is a larger number of best-of-run DRs which have a poor performance in terms of TWT.

This chapter has provided greater insight into the interpretability of DRs, and what factors could potentially be used as part of a measure to quantify interpretability, allowing better comparison between evolved rules. A key component is the size of the evolved DRs, which could be measured in terms of depth, or in terms of the number of nodes. Further, the useful fragments that we observed appearing in multiple rules could suggest that a count of such useful fragments which do not violate a given grammar could also be a potentially useful measure. Although defining a function based on DR size and counting fragments or interactions would be a step closer to quantifying interpretability, actually defining a widely acceptable measure of interpretability *remains* a very big challenge.

Experimental results show that the dispatching rules evolved in the semantically constrained search space do not have (on average) performance that is as good as unconstrained. However, the interpretability of evolved rules is substantially improved. This is the first work using GP for the automatic discovery of dispatching rules that has explored their interpretability in depth and considered it as an important trait of an effective dispatching rule.

This chapter has been a first step investigation towards the long term goal of quantifying interpretability. The grammars used can be further expanded and refined, which may improve the effectiveness of evolved rules. In Chapter 6 we will use the insights from fragment analysis to add popular fragments as higher-level terminals and investigate a multiobjective approach, with objectives effectiveness of the DR, compactness of the DR and the number of distinct terminals featuring in the DR.

Chapter 6

Multiobjective Genetic Programming for Feature Selection and Dispatching Rule Generation in Job Shop Scheduling

6.1 Introduction

A difficulty in the automatic generation of DRs through GP is that the evolved DRs can be difficult for humans to interpret. Typically GP based approaches to discovering DRs have focused on the effectiveness of DRs developed in this way. However, we believe it is important to also focus on the interpretability of the DRs evolved, particularly if DRs discovered by GP are ever to be implemented in real manufacturing environments. Focusing on both the effectiveness of scheduling performance and interpretability of the DR turns the search for effective DRs into a multiobjective problem.

Chapter 5, showed the use of strongly typed GP [85] to enforce a grammar (restricting the allowable interactions within the DR) to be a useful means of constraining the search space, allowing the development of rules with greater interpretability, and enabling better understanding of what information effective rules are capturing from the shop system. Evolved DRs were competitive with those evolved without semantic constraint and, of equal importance, were much easier to interpret and understand.

Another factor affecting the interpretability of a DR is size; a smaller rule is generally much easier to understand than a larger rule. It is also known that simple rules are often not effective enough, thus GP is often used to generate more sophisticated rules. In this chapter we investigate the benefits of incorporating the size of the DR into the fitness of the DR.

One of the challenges of using a GP based approach is deciding what information from the domain should feed into the GP system through the terminal set. In JSS environments, there are many features of jobs, machines and the shop as a whole that *could* be useful as elements of the DR. However, we cannot put everything into the terminal set. Further, combinations of features may be more useful than a single feature by itself. This is an issue of how much domain knowledge we want to give the GP system. The size of the DR is also a factor in how quickly it is able to assign priority values.

A benefit of GP is that it may be able to discover domain knowledge. Geiger et al. [42] noted that the performance of scheduling rules can be of secondary importance to the "interrelationships of the attributes within the rule, and their relevance to the problem structure". This information can be exploited in future rule design.

Although GP has been used for feature manipulation [77, 91, 92] across a wide variety of machine learning applications, GP has not previously been used for feature manipulation for job shop scheduling. Often within a feature set, there are a very large number of available features, creating a very large search space, or some features are made redundant by the preference for other features. Reducing the feature set size, through feature selection, reduces the search space (and potentially improves the learning performance), and can potentially make the evolved DRs easier to interpret.

In this investigation the number of distinct terminals appearing in a DR is also considered as a third objective to be minimised. Minimising the number of distinct terminals which appear in the rule is a form of implicit feature selection.

Feature construction [77, 92] will be performed by using the evolved DRs discovered by GP to propose new higher-level terminals, which will be added into the feature set.

This is not an investigation into the development of a new GP algorithm, but analysis of how we can better use GP for automatic generation of DRs. While scheduling is clearly a multiobjective problem, we believe that DR performance is also multiobjective; performance in terms of the scheduling objectives is important but so is the interpretability of DRs and the trust that practitioners are able to place in DRs. Further, the knowledge discovered by GP through the terminals selected can be viewed as feature selection and can potentially be used to alter the terminal set.

6.1.1 Chapter Goals

This chapter aims to use multiobjective genetic programming (MOGP) [114] to investigate how the ability to discover interpretable DRs can be improved by the inclusion of objectives relating to the size of the DR and through feature manipulation guided by GP. We want to study the tradeoffs in performance and interpretability of DRs evolved using MOGP. We expect them to be more interpretable due to their smaller size, and closer to manually designed rules from the literature in terms of ease of interpretation. The objectives for this chapter are:

1. Develop an MOGP method encouraging the evolution of more understandable DRs for the dynamic job shop environment.

2. Investigate whether interpretability can be improved by the inclusion of objectives relating to the size of the DR.

3. Explore which of the two MO algorithms, NSGA-II [32] and SPEA2 [146], performs better for GP to evolve interpretable rules, compared with those evolved with single objective GP, and existing DRs from the literature.

4. Investigate whether the restriction of the search space through STGP reveals good components or combinations of components that we can use to further restrict or alter the terminals included in our terminal set.

5. Investigate how feature manipulation guided by GP can improve interpretability.

6.1.2 Chapter Organisation

The remainder of this chapter is organised as follows. The proposed MO-GP approach and parameter settings are described in Section 6.2. Section 6.3 presents the first stage of this chapter using an initial terminal set. Section 6.4 presents the second stage of our investigation where the terminal and function sets will be extended to include higher-level constructed terminals. Section 6.5 presents the third stage of our investigation where three specialised conditional operators will be added to the function set, following Chapter 5. Section 6.6 presents the final stage where the results of the second and third stages will be used to perform feature reduction to see whether the performance of GP may be be further improved by removing infrequently used terminals from the terminal set. Frequency analysis is performed on the evolved DRs of the first two stages to determine which terminals are less useful. Further analysis and comparison of results is performed in Section 6.7. Section 6.8 draws conclusions.

6.2 MOGP for Evolving Interpretable DRs

This section describes the proposed multiobjective genetic programming method for evolving dispatching rules for the dynamic job shop environment. First the implementation of STGP to semantically constrain DRs for interpretability will be described, with the function and terminal sets and grammar of allowable interactions. Then the job shop simulation model used for training and testing dispatching rules will be described.

6.2.1 Representation of Semantically Constrained DRs

In this chapter we are using a multiobjective approach, seeking to find a front of non-dominated DRs across the objectives used. In each problem instance the objectives of interest are calculated, as described below. At the final generation of evolution, the Pareto front of non-dominated DRs is taken and then tested on independent test problem instances.

Function Set

The GP function set contains four arithmetic operators, $+$, $-$, $*$ and $\%$, which each take two arguments, and `if>0`, `max`, `min`. These functions are as defined in Section 4.2.1 (see page 107).

Terminal Set

The GP terminal set is shown in Table 6.1, following from Chapter 4 see Table 4.1 (on page 108). The terminals are split into features of the particular job being considered, the machine at which the scheduling decision is being made, and features of the wider shop system. `RJ` is the release time of the job, or the time its last completed operation finished. `NPR`, `NNQ` and `NQW` will return zero if the job is on its final operation. If fewer than five jobs have visited a machine, then `NQW` and `AQW` return the average wait time of the jobs which have been visited.

Table 6.1: Terminal set used in the GP system with type.

Feature	Symbol	Type
Job Properties		
Processing time of operation	`PR`	D
Remaining processing time of job	`RT`	D
Remaining number of operations	`RO`	C
Ready time of job	`RJ`	T
Due date of job	`DD`	T
Weight of job	`W`	X
Inverse processing time of operation	`[1/PR]`	I
Next operation's processing time	`NPR`	D
Number of jobs in queue at the next machine job visits	`NNQ`	C
Machine Properties		
Ready time of machine	`RM`	T
Number of jobs in queue	`NQ`	C
Average wait time of last five jobs processed at the next machine job visits	`NQW`	D
Average wait time of last five jobs processed	`QW`	D
Shop Properties		
Current time	`CT`	T
Average wait time of last five jobs processed across all machines in the shop	`AQW`	D

Strongly Typed Genetic Programming

One key focus of this work is to evolve dispatching rules that are more interpretable to human operators. The allowable interactions of terminals in GP individuals are defined using the strongly typed functions of ECJ. The grammar shown in Table 5.3 (see page 171) will be used. Recapping from Section 5.2.3 (see page 166), this grammar is based on splitting the terminal set into five different type categories as in Table 6.1; counts, weights (or ratios), time durations, absolute (clock) times, and inverse time durations. Recall that due to the limitations of the STGP implementation of ECJ20 [78], the inclusion of a terminal of this type is required, hence `[1/PR]` is part of the terminal set shown in Table 6.1 as well as `PR`. This grammar has been implemented to prohibit interactions which do not make sense semantically, e.g., adding or subtracting a count terminal to a time terminal (or certainly does not make *as much sense* as multiplying or dividing). The allowable interactions of arithmetic operators for this grammar were shown in Table 5.4 (see page 173).

6.2.2 Multiobjective Fitness

Throughout the evolutionary process, the multiobjective fitness of a DR consists of one value for each of the following three objectives.

TWT: the average of the objective function, *TWT*, normalised by expected utilisation, over all problem scenarios used.

NumNodes: the size of the DR, calculated as the number of nodes in the DR.

NumDistinct: the number of distinct terminal nodes appearing in the DR.

With the complex terminals, we count each function and terminal within the complex terminal, e.g., `[1/PR]` counts as three nodes. Both NumNodes and NumDistinct do not rely on discrete-event simulation, but can be calculated (counted) directly from the DR.

6.2.3 Job Shop Simulation Model

In this section, we randomly create problem instances of a ten-machine job shop, using the method described in Section 4.2.2, (see page 110). Each job has equal probability of being processed at each machine in the shop, leading to each machine having the same expected utilisation (on average throughout the simulation). This means that the job shop is symmetrical. We randomly generate scenarios using the properties given in Table 5.1 (see page 164) to determine the arrival rate using Equation (4.1) (see page 110). Due dates are assigned using Equation (4.2) (see page 110).

Training

In training, job operation processing times follow a discrete Uniform(1,49) distribution (with mean 25). Jobs are given weight 1, 2 or 4, with probability $(0.2, 0.6, 0.2)$ following [111]. Jobs are assigned a due date tightness parameter randomly chosen from $\{3, 5, 7\}$. Jobs also have either "full" or "variable" operations. With "full" operations, each job has an operation at every machine in the shop; with the "variable" setting the number of operations per job is a random integer between two and ten. These settings are summarised in Table 5.1 (see page 164). A warm up period of 500 jobs is used, and we collect data from the next 2000 jobs to arrive ($N = 2000$). New jobs continue to enter the system until the completion of the 2500th job.

Testing

The Pareto front of non-dominated DRs at the end of the evolutionary process is tested on ten-machine job shop scenarios with the parameter settings of Rajendran and Holthaus [117]. Operations have processing times which are randomly drawn from the discrete Uniform(1,49) distribution. Jobs are assigned an importance weighting of 1, 2 or 4, with probabilities of 0.2, 0.6 and 0.2 respectively [111]. There are three levels of due date tightness parameter, 4, 6 and 8, which are used for every job in the given scenario. There are four levels of machine utilisation, 80%, 85%, 90% and 95%. Jobs have either "full" or "variable" operations as in training. There are therefore 24 combinations of due date tightness, machine utilisation and operation type, the resulting 24 scenarios which are given in Table 5.1 (see page 164).

Extreme Testing

As in Chapter 5 we use an additional set of *extreme test* instances to test the generalisation ability of the Pareto front of non-dominated DRs. Once again this uses a geometric distribution with mean $\mu = 25$, job weights of 1, 2, 4 or 8 with probability $(0.2, 0.5, 0.2, 0.1)$, and the due date tightness parameter is equally likely from {4,6,8}, {3,5,7}, or {3,4,5}. The 24 extreme test scenarios are given in Table 5.1 (see page 164).

Result Normalisation

In order to summarise the performance of the Pareto front of non-dominated DRs across all 48 test and extreme test scenarios, for ease of presentation, we normalise each and take the average of test scenarios and the average of extreme test scenarios. The fitness of a DR on scenario i is normalised by

$$f_{norm,i} = \frac{f_i - f_{min,i}}{f_{max,i} - f_{min,i}} \tag{6.1}$$

where $f_{min,i}$ is the minimum TWT value attained on scenario i across *all methods*, and $f_{max,i}$ is the maximum TWT value attained on scenario i across *all methods*. This gives two summary fitness values:

$$f_t = \tfrac{1}{24} \sum_{i=1}^{i=24} f_{norm,i} \qquad f_{xt} = \tfrac{1}{24} \sum_{i=25}^{i=48} f_{norm,i} \tag{6.2}$$

and one overall fitness value

$$f_{TWT} = \tfrac{1}{2} \left(f_t + f_{xt} \right) \tag{6.3}$$

These normalised values will help us to compare the performance of the non-dominated fronts from MOGP with NSGA-II to those from MOGP with SPEA2 at each stage of experiments as feature manipulation is performed, and compare the MOGP results to those attained under SOGP.

GP System Parameters

The initial population is generated using the ramped-half-and-half method [70] with a minimum depth of two and maximum depth of six. The population size for NSGA-II is 200, and for SPEA2 is 300 with an elitist archive of size 100 following the ECJ20 guidelines, as the implementation of NSGA-II builds the archive separately whereas SPEA2 uses the specified portion of its population as the archive. Evolution is for 50 generations. GP trees have a maximum depth of six. Two genetic operators are used, namely crossover and mutation with rates 90% and 10% respectively. These rates have been previously used [100].

6.2.4 Benchmark Methods for Comparison

Single objective GP (SOGP) is used as a benchmark method to compare and examine the benefit of using the multiobjective approach. The fitness function used is mean TWT (the first objective of the multiobjective approach). SOGP uses the genetic operators crossover, mutation and elitism, with rates of 85%, 10% and 5% respectively. Tournament selection with a tournament size of seven is used to select individuals for genetic operators. This is a common setting that has been previously used [95]. At the end of the evolutionary process we are able to calculate the value of NumNodes and NumDistinct directly from the best-of-run DR, counting the number of nodes and the number of distinct terminal nodes. We can then compare the results of both MOGP approaches to the best-of-run DRs of the benchmark method, across all three objectives.

We will also compare the results obtained to the results obtained with DRs from the literature. The benchmark DRs are those discussed in Section 2.2.3 (see page 40): FCFS, EDD, MS, WSPT, ATC and WCOVERT.

6.3 Stage One: Initial Investigation

The first stage of this chapter will use the terminal set described above. We will refer to these methods as $MOGP_{1,N}$ when NSGA-II is used, $MOGP_{1,S}$ when SPEA2 is used, and $SOGP_1$ for the benchmark SOGP method with this terminal set.

In this section the results of the experiments are presented. Here, 500 independent runs of each MOGP method are performed, and the evolved Pareto fronts of non-dominated DRs obtained are recorded for each. We also perform 500 independent runs of the benchmark SOGP methods. Usually 30 or 50 GP runs would be sufficient when comparing average performances between methods, but as we are counting occurrences of nodes in the evolved trees, many more runs are needed to compare the counts between methods.

Figure 6.1 gives plots of each pair of the objectives TWT, NumNodes and NumDistinct on the test and extreme test instances, and Figure 6.2 plots f_t vs f_{xt} for each evolved DR, using NumNodes as the plotting symbol on the test and extreme test instances. Immediate observations are that the DRs with the best TWT performance have between five and eight distinct terminals, and have between 15 and 40 nodes. Almost all of the DRs evolved with MOGP have less than 50 nodes, and most have less than 30 nodes. In comparison SOGP evolved several DRs with over 100 nodes and there are many with more than 50 nodes. In the plot of TWT vs NumNodes, it is clear that for similar TWT values, there are a large number of MOGP evolved DRs that have smaller size than DRs evolved by SOGP. Note also that the MOGP methods produce DRs with only one distinct terminal, whereas the minimum NumDistinct for SOGP_1 is three. The mean NumNodes of DRs which attain TWT<0.01 is 39.9, 24.3 and 22.7 for SOGP_1, $\text{MOGP}_{1,N}$ and $\text{MOGP}_{1,S}$, although SOGP_1 had more DRs attaining this level of TWT performance (24 compared to 19 from $\text{MOGP}_{1,N}$ and 6 from $\text{MOGP}_{1,S}$). This shows that the MO approach is leading to smaller sized DRs, which is likely to mean more easily interpretable DRs.

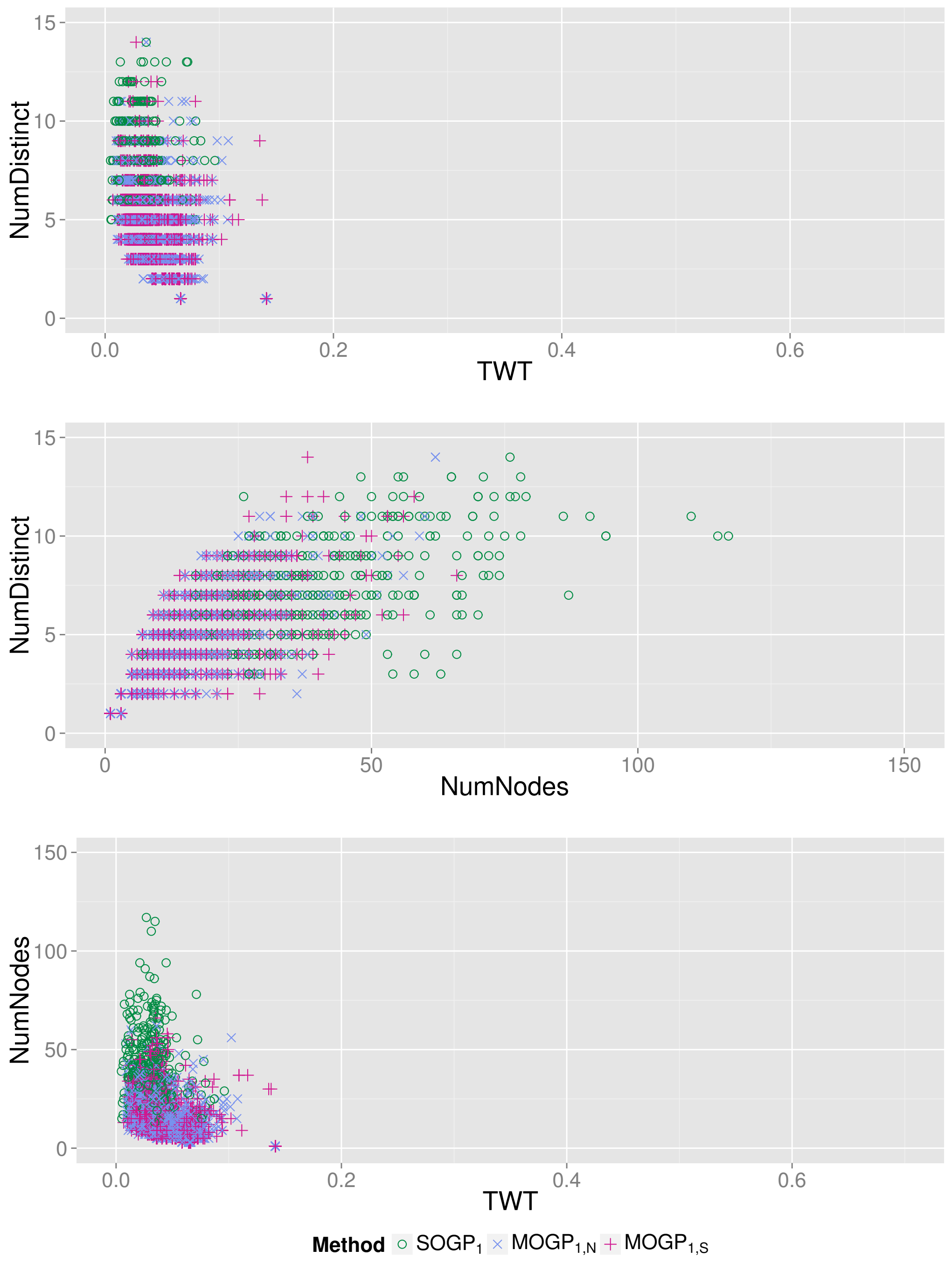


Figure 6.1: Paired plot of Stage 1 results. This plot contains the Pareto fronts of 500 runs for all methods, 5188 DRs (2495 distinct DRs) from MOGP$_{1,N}$, 4640 DRs (1982 distinct DRs) from MOGP$_{1,S}$ and 500 DRs from SOGP$_1$.

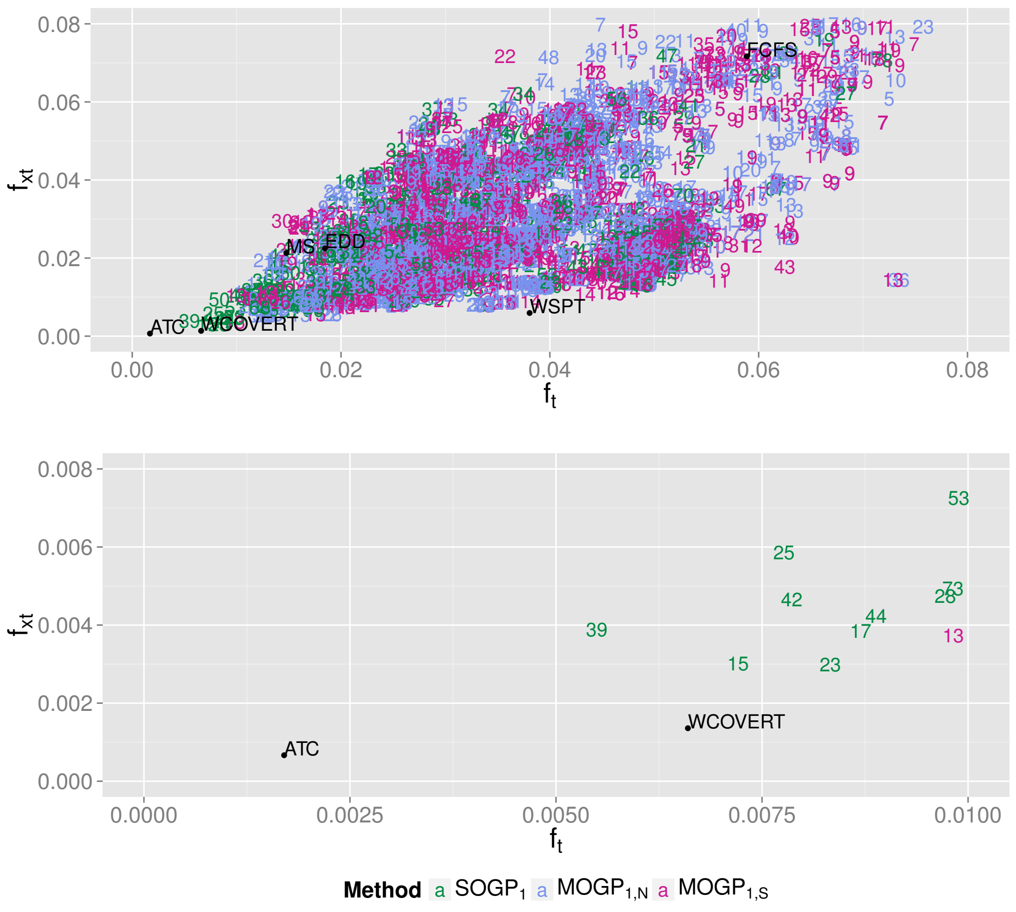


Figure 6.2: Plot of normalised TWT over test instances vs extreme test instances. The bottom plot zooms in on the area of best test and extreme test instances.

From Figure 6.2 we can see that the DR with the lowest TWT was evolved by SOGP$_1$, and there are a number of rules with performance similar to that of WCOVERT. However, FCFS is outperformed by the majority of evolved DRs. MS, EDD and WSPT have better performance, and interestingly WSPT performs better on the extreme test instances than on the test instances, whereas MS and EDD perform better on the test instances than on the extreme test instances.

The average $\pm$ standard deviation front size of GP runs using MOGP$_{1,N}$ is 10.4$\pm$4.2, and using MOGP$_{1,S}$ is 9.3$\pm$3.8. This is calculated after duplicate rules have been removed from the Pareto front.

6.4 Stage Two: Higher-level Constructed Terminals

In the second stage of our investigation, we extend the terminal and function sets to include higher-level constructed terminals. This involves feature construction, reliant on fragment analysis to find the most frequently occurring two level fragments in the DRs that make up the Pareto front of non-dominated individuals from the 500 GP runs of both *MOGP*$_{1,N}$ and *MOGP*$_{1,S}$. These terminals will incorporate domain knowledge as discovered by GP. Increasing the size of the terminal set will increase the size of the search space. These methods will be referred to as *MOGP*$_{2,N}$, *MOGP*$_{2,S}$ and *SOGP*$_2$.

In this stage we perform feature construction, based on the results of Section 6.3. The most frequently occurring fragments in the evolved Pareto fronts of DRs for MOGP$_{1,N}$ and MOGP$_{1,S}$ and the evolved DRs for SOGP$_1$ are shown in Table 6.2. This table combines both possible orders of the arguments of functions, i.e., $*${`W`,`NNQ`} includes both ($*$ `W` `NNQ`) and ($*$ `NNQ` `W`). However, fragments that are essentially the same, e.g. ($*$ `W [1/PR]`) and (% `W` `PR`), have not been combined.

From Table 6.2 we observe that the same five fragments take up the top five places. Further, the fragments which take the difference between `RM` or `CT` and `DD` are producing very similar results, as most scheduling decisions are made when the machine becomes available, hence `RM`=`CT` (for this to not be true, multiple jobs must arrive at the same time at a machine which is currently idle). Therefore we choose just one of these two to be one of our higher-level constructed terminals. We also note that ($*$ `W [1/PR]`) gives the very popular weighted shortest processing

Table 6.2: Stage 1. Top six most frequently appearing fragments in Pareto-fronts of MOGP$_{1,N}$ and MOGP$_{1,N}$, and top six most frequently appearing fragments in best-of-run DRs of SOGP$_1$ at Stage 1. The number of times the fragment appears and the number of rules the fragment appears in are given for each.

MOGP$_{1,N}$		
Fragment	Frequency	Rules
% {W,NNQ}	1355	1242
% {NNQ,[1/PR]}	1051	938
− {DD,RM}	518	490
∗ {W,[1/PR]}	518	456
− {DD,CT}	465	414
% {NQ,[1/PR]}	215	163
Total DRs		5188
MOGP$_{1,S}$		
Fragment	Frequency	Rules
% {W,NNQ}	1190	1070
% {NNQ,[1/PR]}	856	755
− {DD,CT}	495	451
∗ {W,[1/PR]}	467	416
− {DD,RM}	431	402
+ {[1/PR],[1/PR]}	122	115
Total DRs		5188
SOGP$_1$		
Fragment	Frequency	Rules
∗ {W,[1/PR]}	188	127
% {NNQ,[1/PR]}	110	79
% {W,NNQ}	89	64
− {DD,CT}	83	63
− {DD,RM}	81	59
% {NQ,[1/PR]}	78	54
Total DRs		500

Table 6.3: Higher-level constructed terminals introduced at Stage 2.

Feature	Symbol	Type	Number of Nodes
(% W NNQ)	[W/NNQ]	X	3
Weighted number of jobs at next machine			
(− DD CT)	[DD−CT]	D	3
Time until due date			
(% NNQ [1/PR])	[NNQ*PR]	D	3
Estimate of work in next queue			
(* W [1/PR])	[W/PR]	I	3
Weighted processing time			
(* (% W NNQ) [1/PR])	[W/[PR*NNQ]]	I	5
Weighted estimate of work in next queue			

time (WSPT) dispatching rule (see page 41). We also searched for fragments containing three terminals, i.e., three level fragments where one argument of the top level function is a terminal. Amongst these fragments the most commonly occurring fragment gives us the fifth higher-level terminal, `[W/[PR*NNQ]]`. When the various ways this fragment could be formed were combined, this fragment appeared over 500 times in MOGP$_{1,N}$. It is interesting to note that three of the other higher-level terminals, `[W/PR]`, `[NNQ*PR]` and `[W/NNQ]`, are contained in this fragment. If we think of the fragment `[NNQ*PR]` as an estimate of the work in the next queue the job visits, then `[W/[PR*NNQ]]` is the weighted estimate of work in the next queue.

The terminal set was augmented with the five new higher-level constructed terminals shown in Table 6.3. Note that of the constructed terminals (see Table 6.3), the first four are deemed to contribute three nodes to the NumNodes and `[W/[PR*NNQ]]` is deemed to contribute five nodes. This ensures the validity of comparisons of NumNodes with Stage 1.

Figure 6.3 presents a plot of each pair of the objectives TWT, NumNodes and NumDistinct. Note that the colours of the legends are intentionally changed from Figure 6.1.

Immediate observations from Figure 6.3 are that the DRs with the best TWT performance have between four and seven distinct terminals, and have size between 10 and 40 nodes. Almost all of the DRs evolved with MOGP have less than 60 nodes, and most have less than 40 nodes. In comparison one SOGP evolved DR has over 140 nodes and a fifth have more than 50 nodes. Once again the minimum NumDistinct from SOGP is three.

Figure 6.4 plots f_t vs f_{xt} for the best performing DRs at Stage 2. Here, 14 DRs attain values of f_t and f_{xt} that are less than 0.05, five of these are evolved by SOGP$_2$, three by MOGP$_{2,S}$ and the rest by MOGP$_{2,N}$. The DRs evolved by SOGP$_2$ are, on average, twice the size of those evolved by MOGP. In Figure 6.4 we can see that there are more DRs which are outperforming WCOVERT (by normalised value) on test instances, including DRs evolved by all three methods. There are several rules evolved by SOGP$_2$ which outperform WCOVERT on both f_t and f_{xt}.

The average $\pm$ standard deviation Pareto front size of GP runs (calculated after duplicate rules have been removed from the Pareto front) using MOGP$_{2,N}$ is 12.6$\pm$5.1, and using MOGP$_{2,S}$ is 13.2$\pm$6.1. In both cases this is a larger average front size than their Stage 1 counterparts. The increase in mean front size is greater from MOGP$_{1,S}$ to MOGP$_{2,S}$ than from MOGP$_{1,N}$ to MOGP$_{2,N}$.

6.5 Stage Three: Specialised Conditional Operators

In this section we include three specific conditionals: `ifOD`, `ifPS` and `ifLO`. These were introduced in Chapter 5 (see page 187) because the conditions evolved for the `if>0` operator, especially without semantic constraint, were frequently observed to be very long and unable to be easily interpreted. The use of the specialised conditional operators with prede-

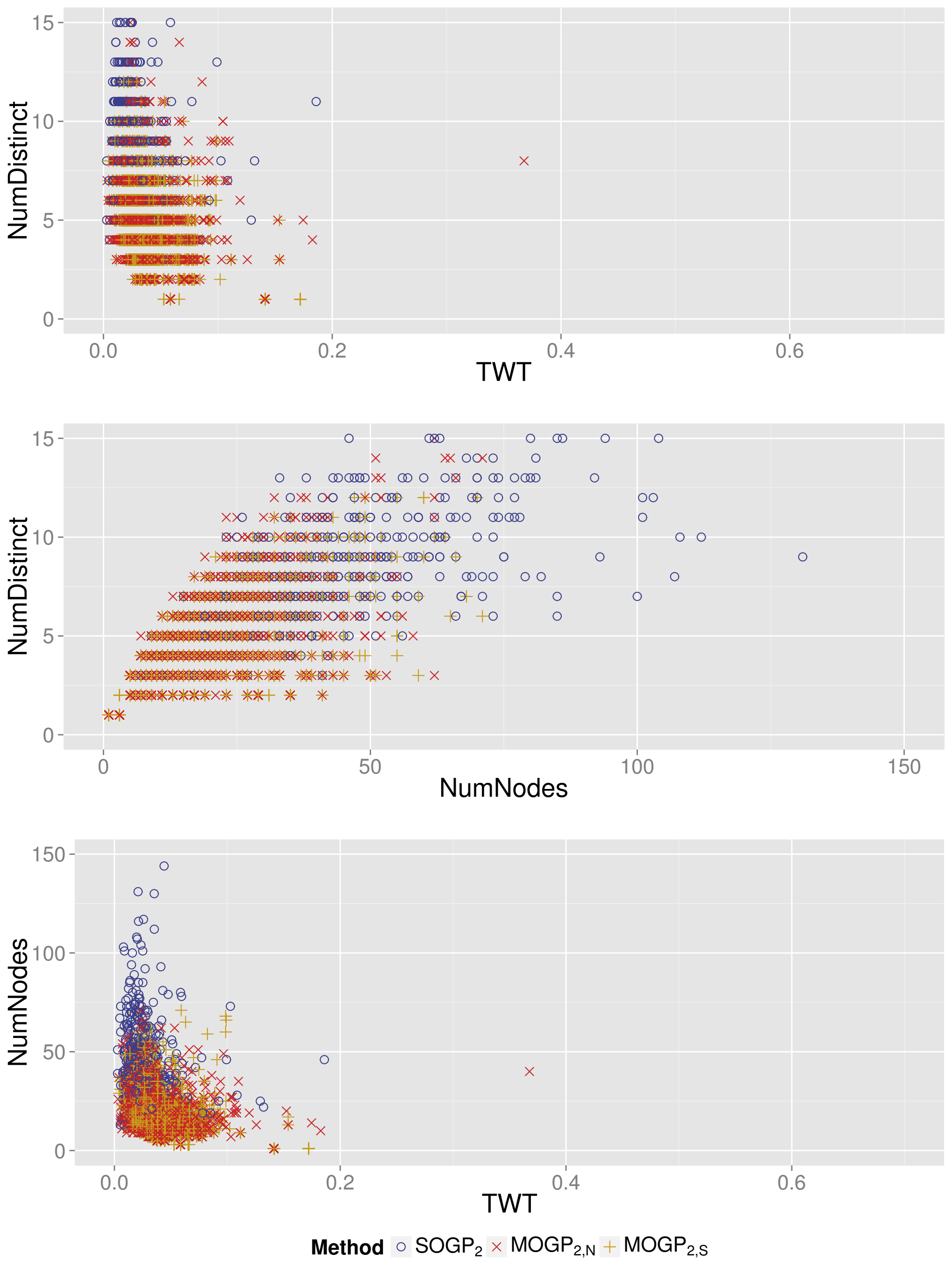


Figure 6.3: Paired plot of Stage 2 results. This plot contains the Pareto fronts of 500 runs for all methods, 6299 DRs (2885 distinct DRs) from MOGP$_{2,N}$, 6585 DRs (2268 distinct DRs) from MOGP$_{2,S}$ and 500 DRs from SOGP$_2$.

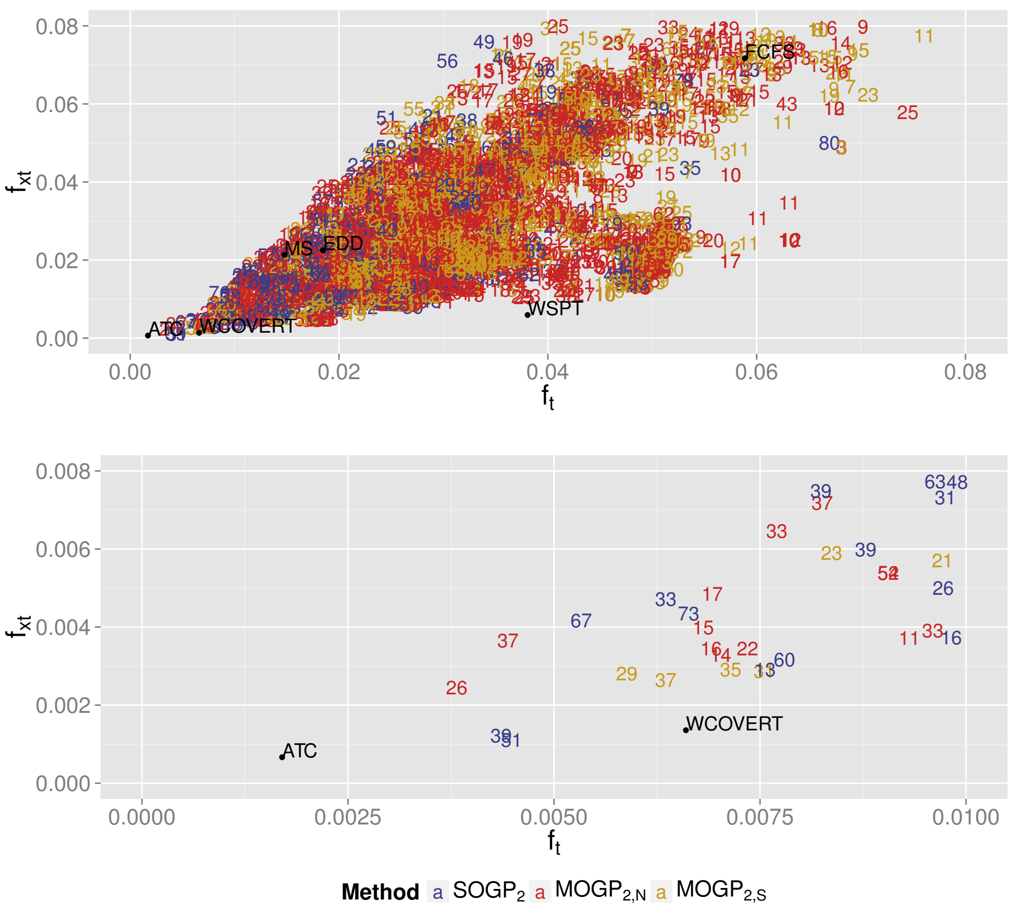


Figure 6.4: Plot of normalised TWT over test instances vs extreme test instances. The bottom plot zooms in on the area of best test and extreme test instances.

termined conditions made the rules easier to interpret; however performance of the evolved DRs was not as good as that of DRs evolved with the `if>0`. This highlights the trade-off between interpretability and TWT performance of DRs evolved using GP. These methods will be referred to as $MOGP_{3,N}$, $MOGP_{3,S}$ and $SOGP_3$.

Each of the specialised conditional operators has a fixed condition, and take two arguments. The `ifOD` operator returns the first argument if the job is overdue ($\texttt{DD} \leq \texttt{CT}$), else the second argument is returned. This terminal contributes four to the NumNodes. The `ifPS` operator returns the first argument if the job has non-negative slack and it is possible for the job to be completed before its due date, i.e., $\texttt{DD} - \texttt{CT} - \texttt{RT} \geq 0$, and returns the second argument otherwise. This terminal contributes six to the NumNodes. The final new conditional operator, `ifLO`, returns the first argument if it is the last operation of the job, or the second argument if it is not. This terminal contributes four to the NumNodes.

For Stage 3 results, Figure 6.5 present a plot of each pair of objectives. From Figure 6.5 we can observe that the DR with the best TWT performance is evolved by SOGP$_3$. The top 15 evolved DRs across all methods at Stage 3 have mean normalised TWT under 0.005. Four of these were evolved by SOGP$_3$, three by MOGP$_{S,3}$ and six by MOGP$_{N,3}$. The largest DR has 67 nodes and 8 distinct terminals, the other top DRs have between 4 and 6 distinct terminals and between 15 and 44 nodes (with mean 27.31).

All of the DRs evolved with MOGP have less than 30 nodes, and most have less than 20 nodes. There are many SOGP evolved DRs with between 30 and 50 nodes. In the TWT vs NumNodes plot it is clear that for similar TWT values, there are a large number of MOGP evolved DRs that have smaller size than SOGP DRs, except for the best performing SOGP DR.

Figure 6.6 plots f_t vs f_{xt} for Stage 3. We can see that the DRs which attain the lowest f_t and f_{xt} are evolved by SOGP$_3$. However of DRs with $0.0 < f_t, f_{xt} < 0.005$ the MOGP methods have mean size half that of those of SOGP$_3$. We note that there are even more DRs with better f_t values than WCOVERT than in Stage 2, however ATC is still the clear best performer overall.

The average $\pm$ standard deviation Pareto front size, after duplicate rules have been removed from the Pareto front, of GP runs with MOGP$_{3,N}$ is 17.1$\pm$6.0, and with MOGP$_{3,S}$ is 14.8$\pm$5.8. Once again these show an increase in average front size from the previous stage of experiments.

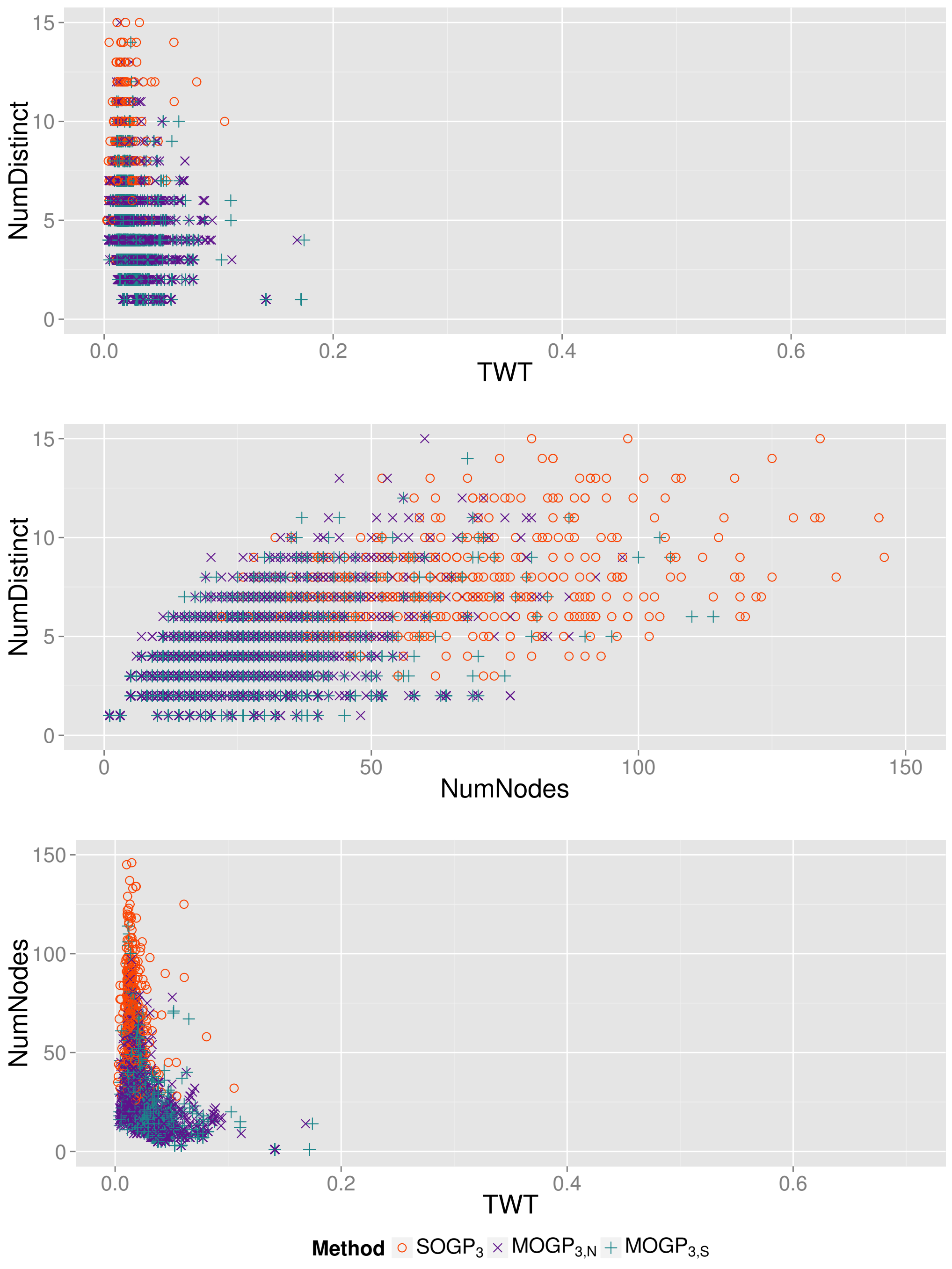


Figure 6.5: Paired plot of Stage 3 results. This plot contains the Pareto fronts of 500 runs for all methods, 8535 DRs (3159 distinct DRs) from MOGP$_{3,N}$, 7376 DRs (2280 distinct DRs) from MOGP$_{3,S}$ and 500 DRs from SOGP$_3$.

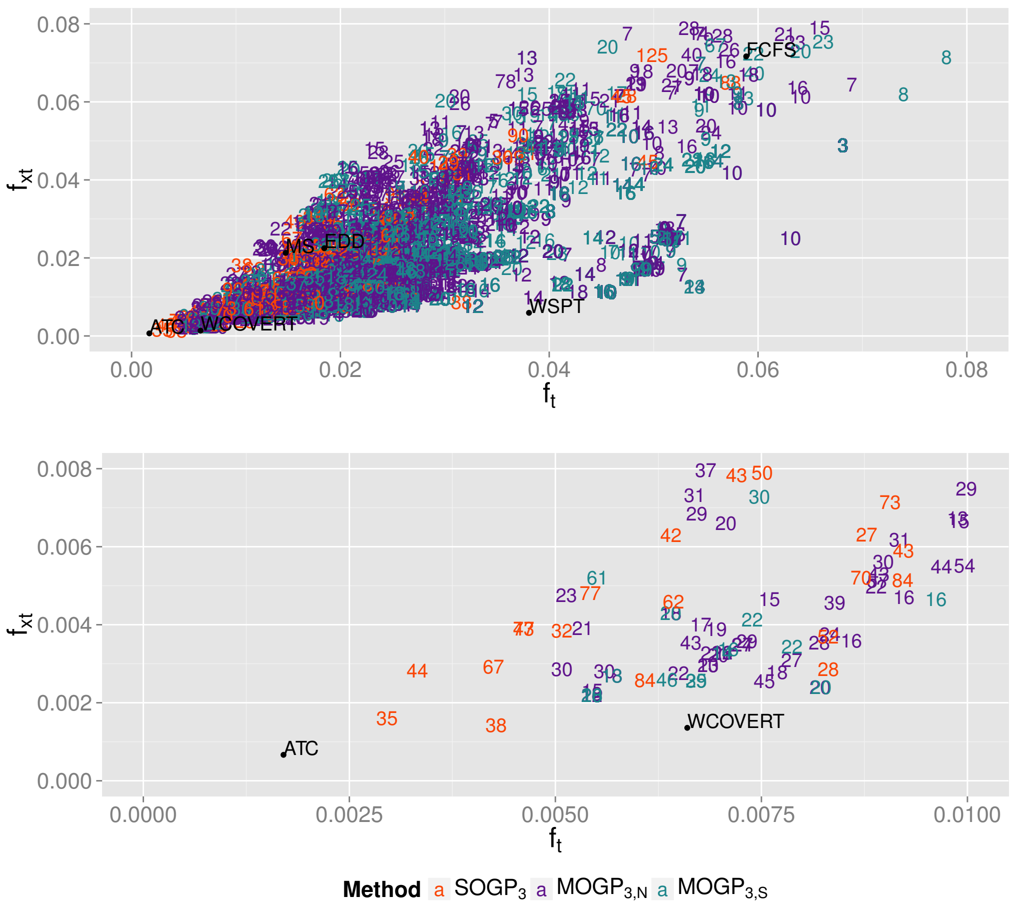


Figure 6.6: Plots of normalised TWT over test instances vs extreme test instances for Stage 3. The bottom plot zooms in on the area of best test and extreme test instances.

6.6 Stage Four: Feature Reduction

This section presents the final stage of our experiments in this chapter. We will use frequency analysis of the evolved DRs of the second and third stages to perform feature reduction. Features which are used infrequently are removed from the terminal set. This reduces the size of the search space and may improve the search, and hence the performance of GP can be improved. The methods with the reduced feature sets in this stage will be referred to as $MOGP_{4,N}$, $MOGP_{4,S}$ and $SOGP_4$.

Table 6.4 presents the frequency of functions and terminals in Stages 1 to 3 (and Stage 4 for comparison). For each method, the percentage of times the given function is used out of all function occurrences is given, together with the percentage of times the given terminal is used out of the total number of terminal occurrences. Based on these results we further adjust the terminal set, removing the terminals which appear infrequently across all experiments. The inclusion of the NumDistinct objective (to be minimised) means that if two DRs attain the same TWT performance, the DR with fewer distinct terminals will dominate, encouraging terminals which are not as useful to appear less frequently in the non-dominated Pareto front. `AQW` and `NQW` never appear as more than 2% of the terminals. Even `NPR` does not appear very often. We initially introduced these terminals to help decrease the "myopic" nature of dispatching rules (see Chapter 4). This frequency analysis however shows that the `NNQ` terminal, which gives the number of waiting jobs at the next machine the job visits, is very popular. It also occurs very frequently via the terminal `[W/[PR*NNQ]]`, which is an approximation of the work in the queue at the next machine, suggesting that this may be a more useful measure of the availability of the next machine than the average waiting time there. Also, `[NNQ*PR]` doesn't appear very frequently, which suggests that although it is a frequently appearing two level fragment in Stage 1 experiments, it is more useful as part of `[W/[PR*NNQ]]` than on its own.

The observations made above lead us to remove terminals that do not reach more than 3% usage (on average) across Stages 1 to 3. This gives the following terminal set for our fourth and final set of experiments: {`PR`, `RO`, `RM`, `DD`, `W`, `NQ`, `QW`, `CT`, `NNQ`, `[1/PR]`, `[W/PR]`, `[W/NNQ]`, `[DD-CT]`, `[W/[PR*NNQ]]`}. We also remove the `if>0` and `max` functions. This aims to see if removing the functions and terminals which seem to be "less useful" can improve the efficiency and effectiveness of the DRs evolved

Table 6.4: Occurrence of each function as a percentage of all function occurrences in DRs evolved by each method, and occurrence of each terminal as a percentage of all terminal occurrences in DRs evolved by each method.

	$M_{1,N}$	$M_{1,S}$	S_1	$M_{2,N}$	$M_{2,S}$	S_2	$M_{3,N}$	$M_{3,S}$	S_3	$M_{4,N}$	$M_{4,S}$	S_4
$+$	9.0	8.5	10.3	17.0	38.9	10.9	14.6	25.4	8.0	13.8	26.9	9.3
$-$	21.0	19.3	15.2	25.0	18.0	20.0	4.7	4.0	6.0	9.0	6.2	9.3
$\star$	25.0	25.4	25.3	24.7	18.9	22.6	33.5	27.8	25	29.7	24.4	25.3
%	34.7	36.6	20.0	18.8	13.7	16.9	13.2	10.0	13.2	16.9	13.1	17.3
`max`	3.7	3.1	8.7	3.8	2.4	9.5	1.9	1.3	6.6	–	–	–
`min`	3.8	3.9	9.4	4.5	3.3	9.0	3.7	2.8	7.1	4.1	3.3	8.0
`if>0`	2.8	3.3	11.1	6.2	4.7	11.0	1.1	1.1	6.2	–	–	–
`ifPS`	–	–	–	–	–	–	21.9	22.0	14.9	18.8	19.1	15.4
`ifOD`	–	–	–	–	–	–	3.4	3.8	5.7	4.6	4.5	6.9
`ifLO`	–	–	–	–	–	–	2.1	1.7	7.1	3.1	2.4	8.5
`PR`	15.9	16.1	16.8	13.4	15.8	12.6	17.7	17.8	18.1	17.5	18.8	16.6
`RT`	1.6	1.2	2.9	1.2	0.7	2.4	0.5	0.2	0.9	–	–	–
`RO`	1.2	1.3	4.3	1.4	0.5	3.1	0.9	0.7	3.0	1.3	0.9	4.2
`RJ`	1.5	1.0	1.9	0.6	0.4	1.7	0.2	0.2	1.2	–	–	–
`DD`	7.2	7.0	5.1	6.7	4.0	7.2	1.7	0.9	2.3	2.7	1.5	5.4
`W`	12.1	12.6	12.0	14.5	16.8	12.7	20.5	21.3	18.3	20.2	20.8	19.4
`RM`	4.8	4.2	3.8	2.1	1.4	2.7	1.1	0.7	1.5	1.6	1.0	2.1
`NQ`	18.0	18.4	17.4	16.4	18	15.4	18.4	19.2	16.2	17.9	18.6	15.3
`QW`	2.8	2.7	5.4	1.8	1.0	3.7	1.0	0.6	2.1	1.2	0.8	2.2
`CT`	4.9	4.8	3.7	7.2	4.2	7.9	1.7	1.2	2.7	3.3	1.7	5.0
`NPR`	1.0	1.0	2.2	0.5	0.3	1.2	0.2	0.2	0.4	–	–	–
`NNQ`	15.0	15.4	9.5	13.9	16.5	10.4	13.4	15.2	10.9	13.8	15.0	10.2
`NQW`	0.6	0.5	1.5	0.3	0.2	0.8	0.1	0.1	0.4	–	–	–
`AQW`	1.3	1.2	1.9	0.8	0.4	1.5	0.5	0.3	0.8	–	–	–
`[1/PR]`	12.2	12.4	11.7	1.3	0.9	2.7	4.1	2.7	5.6	2.6	2.0	4.9
`[W/PR]`	–	–	–	1.6	0.8	2.4	4.9	4.0	5.9	4.5	4.2	5.0
`[NNQ⋆PR]`	–	–	–	2.0	1.3	1.9	0.4	0.3	0.8	–	–	–
`[W/NNQ]`	–	–	–	2.4	1.9	2.4	4.2	3.7	3.1	3.1	2.3	3.3
`[DD-CT]`	–	–	–	4.8	2.9	4.7	0.8	0.5	1.0	1.6	0.8	2.5
`[W/[PR⋆NNQ]]`	–	–	–	7.1	11.9	2.5	7.6	10.4	4.6	8.7	11.4	4.0

through GP.

Figure 6.7 presents pairs plots of the objectives TWT, NumDistinct and NumNodes for Stage 4 results. All of the DRs evolved with MOGP have fewer than 100 nodes, and most have fewer than 60 nodes. There are many SOGP evolved DRs with between 60 and 130 nodes.

Figure 6.8 plots f_t vs f_{xt} for Stage 4. We can see that the DRs which attain the lowest f_t and f_{xt} are evolved by SOGP$_4$. However of DRs with $0.0 < f_t, f_{xt} < 0.005$ the two evolved by MOGP$_{4,N}$ methods have mean size considerably smaller than of those of SOGP$_4$. We also note that there are fewer DRs attaining this level of performance compared to Stage 3. However these results in terms of f_t and f_{xt} are still better than the initial results prior to feature manipulation in Stage 1.

The average Pareto front size, after removing duplicate DRs, of GP runs with MOGP$_{4,N}$ is 16.8±6.6; this is the largest average front size of all methods, and is nearly double the mean front size in MOGP$_{1,N}$ runs. The average Pareto front size with MOGP$_{4,S}$ is 15.3±6.0, which nearly doubles the mean Pareto front size of MOGP$_{1,S}$ runs.

6.7 Further Analysis

This section presents a comparison of results through the various stages of feature manipulation. We also obtain the aggregate Pareto front of non-dominated DRs from *all* evolved DRs for each method, and examine and compare some of the best performing DRs. Further analysis also includes an examination of the trade-offs between pairs of objectives and exploration of useful subsets of features.

6.7.1 Comparison of Results by Stages

In this subsection we are interested in exploring whether the effectiveness of the GP search has been improved by the inclusion of higher-level functions and terminals, and the reduction of the terminal set.

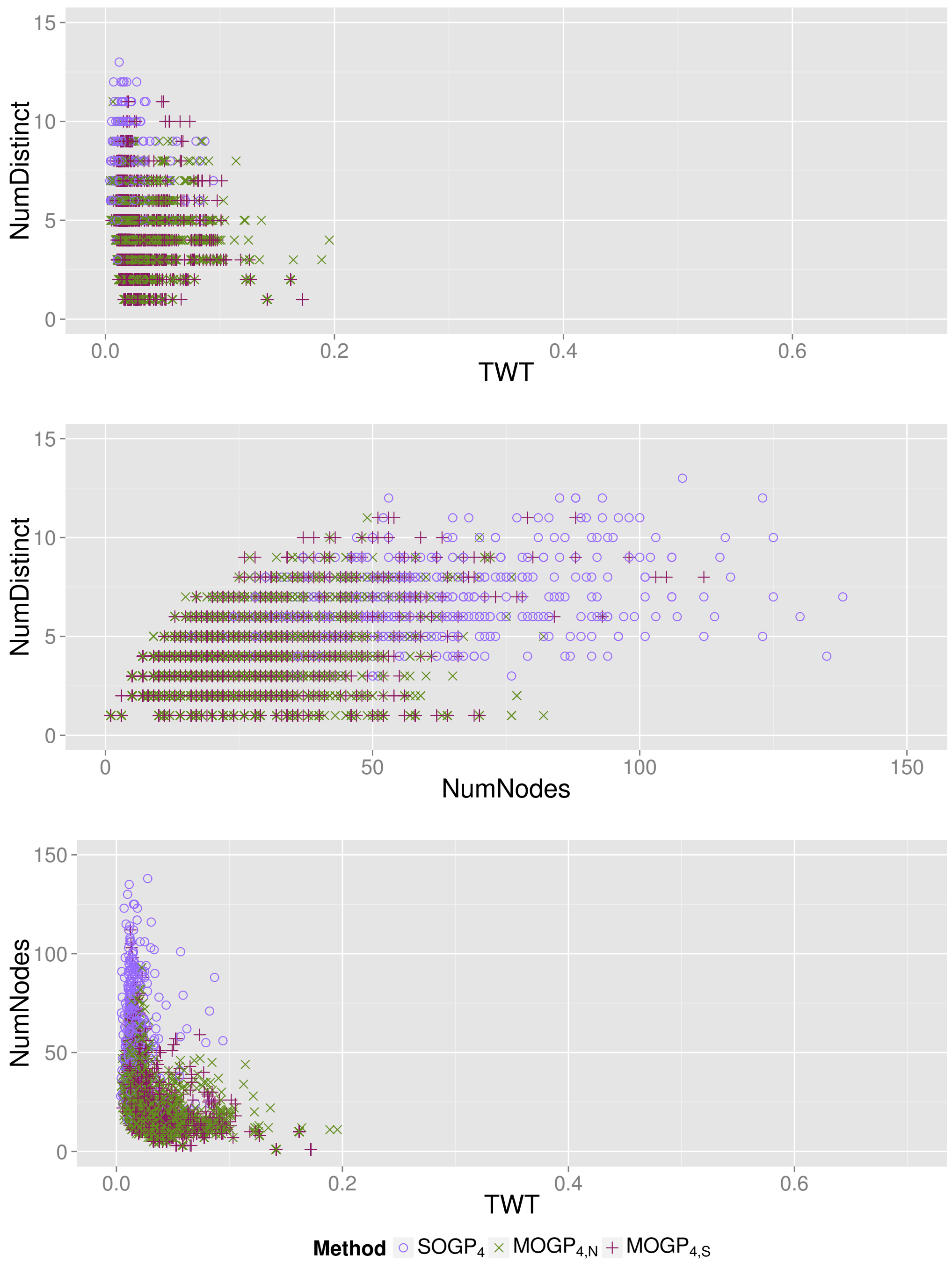


Figure 6.7: Paired plot of Stage 4 results. This plot contains the Pareto fronts of 500 runs for all methods, 9610 DRs (3401 distinct DRs) from MOGP$_{4,N}$, 7894 DRs (2343 distinct DRs) from MOGP$_{4,S}$ and 500 DRs from SOGP$_4$.

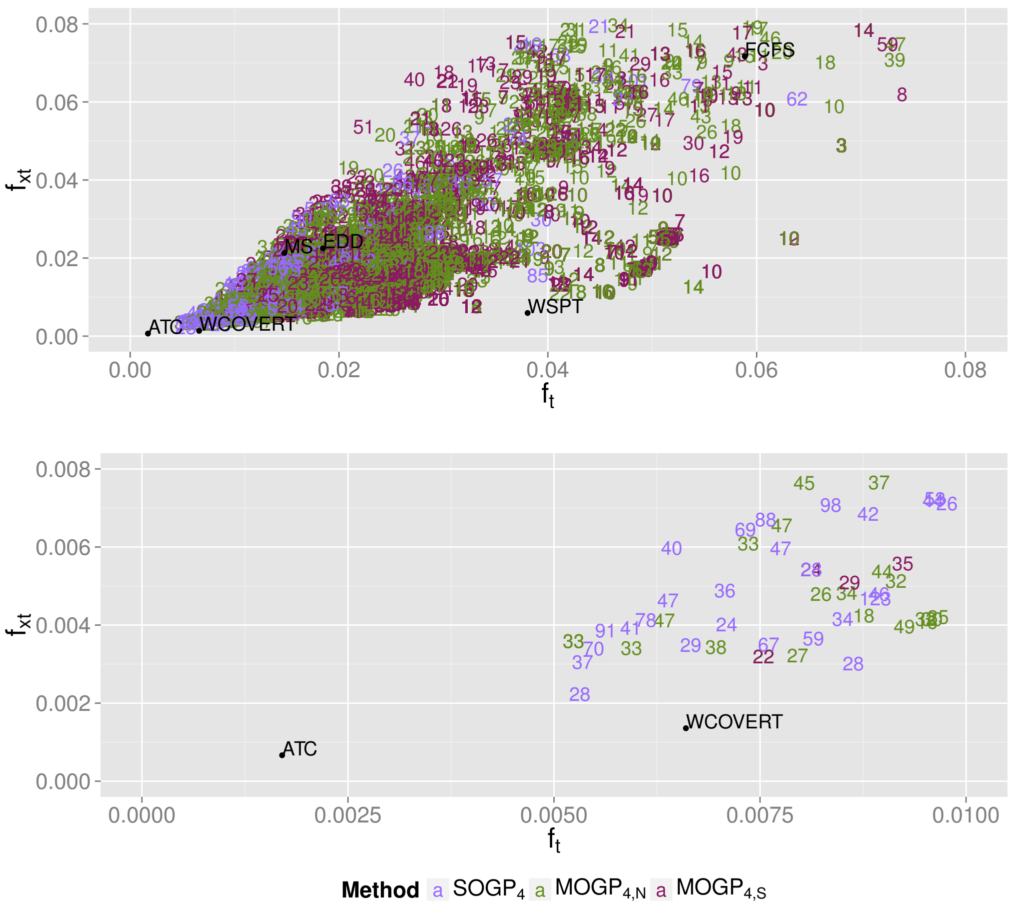


Figure 6.8: Plot of normalised TWT over test instances vs extreme test instances for Stage 4. The bottom plot zooms in on the area of best test and extreme test instances.

Figures 6.9(a)–(d) compare the results of MOGP with NSGA-II across all four stages. Figure 6.9(d), which plots f_t vs f_{xt}, highlights the difference in TWT performance across the four methods. As we want to minimise TWT, this shows the best performing DRs. It is clear to see that there are far more DRs from MOGP$_{2,N}$, MOGP$_{3,N}$ and MOGP$_{4,N}$ than MOGP$_{1,N}$ that attain this level of TWT performance. Further, MOGP$_{2,N}$ and MOGP$_{3,N}$ attain even lower (better) TWT performance than MOGP$_{4,N}$, observed by

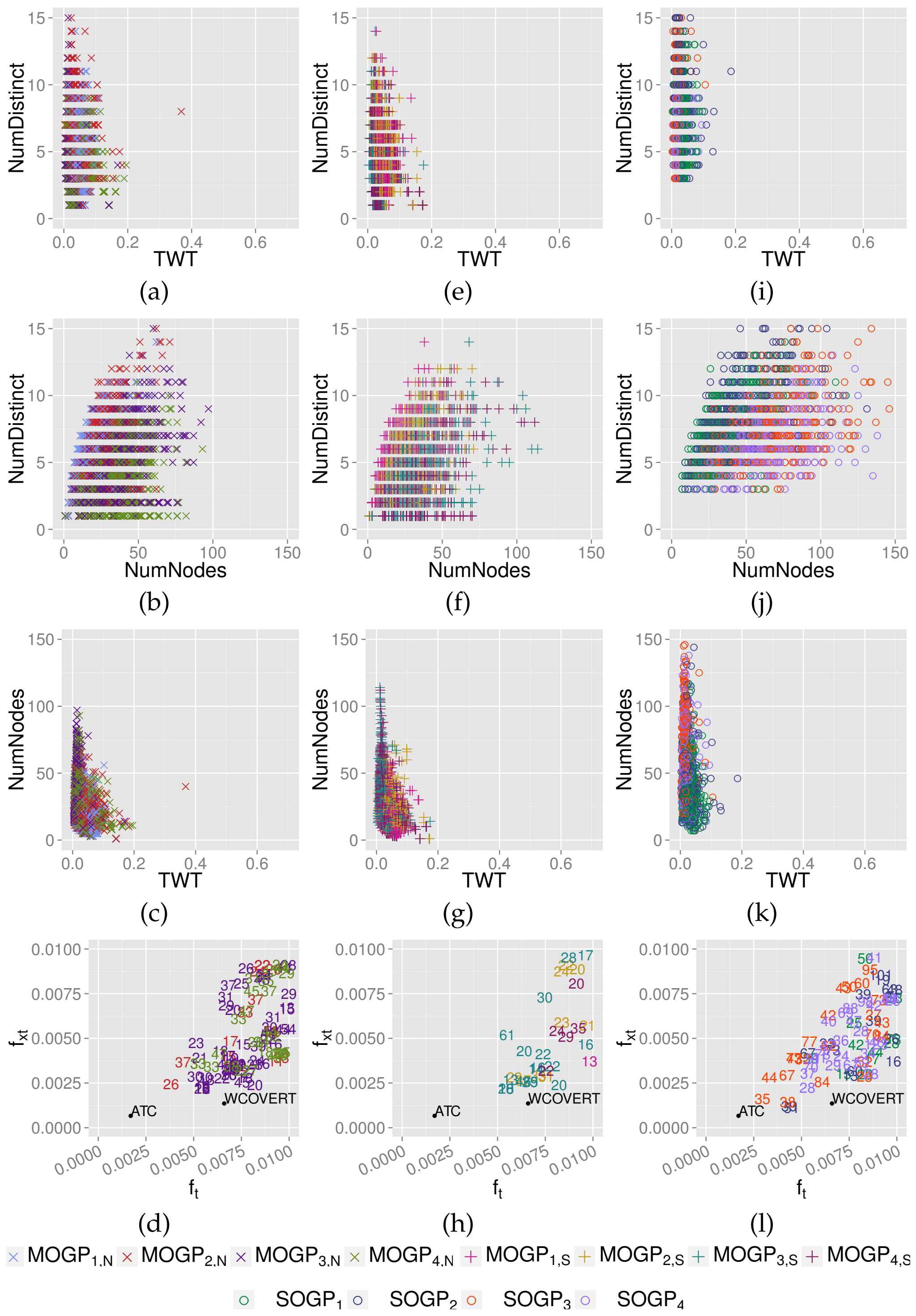


Figure 6.9: Paired plots of results and plots of normalised TWT over test instances vs normalised TWT over extreme test instances using the NumNodes as the plotting symbol for MOGP with NSGA-II (a)–(d), MOGP with SPEA2 (e)–(h) and SOGP (i)–(l).

the large number which attain $0 < f_t, f_{xt} < 0.004$, which is not attained by any DRs from MOGP$_{1,N}$. This shows the benefits of including the higher-level terminals. Also there are a greater number of MOGP$_{3,N}$ DRs (than MOGP$_{2,N}$) in the region shown, which shows the benefit of including the specialised conditionals. There is a greater number of MOGP$_{4,N}$ DRs than MOGP$_{2,N}$ but less than those from MOGP$_{3,N}$ in this region.

Figures 6.9(e)–(h) and (i)–(l) compare the results of MOGP with SPEA2 and SOGP across all four stages respectively. We note the same trends with both as for MOGP with NSGA-II, with better performing DRs evolved by MOGP$_{2,S}$, MOGP$_{3,S}$ and MOGP$_{4,S}$, and the best DRs being evolved by MOGP$_{2,S}$ and MOGP$_{3,S}$.

Comparing Figures 6.9(a)–(d) with (e)–(h) we see that the best NSGA-II DRs are better than the best SPEA2 DRs. Although there are more DRs attaining $0 < f_t, f_{xt} < 0.004$ using SOGP, the mean size of these DRs is over twice that of the DRs from the MOGP methods. This result highlights the *combined* worth of feature construction and multiobjective optimisation on this problem. Looking at the TWT performance of the benchmark SOGP methods throughout the four stages shows more clearly the impact of the addition and removal of additional terminals and functions.

The mean TWT is smallest for SOGP$_3$ followed by SOGP$_4$, SOGP$_2$ and SOGP$_1$. The difference of means between SOGP$_3$ and both SOGP$_1$ and SOGP$_2$ is statistically significant at the 5% significance level. This suggests that the inclusion of higher-level constructed terminals and specialised conditionals improves the effectiveness of DRs to dispatch jobs. Further, the reduction of the terminal set at Stage 4 shows that some terminals, although used rarely, can still be useful. However, the results of Stage 4 are still better than Stages 1 and 2, showing that more careful selection of terminals which contain better domain knowledge can be beneficial. We observe similar trends to these noted on the SOGP methods if we look only at the DR with the lowest TWT from each Pareto front from MOGP methods.

Table 6.5: Table of mean ± standard deviation of evolution times in minutes for each method.

	MOGP$_N$	MOGP$_S$	SOGP
Stage 1	186.9±101.7	232.6±177.4	74.2±30.5
Stage 2	157.6±33.8	192.3±68.3	79.6±32.7
Stage 3	173.7±66.9	198.3±91.0	79.9±34.5
Stage 4	178.4±81.6	206.0±86.5	81.2±46.1

Other trends are noticed for MOGP with both NSGA-II and SPEA2. The TWT performance by NumDistinct improves, as expected, because high-level terminals contain more domain knowledge. With DR size of 20 to 40 nodes, there is a noticeable improvement in mean TWT from Stage 3 and Stage 4.

Table 6.5 shows that the mean evolution time was shortest at Stage 2 for both MOGP methods, followed by Stages 3, 4 and then 1 for MOGP methods and was shortest at Stage 1 followed by Stages 2, 3 and 4 for SOGP methods. Standard deviations increase following the same trends as the means.

6.7.2 Extremes

Here we examine the behaviour of DRs at the extreme lower end of the three objectives: dispatching rules with the only one recurring terminal, dispatching rules with a small number of nodes, and dispatching rules with the best (lowest) attained TWT.

Dispatching rules with only one recurring terminal

Approximately one quarter of all DRs appearing in the non-dominated front (or are a best-of-run solution of SOGP methods) have only one distinct terminal. However there are only just over 100 different rules amongst these; the rest are duplicates as the same DR appears in multiple

Pareto fronts of many methods, or as DRs which simplify to the same DR. Stage 1 MOGP methods only give two such DRs: [1/PR], i.e., the shortest processing time DR; and W. These DRs were evolved by both MOGP$_{1,N}$ and MOGP$_{1,S}$. At Stage 2 almost all of the DRs with one distinct terminal contain [W/[PR*NNQ]], with a few having a single node of one of the other terminals. There are many large DRs which contain only [W/[PR*NNQ]], ranging in size up to 41 nodes. The DRs at Stage 2 attained better TWT performance than those of Stage 1. At Stage 3, the size of DRs with only one distinct terminal increased, ranging to over 60 nodes. These rules generally contained both [W/[PR*NNQ]] and either ifPS or ifOD. The most frequently used conditional in the better performing of these DRs is ifPS, although several DRs use both. These DRs obviously contain more domain knowledge than those of Stage 1 and Stage 2, and the improvement in TWT performance attained is notable. This shows that effective DRs can be constructed from a limited terminal set.

Calculating whether a job has positive slack uses the job's remaining processing time and due date, and the current time. This poses the question of *what is a realistic number of terminals for a dispatching rule?* We expect that with a TWT objective to be minimised, at least due date and weight of the job are needed for a rule to be effective. The best performing of all DRs with only one recurring terminal is from MOGP$_{3,N}$, and attains $f_{TWT} = 0.1454$, which is far from the best obtained TWT performance. This rule simplifies to

$$\texttt{ifPS[W/PR]}(8 * \texttt{[W/PR]}),$$

which contains CT, DD, RT in the ifPS statement, so this is a DR which contains far more domain knowledge than a single terminal (e.g. W), however this DR simplifies to be very easily understood.

Dispatching rules with a small number of nodes

The DRs with the smallest number of nodes are the DRs which have one node, and hence one distinct terminal, so are the same DRs discussed

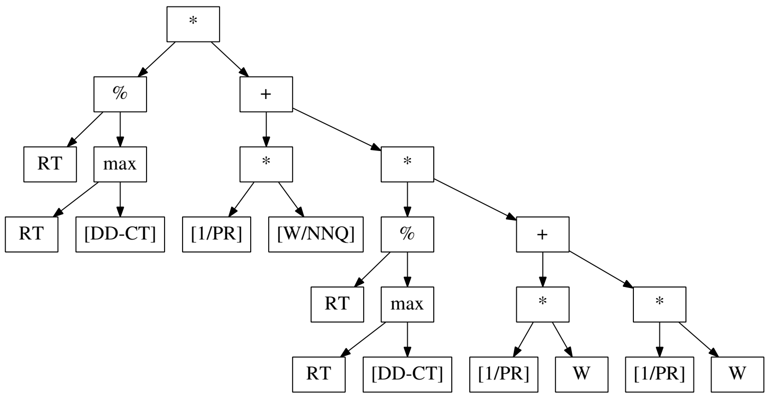
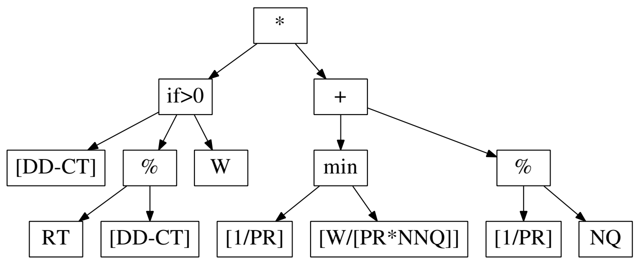


Figure 6.10: DR with the lowest mean normalised TWT on test and on extreme test scenarios.

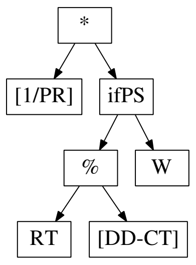
above. There are a number of DRs which have only three nodes; however the performance of these (particularly for Stage 1 methods) is not good, although they do improve on the TWT values achieved by single node DRs. While these very small DRs are efficient in terms of computational time and are easy to interpret, the trade-off at these *very* small sizes is perhaps not worth it. Mean performance for NumNodes increases at around 10 nodes.

Dispatching rules with best TWT performance

Figure 6.10 shows the DR which attains the lowest mean normalised TWT values across all methods and stages. This DR was evolved by SOGP$_3$, has 35 nodes, and uses 5 distinct terminals. In comparison, Figures 6.11(a) and 6.11(b) show two DRs which also have good TWT performance, the first evolved by MOGP$_{2,N}$ and the second by both MOGP$_{3,N}$ and MOGP$_{3,S}$. These DRs are obviously smaller in size than the DR of Figure 6.10. In particular, the DR of Figure 6.11(b) has only 16 nodes, and four distinct terminals from the terminal set. This rule is very easy to interpret, with the priority assigned depending on whether the job has positive slack or not. Jobs with non-positive slack are assigned WSPT priorities, and jobs with



(a) DR with low mean normalised TWT and smaller size, evolved by MOGP$_{2,N}$.



(b) DR with low mean normalised TWT and even smaller size, evolved by MOGP$_{3,N}$ and MOGP$_{3,S}$.

Figure 6.11: Two of the best DRs evolved by MOGP methods.

positive slack have priority $\frac{RT}{PR*[DD-CT]}$. This is the ratio of time remaining to be processed, to the remaining time until overdue, multiplied by the SPT priority.

The top 25 DRs have mean normalised TWT less than 0.043. Of these rules, none were evolved at Stage 1. Those that were evolved under SOGP have size between 32 and 77, whereas those under MOGP have size 15–37 with NSGA-II and 13–29 with SPEA2. This is a clear improvement in size of DR, for not too great a difference in effectiveness. Interpreting a DR with 77 nodes is far more difficult than one with 20 nodes.

6.7.3 Aggregate Pareto Fronts

Figure 6.12 presents three rotated views of the aggregate Pareto fronts of non-dominated DRs across 500 runs for each method. The number of

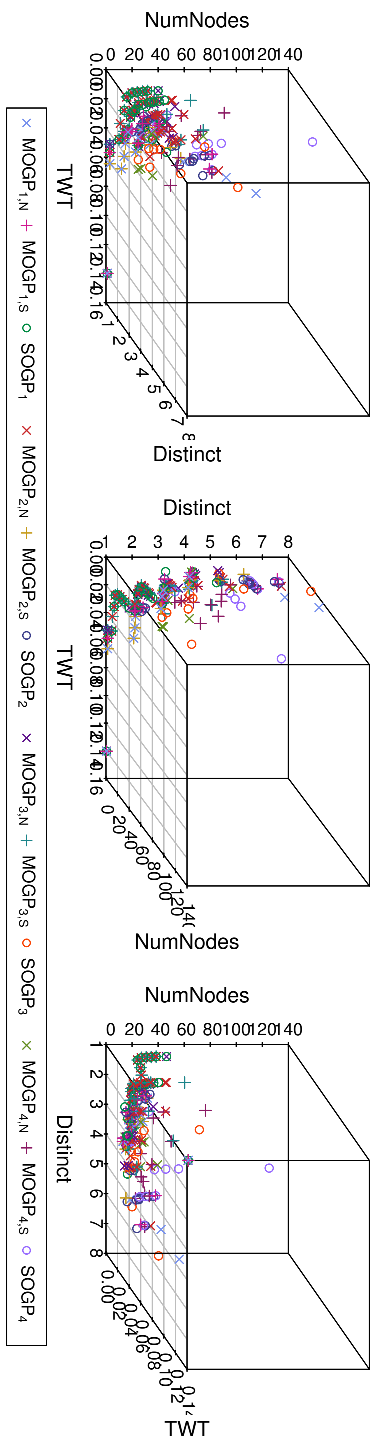


Figure 6.12: Rotated 3D plot of aggregate non-dominated Pareto fronts from each method.

points (the number of DRs) which are represented for each method are: $\text{MOGP}_{1,N}$ has 14 (1466), $\text{MOGP}_{1,S}$ has 14 (1470), SOGP_1 has 9 (9), $\text{MOGP}_{2,N}$ has 21 (1580), $\text{MOGP}_{2,S}$ has 22 (1884), SOGP_2 has 8 (8), $\text{MOGP}_{3,N}$ has 33 (1473), $\text{MOGP}_{3,S}$ has 37 (984), SOGP_3 has 13 (15), $\text{MOGP}_{4,N}$ has 46 (1601), $\text{MOGP}_{4,S}$ has 37 (1515), and SOGP_4 has 14 (14). There is a clear trend of increasing aggregate Pareto front size moving from Stage 1 through to Stage 4.

Figure 6.12 shows the clear trend amongst the aggregate Pareto fronts of improving TWT as NumNodes increases and NumDistinct increases. The fronts of methods from Stage 2 to Stage 4 are noticeably better than those of Stage 1, and with larger front size trends are clearer. From Figure 6.12 it can be observed that the objective TWT conflicts with both NumNodes and NumDistinct. When tracing along the Pareto front to find DRs that attain low TWT, the value of NumNodes and NumDistinct increases.

6.7.4 Trade-Offs between Pairs of Objectives

Here we will examine the trade-offs between each pair of the three objectives, to see what trends we can identify and insight we can gain.

TWT and Size of DR

Figure 6.13 shows a graph of the mean $\pm$ standard deviation of mean normalised TWT vs DR size for each method. The methods are offset from each other for visibility. At Stages 1 and 2, across both SOGP and the two MOGP methods, there is a clear trend of decreasing (i.e. improving) TWT as NumNodes increases from 1 to 15. Once 15 nodes is reached, the TWT plateaus. At Stages 3 and 4 a decreasing trend in TWT is again noted as NumNodes increases from 1 to 19; the mean TWT is lower for these stages in which the specialised conditional operators were introduced. As the introduction of these is the only difference between Stage 2 and Stage 3, we can attribute the difference to the inclusion of these functions. For each

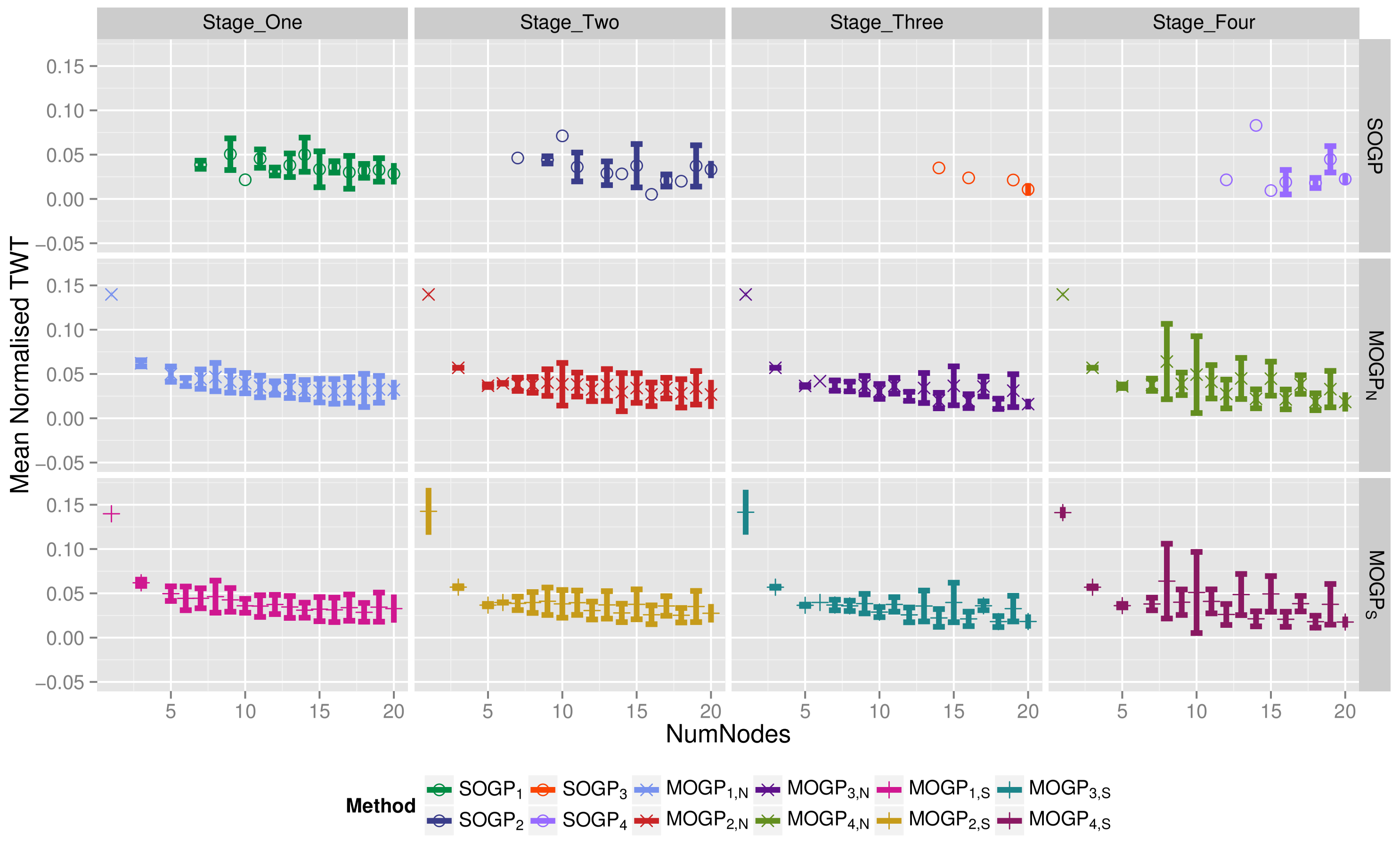


Figure 6.13: Graph of mean±standard deviation of mean normalised TWT vs size of DR. The graph is cut off at 20 nodes for space, and methods are offset from each other for visibility.

size of DR, similar trends can be noted amongst the methods. For an even size of DR, the mean normalised TWT decreases from Stage 1 through to Stage 4. For an odd number of nodes the performance is often more similar.

TWT and Number of Distinct Terminal Nodes

Figure 6.14 shows a graph of the mean $\pm$ standard deviation of mean normalised TWT vs NumDistinct for each method. Again the methods are offset for visibility. Note that 17 is the highest value of NumDistinct appearing in any DR evolved through SOGP or MOGP out of a possible 15, 20, 20 and 14 at Stages 1, 2, 3 and 4 respectively. This means that at Stages 2 and 3 no DR has been evolved which includes all of the available terminals, suggesting that there is possibly a redundancy amongst the terminal set in all stages. Note that a DR with an even number of nodes must contain an odd number of `if>0` statements, as these are the only functions which take three arguments.

Above we noted the plateau in TWT performance occurred at a DR size of 15 to 19 nodes, dependent on method. NumDistinct in DRs with size 19 ranges from 1 to 9, with a mean of 4.6. The graph in Figure 6.14 shows a plateau occurring at three distinct terminals for DRs from Stages 3 and 4. However, for those from Stage 1 and Stage 2, TWT performance continues to improve as NumDistinct increases to six.

At this point we consider *why using fewer distinct nodes might be advantageous*. One reason is that there might be less computational time required (perhaps) to obtain values from jobs and the shop and hence quicker evaluation of jobs in the machine queues at each dispatch decision point. Dispatching rules with fewer distinct terminals are also likely to have fewer "less-myopic" terminals which require calculating values (not just a lookup of a job or shop property).

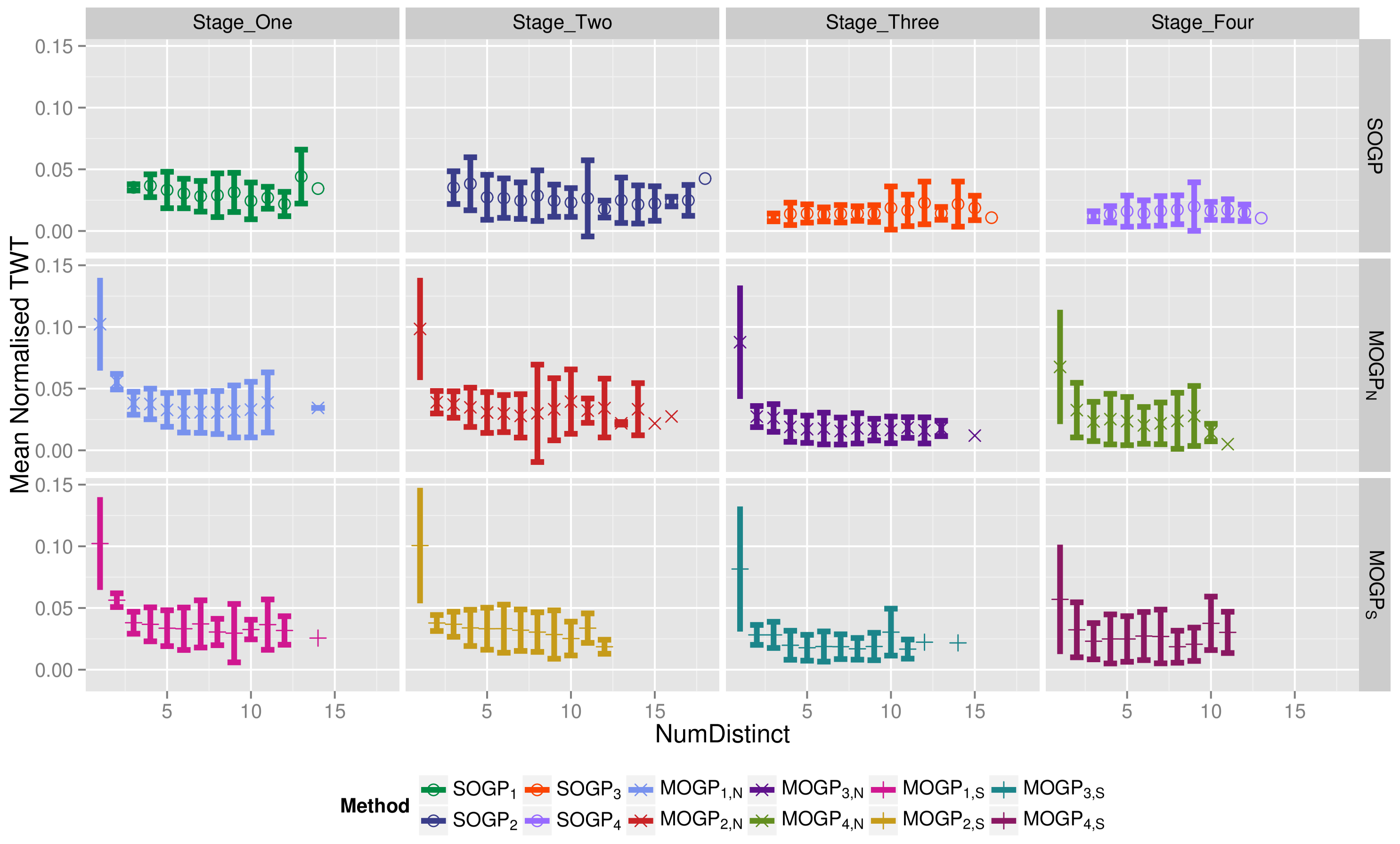


Figure 6.14: Graph of mean±standard deviation of mean normalised TWT vs number of distinct terminal nodes. The methods are offset from each other for visibility.

Size of DR and the Number of Distinct Terminal Nodes

There is clearly a link between the size of DR and the number of distinct terminal nodes. As NumNodes increases, NumDistinct is able to increase, as the relationship between NumDistinct and NumNodes is a floor function: NumDistinct$\leq \frac{1}{2}$NumNodes. Consider DRs with 16 nodes; of such DRs evolved by our methods, NumDistinct ranges between one and seven. First we look at several of the best performing DRs (in terms of TWT) with 16 nodes, e.g., the rule

$$\texttt{ifPS}\left(\frac{\texttt{RT}}{\texttt{PR}(\texttt{DD}-\texttt{CT})}\right)\left(\frac{\texttt{W}}{\texttt{PR}}\right)$$

was evolved in Stage 3 by both MOGP$_{3,S}$ and MOGP$_{3,N}$. This rule is one of the overall best performing DRs in terms of TWT performance. The next best 16 node rule,

$$\texttt{ifOD}\left(\frac{\texttt{W}}{\texttt{PR}}\right)\left(\frac{\texttt{RT}}{\texttt{PR}(\texttt{DD}-\texttt{CT})}\right)$$

is very similar, testing if a job is overdue rather than whether it has positive slack. This rule was evolved by MOGP$_{3,N}$. The third best 16 node rule is again, very similar. This time the condition has been evolved using `if>0` with first argument `[DD-CT]`, however the rule simplifies to become

$$\frac{1}{\texttt{NQ}}\left(\texttt{ifOD}\left(\frac{\texttt{W}}{\texttt{PR}}\right)\left(\frac{\texttt{RT}}{\texttt{PR}(\texttt{DD}-\texttt{CT})}\right)\right).$$

This rule was evolved by MOGP$_{2,N}$. In comparison, of the DRs with 16 nodes, the best performing DR with only one distinct terminal simplifies to

```
ifPS([1/PR])(2*[1/PR])
```

which attains normalised TWT eight times greater than the best attained by the rules above. The best performing DR with size 16 and seven distinct nodes is

```
(CT-DD)*(if>0 ([DD-CT]*[W/PR]) (PR/RT) W)
```

and outperforms the best DR with one distinct terminal. However, of 16 node DRs, the average TWT performance of DRs with seven distinct nodes (0.0405±0.0252) is worse than those with just a single distinct terminal (0.0318±0.0099). We note that all these rules are similar in their structure and terminals used, and further that each component of the `if` statements used is able to be easily understood.

6.7.5 Attribute Sets

An interesting aspect of analysis which expands on the terminal frequency analysis of Table 6.4 (see page 246) is to identify frequently appearing attribute subsets amongst the evolved rules, and see which attributes are the most useful as the subset size increases. It is clear that NumDistinct does not correspond with the true number of distinct attributes of the job shop. Therefore we have also evaluated the most frequently used subsets of distinct attributes. We will denote the most popular subsets as S_n where n is the number of distinct attributes appearing in the rule. The increasing sequence of sets is illustrated in Figure 6.15.

The most frequently used subset of two distinct attributes is

$$S_2 = \{\texttt{W}, \texttt{NNQ}\},$$

with over 5000 rules appearing in the non-dominated fronts with this pair. The next most popular subset of size two is {`W`, `PR`} with only 1030 rules. This is a considerable drop in frequency.

Unsurprisingly the most common subset of attributes of size three consists of the three attributes appearing in the most common subsets of size two:

$$S_3 = \{\texttt{W}, \texttt{PR}, \texttt{NNQ}\}.$$

There are nearly 15,000 evolved rules with this subset of attributes. All the higher-level constructed terminals (`[1/PR]`, `[W/PR]`, `[W/NNQ]`, `[NNQ*PR]` and `[W/[PR*NNQ]]`) are made from this subset of attributes. Given that

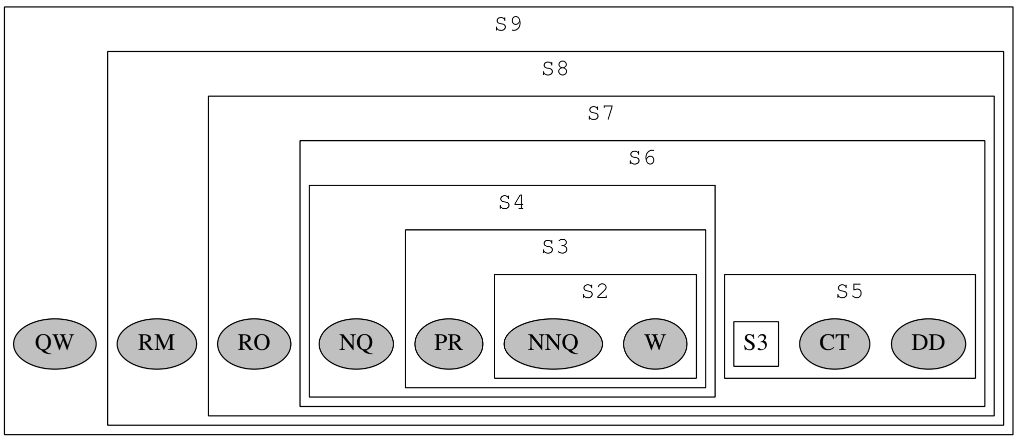


Figure 6.15: Most frequently occurring attribute subsets of size two to nine.

these attributes make up the higher-level constructed terminals, it follows that they are a useful subset and hence appear very frequently.

The attribute subset of size four builds upon S_3, also including NQ:

$$S_4 = \{\texttt{W}, \texttt{PR}, \texttt{NNQ}, \texttt{NQ}\}.$$

With 5806 rules using S_4, this is far less than the number of rules using S_3. The second most popular subset of size four is

$$S_{4B} = \{\texttt{W}, \texttt{PR}, \texttt{NNQ}, \texttt{RM}\}.$$

Increasing the subset size further,

$$S_5 = \{\texttt{W}, \texttt{PR}, \texttt{NNQ}, \texttt{CT}, \texttt{DD}\}.$$

It is interesting that this does not build directly on S_4, however it shows that CT and DD together add value to the attribute subset. As RM, the time the machine becomes available to dispatch the next job, is almost always the same as CT (except when the machine has been idle and multiple jobs arrive at the machine at the same time), S_5 can be considered to build on S_{4B}. There are 1086 rules using only S_5.

S_6, consists of all attributes that have appeared so far,

$$S_6 = \{\texttt{W}, \texttt{PR}, \texttt{NNQ}, \texttt{CT}, \texttt{DD}, \texttt{NQ}\}.$$

Figure 6.15 shows that S_7, S_8 and S_9 are made by adding `RO`, `RM` and `QW` consecutively.

6.8 Chapter Summary

One advantage of manually designed dispatching rules over those automatically designed by GP is their ease of interpretation. In this chapter we have continued to address this issue, building on the material of Chapter 5, through the use of a MOGP approach to investigating whether the inclusion of additional objectives of dispatching rule size and the number of distinct terminals can improve the interpretability of evolved dispatching rules. The original contributions of this chapter can be summarised as follows.

Firstly, the inclusion of NumNodes and NumDistinct can improve the interpretability of the evolved dispatching rules. The experimental results from the first stage of our investigation showed that SOGP and MOGP have a similar range of mean normalised TWT. However, for each SOGP DR, there is a MOGP evolved DR with similar TWT performance with smaller size and smaller subset of terminals used. When the best performing DRs were examined, those from MOGP methods were approximately half the size of those from SOGP.

The second contribution is the investigation of feature selection and feature construction using MOGP for dynamic JSS. Feature construction occurred at Stage 2. Fragment analysis of evolved DRs from Stage 1 identified four two-level fragments and one three-level fragment which appeared in a large number of DRs in both MOGP$_{1,S}$ and MOGP$_{1,N}$. These constructed higher-level terminals were added to the terminal set for Stage 2. Inclusion of these terminals, with greater domain knowledge, improved the mean TWT performance of SOGP and of MOGP in the region of the Pareto front with best TWT performance.

Three specialised conditional statements were introduced alongside the existing `if>0`. The inclusion of these terminals was also an improvement on the results of Stage 1, and in all cases lead to a greater number of DRs attaining a very high level of TWT performance.

The third contribution is at Stage 4, where the terminal set of Stage 3 was reduced by removing terminals which did not appear frequently. The results of Stage 4 are better than those of Stage 1, with many more DRs achieving low TWT performance, however fewer achieve this level of performance than in Stages 2 and 3. This is an interesting result, suggesting some terminals are more useful as part of larger constructions, as `RT` is in `ifPS`.

The average evolution time is shorter for MOGP methods using NSGA-II than those using SPEA2, and the best from NSGA-II is better than the best from SPEA2. In general the performances from the two algorithms are similar. Performance of evolved rules was competitive with WCOVERT and many rules outperformed FCFS, MS, EDD and WSPT. ATC, although outperformed on a number of test and extreme test instances, had the best overall performance. This suggests that using TWT values obtained by ATC as a comparison during evolution may be useful to try to improve performance.

This work is an incorporation of automatic generation of DRs via MOGP and feature manipulation, which shows how combining the domain knowledge rediscovered by GP to alter the input of GP has been able to improve the performance and learning quality.

This chapter has investigated how MOGP can be used to encourage the evolution of DRs which are more readily interpretable by human operators, by including the size of the DR and the number of distinct terminals included in the DR. For future studies we suggest including further objectives that cover generalisation ability and ability to cope with changing arrival processes throughout time.

Chapter 7

Conclusions

This thesis has focused on using genetic programming for the automatic discovery of new dispatching rules for job shop scheduling problems. The overall goal was to develop genetic programming based hyper-heuristics for the efficient evolution (automatic generation) of robust, reusable and effective scheduling heuristics for job shop scheduling environments, with greater interpretability. This goal was successfully achieved through studying the representations used, the incorporation of additional feedback on dispatching rule performance through other heuristic methods and the inclusion of additional objectives during the evolutionary process. The evolved rules were compared to existing approaches and dispatching rules from the literature.

The remainder of this chapter first reviews the objectives achieved, and the conclusions of the main contribution chapters. Then suggestions for future work to extend the work in this thesis are given.

7.1 Achieved Objectives

This thesis has fulfilled the following research objectives.

1. This thesis presented the first investigation into using GP to evolve optimal dispatching rules for the $J2||C_{max}$ problem (Chapter 3). It

was shown that Jackson's algorithm is able to be represented as a dispatching rule. A GP representation was developed and it was shown that GP was capable of discovering a dispatching rule which is equivalent to Jackson's algorithm and will always dispatch jobs in a way which attains the minimum makespan. In the dynamic two-machine job shop, a representation involving two dispatching rules simultaneously, one for each machine in the dynamic two-machine job shop, was compared to the standard approach of using one dispatching rule to make scheduling decisions at all machines in the shop. The results show that no method consistently outperformed another. However, using machine-specific rules resulted in shorter evolution and training times. It cannot be assumed that one rule is sufficient for all machines.

2. This thesis investigated approaches to discover "less-myopic" dispatching rules (Chapter 4). It was shown that the inclusion of additional terminals in the feature set led to the evolution of DRs which attain better mean performance on test instances with high utilisation levels, and decrease the mean queue length. A local search based means of providing additional feedback on the fitness of DRs over an extended decision horizon was proposed. The results show that the inclusion of local search leads to the evolution of DRs which make better decisions over the local time horizon and attain lower total weighted tardiness. The use of local search as a tie break mechanism was investigated, and results show that there is not sufficient benefit from this to justify the increase in computational time required.

3. This thesis investigated encouraging the interpretability of automatically discovered dispatching rules through the use of strongly typed GP (Chapter 5 and Chapter 6). The allowable interactions between functions and terminals were described using a sequence of grammars and implemented through strong typing. Although the

mean performance of DRs evolved with STGP was (as expected) not as good as the mean performance of DRs evolved without semantic constraint, there were very effective rules evolved under STGP. The use of a grammar decreased the mean program size. Results suggested that the size of the evolved DRs, and useful fragments which do not violate a given grammar, may be useful components of a measure of interpretability. Defining a widely acceptable measure of interpretability is still believed to be a very difficult task. However, a function based on DR size and fragments or interactions would be one step closer to truly quantifying interpretability.

4. This thesis has investigated how multiobjective GP can be used to evolve dispatching rules for the dynamic JSS problem which are effective in terms of scheduling objectives and are also more easily understood by human operators (Chapter 6). There is a trade-off between the size of the GP trees (compactness of heuristics) and the interpretability of heuristics was considered. The results showed that interpretability was improved by the inclusion of objectives relating to the size of the DR, as for each DR evolved without the additional objectives of minimising the number of nodes and distinct nodes in the evolved DR, there was a MOGP evolved DR with similar TWT performance with a smaller size and a smaller subset of terminals used. This smaller size makes interpretability much greater, for little-to-no decrease in TWT performance. In general, the performances from the NSGA-II and SPEA2 were similar. The average evolution time was quicker for MOGP methods using NSGA-II than those using SPEA2, and the best from NSGA-II are better than the best from SPEA2.

5. This thesis explored the use of knowledge implicitly discovered through GP to perform feature manipulation to improve dispatching rule performance (Chapter 6). Best-of-run evolved DRs were examined and five commonly occurring fragments of terminals and functions

were added to the terminal set as higher-level constructed terminals. The inclusion of these terminals improved the region of the non-dominated front with the best TWT performance. Three specialised conditional statements were also introduced, also causing an improvement in the best TWT attained. Finally, feature selection was performed by frequency analysis of used terminals. Results suggested that some attributes are more useful as part of higher-level terminals than as a base-level terminal. Results showed that the use of higher-level constructed terminals, which have clear definitions in the domain space, improves the interpretability of the evolved rules.

7.2 Main Conclusions

This thesis finds that GP is able to evolve dispatching rules which are less-myopic, more interpretable than rules previously evolved with GP and which perform competitively with existing methods. This section presents the main conclusions from the five research objectives in the four major contribution chapters (Chapter 3 to Chapter 6).

7.2.1 Optimal Dispatching Rules

Chapter 3 explored one of the simplest job shop scheduling environments: the static and dynamic two-machine job shops. The static two-machine job shop is known to have an optimal solution with the makespan objective function. There are few job shop environments with simple algorithms known to provide the optimal solution, so this provided a test ground of concepts if the algorithm was able to be represented as a dispatching rule. The dynamic two-machine job shop is one of the simplest of the dynamic job shop scheduling environments, and allowed for exploration of ideas of the trade-offs between simplification of the job shop model and the computational cost of discovering and evaluating DRs.

Representation of Jackson's Algorithm as a DR

This thesis finds that there exists a dispatching rule which will generate an ordering of jobs which always gives the same makespan as Jackson's algorithm. This dispatching rule can be represented very succinctly with a tree-depth of four, and only 12 nodes. This dispatching rule will not dispatch exactly the same sequence of jobs as the sequence that would be constructed using Jackson's algorithm. However it is able to work very quickly as production is progressing without requiring the schedule to be produced ahead of time.

Capability of GP to find Optimal DRs

This thesis finds that GP is sufficient in representation and search capability to discover DRs which are able to always dispatch waiting jobs to obtain the minimum makespan. In other words, the makespan obtained using these DRs will give the same value as the value of makespan obtained by scheduling jobs using Jackson's algorithm. It is the first time that optimal DRs for the static two-machine job shop with makespan objective function have been evolved by GP. The search capability of GP was improved by altering the terminal set to include an additional terminal. Additionally, it was shown that changing the problem instances every 10 generations vastly improved the search, and optimal DRs were found with greater ease.

Dynamic Two-Machine Job Shop

This thesis investigates the combined effects of using one scheduling rule versus machine specific scheduling rules and changing the problem instances throughout evolution in non-symmetric dynamic two-machine job shops. The results show that methods with changing problem instances do not consistently outperform methods with the same problem instances, nor do individuals with machine specific rules consistently outperform individuals consisting of only one scheduling rule. The two-machine dynamic job shop is a small job shop environment, and we cannot assume that one rule for all machines is sufficient for all job shop problems.

7.2.2 Less-Myopic Dispatching Rules

Chapter 4 explored two methods to encourage the evolution of DRs which are less-myopic, and take into account more than just the current state of the job shop at the current machine.

Less-myopic Attributes

The first instance where the inclusion of additional terminals which considered the future state of the shop was in Chapter 3, when the terminal `NPR`, the processing time of the next operation in the job, was added to the terminal set. The inclusion of the processing time of the next operation (`NPR`) as well as the total remaining processing time (`RT`) and the processing time of the current operation (`PR`), improved the GP search, and a greater number of DRs which passed the testing phase were found.

In Chapter 4, `NPR` and three further terminals, `AQW`, `NQW` and `NNQ`, were added to the terminal set. These are just a few of many possible terminals which consider properties from the shop that look further forward (or backward) in time, or wider in space across the shop. The inclusion of these less-myopic terminals was shown to lead to a statistically significant improvement in performance in terms of TWT over DRs evolved with the initial terminal set. This improvement in performance is due in part to how the less-myopic DRs dispatched jobs so that the mean and standard deviation of queue length was less, keeping the queues at machines more even in length and preventing queues from getting too long. The less-myopic DRs were also quicker to evaluate and dispatch jobs than their normal counterparts.

Local Search based Fitness Feedback

In Chapter 4, possible improvements to training DRs for the dynamic ten-machine job shop in GP through the use of local search were investigated. The expected contribution to the total weighted tardiness over an extended decision horizon was used to determine whether the order of queued jobs could be improved. A penalty was assigned to DRs which did not schedule queued jobs in the best order based on this expected contribution. Initial results showed that the inclusion of the additional feedback from local search led to the evolution of DRs which have better local performance and achieve some better TWT values. This was supported by our more in-depth investigation. Three local search operators were investigated: MoveFront, SwapFront and Transpose. SwapFront was shown to often lead to the best front of non-dominated DRs. We have analysed the mean queue length overall, the mean queue length when a positive penalty was assigned, and the mean queue length when no penalty (no improvement to the expected contribution to TWT was found) was assigned. This analysis showed that the mean queue length is longer when a positive penalty is assigned; however there is a large overlap in mean $\pm$ standard deviation of queue lengths.

Local Search based Tie-breaking

In Chapter 4, the use of local search to improve the tie-breaking behaviour of DRs was investigated. This thesis finds that most of the best-of-run evolved DRs have a tie-breaking penalty of 0. This supports the hypothesis that dispatching rules which are better at separating jobs by assigning distinct priorities are more effective. The spread of TWT and penalty results is similar across all four scenarios, which suggests that performing tie breaking with local search does not offer enough improvement in its current implementation to justify the additional computational cost which is incurred.

7.2.3 Interpretability of Evolved Dispatching Rules

Results in Chapter 4 suggest that the best-of-run dispatching rules evolved using standard GP are not easily interpretable. It is important that rules evolved using GP are able to be understood and trusted by the human operators and managers of the real-world scheduling environments. If they are not able to be trusted then they are unlikely to be used in practice.

There are many factors that affect the interpretability of evolved DRs. One factor identified in Chapter 4 was the way that features were combined with mathematical operators, e.g., it makes more sense to multiply or divide by the job weight than to add it to or subtract it from the current time.

This thesis has been the first work using GP to develop DRs which has considered interpretability of evolved DRs to be of as much importance as their performance.

Development of Grammars to Enforce Semantic Constraint

This thesis has developed new grammars to enforce semantic constraint on DRs within the evolutionary process through the use of strongly typed GP. In Chapter 5 terminals were initially partitioned into four basic types (count, weight, time duration, and clock time), and the allowable interactions of these four types were used to develop a new grammar. This grammar was extended incrementally to include three additional types, inverse time duration, squared time duration and inverse squared time duration.

Comparative Interpretability and Performance

The mean performance of DRs that were evolved with semantic constraints of STGP was not as good as the mean performance of DRs evolved without semantic constraint. Effective rules were still evolved under STGP from the best performing rules of each method, and the most effective

DRs were evolved under semantic constraint. The DRs evolved by STGP methods are shorter and easier to analyse and interpret. Interpretability of evolved rules was also improved by the introduction of three specialised conditional operators. These conditional operators, `ifPS`, `ifOD` and `ifLO`, tested whether the job has positive slack, whether the job was overdue, and whether the job was on its last operation. These are not only able to be understood more easily, but are using domain knowledge rediscovered by GP.

Insight into a Measure of Interpretability

Chapter 5 has provided greater insight into the interpretability of DRs, and what factors could potentially be used as part of a measure to quantify interpretability, allowing better comparison between evolved rules. Key components are the size of the evolved DRs, which could be measured in terms of depth, or in terms of the number of nodes, and the useful fragments, which could suggest that a count of such useful fragments which do not violate a given grammar could also be a potentially useful measure. Although defining a function based on DR size and fragments or interactions would be a step closer to quantifying interpretability, actually defining a widely acceptable measure of interpretability *remains* a very big challenge.

7.2.4 Multiobjective GP for Interpretable DRs

Job shop scheduling is inherently a multiobjective problem. DR performance is also a multiobjective problem; performance in terms of the scheduling objectives is important, as is the interpretability of DRs and practitioners being able to trust automatically generated DRs. The results of Chapter 5 suggested that including the size of DRs as an objective to be minimised alongside a more traditional measure of shop performance (e.g. TWT) was a sensible direction to investigate.

Interpretability Related Objectives

In Chapter 6, the evolution of dispatching rules with good scheduling performance and improved interpretability was investigated for the first time through the inclusion of two additional objectives. The fitness of a GP individual was evaluated in terms of three objectives: the scheduling performance in terms of TWT, the size of the individual in terms of the number of nodes, and the number of distinct terminals from the feature set that appeared in the individual. The inclusion of the size means that the smaller of two rules with identical performance is fitter, and it is likely to be more interpretable due to this smaller size. The inclusion of the number of distinct terminals as an objective was to encourage DRs to find smaller subsets of the terminal set which are sufficient to attain good performance and to enable feature manipulation. By comparison with single objective GP, the experimental results showed that SOGP and MOGP have a similar range of mean normalised TWT. However, for each DR evolved by SOGP, there was a DR evolved by MOGP which attained similar TWT performance with smaller size and a smaller subset of terminals used. The best performing DRs from MOGP methods were approximately half the size of the best performing DRs from SOGP.

MOGP Algorithms

The two MOGP algorithms used, NSGA-II [32] and SPEA2 [146], performed, in general, very similarly. The main observable differences were in the evolution time. The average evolution time is shorter for MOGP methods using NSGA-II than using SPEA2. Results also showed that the best evolved DRs from NSGA-II are better than the best evolved DRs from SPEA2.

7.2.5 Feature Manipulation in JSS using GP

The job shop scheduling problem has a very large number of potential terminals (and functions) that can be given as parameters to the GP system for the automatic discovery of dispatching rules. Therefore attribute/feature selection is an important part of developing an *effective* GP based approach for the automatic discover of dispatching rules. Feature manipulation was investigated in Chapter 5 and Chapter 6.

Feature Construction

In Chapter 5 three specialised conditional operators were introduced, based on analysis of the conditions that were found in best-of-run evolved DRs. Many of the conditions were overly long and complicated, or were always negative or always non-negative, hence the `if>0` statement was unnecessary. The remaining conditional statements lead us to introduce `ifPS`, `ifOD` and `ifLO`, which tested whether the job has positive slack, whether the job was overdue, and whether the job was on its last operation respectively. These specialised conditional operators were not only able to be understood more easily, but are using domain knowledge rediscovered by GP, and can be considered as feature construction of a condition statement.

Feature construction was also performed in Stage 2 of Chapter 6, where the best-of-run evolved dispatching rules were searched for commonly occurring fragments of terminals and functions. Four two-level fragments and one three-level fragment were used as constructed higher-level terminals that were added to the terminal set. Inclusion of these terminals with greater domain knowledge improved the mean TWT performance of all methods in the region of the Pareto front with the best TWT performance.

Feature Selection

The process by which GP automatically discovers knowledge through the terminals selected can be viewed as *feature selection*. At Stage 4 of Chapter 6, the terminal set of Stage 3 was reduced by removing terminals which did not appear frequently. This led to an improvement over the original terminal set used in Stage 1, with many more DRs achieving low TWT performance. This result suggests that some terminals are more useful as part of larger constructions, as `RT` is in `ifPS`.

Improving Interpretability

The introduction of higher-level terminals, each of which we were able to interpret, meant that these terminals did not reduce the interpretability of the evolved rules. The introduction of the three specialised terminals, and the removal of the `if>0` statement improved the interpretability of the evolved rules at the cost of slightly worse TWT performance.

7.3 Future Work

There has been extensive exploration of GP for the automatic generation of dispatching rules for JSS in this thesis. Although many of the limitations have been discussed in this thesis, there are further improvements that can be made to the state-of-the-art. This section highlights suggested directions for future work that are motivated by the investigations in this thesis.

Investigation of Less-Myopic Feature Sets. A direction for future work is to investigate whether the additional less-myopic terminals continue to offer improved performance as the scale of the job shop increases. It is also recommended to investigate additional terminals which capture further properties of the job shop's current and potential future states, and incorporate a look-ahead element. It could also be interesting to investigate which is more important, improving performance to be less-myopic in space or to be less-myopic in time. Further, do terminals which consider looking *backward* in time aid performance as well as terminals which look *forward* in time? A final suggestion in this direction would be to investigate estimators vs actual measures of properties. For many properties of the job shop there are different ways it could be measured, e.g., which is the best way to measure how busy a machine is; is it the number of waiting jobs, the utilisation, the work in the queue, or the average waiting time? For the terminals which take the average of observed values, how many values should be used for these? Five is an arbitrary number that we have used so far. Could GP be allowed to determine the number of values it takes the average of dynamically?

Local Search Extensions. When the local search based additional feedback over the extended decision horizon was used, the performance of evolved rules improved. We have investigated the queue lengths at machines, and compared queue lengths when a positive penalty was assigned and when no penalty was assigned. The next step in this investigation would be to consider *how* this information can be used to improve the performance of evolved DRs without increasing the computational time as much. With an improved use of computational time, further consideration to alternative local search operators, and combinations of local search operators, could be considered.

Comparison of two dispatching rules in depth. It would be interesting to develop a method to compare two dispatching rules, seeing how they assign priority values to queued jobs in the same decision situations, and examining when one rule outperforms the other. It would also be interesting to incorporate knowledge of the *rate of change* of priority values that are assigned to jobs. Some jobs may be re-evaluated multiple times at the same machine as other jobs are selected ahead of them; means of identifying jobs that are rapidly increasing in priority value relative to other queued jobs could improve the scheduling performance. It may be the case that selecting a relatively high priority job with greater rate of change would be a better decision than scheduling a higher priority job with lower rate of change. Through pairwise comparisons of dispatching rules the trade-offs in complexity and scheduling performance could be further explored.

Defining a Measure to Quantify Interpretability. Measuring the interpretability of dispatching rules is a difficult task, and requires manual examination of evolved dispatching rules. Insight gained through this thesis shows that the use of higher-level terminals, made of commonly used fragments, and encouraging smaller sized dispatching rules improved interpretability. A direction for future work is to investigate combining these, and other, elements into a measure of interpretability. Dispatching rules could be evolved without semantic constraint and given a value of how much they are acceptable under a grammar, and this value could then be combined with a measure of size. We still believe that defining a widely acceptable measure of interpretability remains a very difficult task, and a function based on DR size and fragments or interactions would only be one step closer to truly quantifying interpretability. Further investigations in this direction need to be made.

Multiobjective GP for Scheduling. Scheduling is a multiobjective problem. In this thesis we investigated including objectives relating to interpretability as well as the scheduling performance. For future studies, further objectives could be included. One such objective is the generalisation ability of the dispatching rule. One approach to this may be to train across job shops with varying numbers of machines and distributions as well as varying utilisation rates and number of operations, and attain a measure of performance difference relative to the state of the art. Another aspect of generalisation could be the ability to cope with changing arrival processes throughout time. This could include training on job shop scenarios in which the arrival process is a non-homogeneous Poisson process, i.e., the rate of arrival of jobs changes over time.

Feature Construction. Feature construction using GP could be improved, and incorporated automatically into the evolutionary process. Automatically defined functions (ADFs) allow for sub-functions to be called by a GP individual [78], however the ADFs are associated to the specific individual rather than belonging to the population as a whole. Modifying the structure so that sub-function trees are able to be evolved and accessed by the whole population could be one method of allowing dynamic feature construction.

Identifying Node Saliency. The use of frequency analysis on unsimplified rules is not the most reliable means of determining how salient a specific terminal is relative to other terminals. Terminals may appear in the DR but never be executed. Manual simplification is time consuming and impractical on a large scale. A possible alternative that could be pursued is to analyse the execution paths and use the proportion of time that the terminal is executed instead.

Uncertainty and Disruption. Job shops with disruption and uncertainty are big issues in scheduling, and could be considered in future studies. In situations where scheduling methods need to cope with uncertainty and disruption it will be even more important for the scheduling rules to be able to take into account the wider state of the shop and to be trusted. Human operators need to be able to trust that a rule will behave sensibly and not bring the shop to a standstill, but allow what can be processed to still be processed, and to recover quickly from disruption.

In summary, there are many future research directions that arise as a result of the work described in this thesis.

Bibliography

[1] AARTS, E., AND LENSTRA, J. *Local Search in Combinatorial Optimization*. Princeton University Press, 1997.

[2] AYTUG, H., LAWLEY, M., MCKAY, K., MOHAN, S., AND UZSOY, R. Executing production schedules in the face of uncertainties: A review and some future directions. *European Journal of Operational Research 161*, 1 (2005), 86–110.

[3] BÄCK, T., FOGEL, D., AND MICHALEWICZ, Z. *Evolutionary Computation 1: Basic Algorithms and Operators*. Taylor & Francis, 2000.

[4] BADER-EL-DEN, M. B., POLI, R., AND FATIMA, S. Evolving timetabling heuristics using a grammar-based genetic programming hyper-heuristic framework. *Memetic Computing 1*, 3 (2009), 205–219.

[5] BAKER, K. R. Sequencing rules and due-date assignments in a job shop. *Management Science 30*, 9 (1984), 1093–1104.

[6] BAKER, K. R., AND TRIETSCH, D. *Principles of Sequencing and Scheduling*. Wiley, 2009.

[7] BANZHAF, W., FRANCONE, F., KELLER, R., AND NORDIN, P. *Genetic Programming: An Introduction. On the Automatic Evolution of Computer Programs and its Applications*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1998.

[8] BEYER, H.-G., AND SCHWEFEL, H.-P. Evolution strategies – a comprehensive introduction. *Natural Computing 1*, 1 (2002), 3–52.

[9] BHOWAN, U., JOHNSTON, M., ZHANG, M., AND YAO, X. Evolving diverse ensembles using genetic programming for classification with unbalanced data. *IEEE Transactions on Evolutionary Computation 17*, 3 (2013), 368–386.

[10] BHOWAN, U., JOHNSTON, M., ZHANG, M., AND YAO, X. Reusing genetic programming for ensemble selection in classification of unbalanced data. *IEEE Transactions on Evolutionary Computation 18*, 6 (2014), 893–908.

[11] BISHOP, A., CIESIELSKI, V., AND TRIST, K. Feature construction using genetic programming for classification of images by aesthetic value. In *Evolutionary and Biologically Inspired Music, Sound, Art and Design: Third European Conference, EvoMUSART 2014, Revised Selected Papers* (2014), J. Romero, J. McDermott, and J. Correia, Eds., Springer, pp. 62–73.

[12] BLAZEWICZ, J., DOMSCHKE, W., AND PESCH, E. The job shop scheduling problem: Conventional and new solution techniques. *European Journal of Operational Research 93*, 1 (1996), 1–33.

[13] BLUM, C., AND ROLI, A. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *Association for Computing Machinery Computing Surveys 35*, 3 (Sept. 2003), 268–308.

[14] BRAMEIER, M. F., AND BANZHAF, W. *Linear Genetic Programming*. Springer, 2007.

[15] BRANKE, J., HILDEBRANDT, T., AND SCHOLZ-REITER, B. Hyper-heuristic evolution of dispatching rules: A comparison of rule representations. *IEEE Transactions on Evolutionary Computation 23*, 2 (2015), 249–277.

[16] BRANKE, J., NGUYEN, S., PICKARDT, C., AND ZHANG, M. Automated design of production scheduling heuristics: A review. *IEEE Transactions on Evolutionary Computation 20*, 1 (2016), 110–124.

[17] BRANKE, J., AND PICKARDT, C. Evolutionary search for difficult problem instances to support the design of job shop dispatching rules. *European Journal of Operational Research 212*, 1 (2011), 22–32.

[18] BURKE, E., HYDE, M., KENDALL, G., OCHOA, G., OZCAN, E., AND WOODWARD, J. R. A classification of hyper-heuristic approaches. In *Handbook of Meta-Heuristics*, M. Gendreau and J.-Y. Potvin, Eds. Kluwer, 2010, pp. 449–468.

[19] BURKE, E., KENDALL, G., SILVA, D. L., O'BRIEN, R., AND SOUBEIGA, E. An ant algorithm hyperheuristic for the project presentation scheduling problem. In *Proceedings of the 2005 IEEE Congress on Evolutionary Computation* (2005), vol. 3, IEEE, pp. 2263–2270.

[20] BURKE, E. K., GENDREAU, M., HYDE, M., KENDALL, G., OCHOA, G., OZCAN, E., AND QU, R. Hyper-heuristics: a survey of the state of the art. *Journal of the Operational Research Society 64*, 12 (2013), 1695–1724.

[21] BURKE, E. K., HYDE, M. R., KENDALL, G., OCHOA, G., OZCAN, E., AND WOODWARD, J. R. Exploring hyper-heuristic methodologies with genetic programming. In *Computational Intelligence*, C. Mumford and L. Jain, Eds., vol. 1 of *Intelligent Systems Reference Library*. Springer, 2009, pp. 177–201.

[22] BURKE, E. K., HYDE, M. R., KENDALL, G., AND WOODWARD, J. R. The scalability of evolved online bin packing heuristics. In *Proceedings of the IEEE Congress on Evolutionary Computation* (2007), pp. 2530–2537.

[23] CHAKHLEVITCH, K., AND COWLING, P. Hyperheuristics: Recent developments. In *Adaptive and Multilevel Metaheuristics*, C. Cotta, M. Sevaux, and K. Srensen, Eds., vol. 136 of *Studies in Computational Intelligence*. Springer, 2008, pp. 3–29.

[24] CHENG, R., GEN, M., AND TSUJIMURA, Y. A tutorial survey of job-shop scheduling problems using genetic algorithms, Part II: hybrid genetic search strategies. *Computers and Industrial Engineering 36*, 2 (1999), 343–364.

[25] CHIEN, B.-C., AND YANG, J.-H. Features selection based on rough membership and genetic programming. In *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics* (2006), vol. 5, pp. 4124–4129.

[26] CHOI, I.-C., AND CHOI, D.-S. A local search algorithm for jobshop scheduling problems with alternative operations and sequence-dependent setups. *Computers & Industrial Engineering 42*, 1 (2002), 43–58.

[27] CHRYSSOLOURIS, G., AND SUBRAMANIAM, V. Dynamic scheduling of manufacturing job shops using genetic algorithms. *Journal of Intelligent Manufacturing 12*, 3 (2001), 281–293.

[28] CONWAY, R., MAXWELL, W., AND MILLER, L. *Theory of Scheduling*. Addison-Wesley, 1967.

[29] COWLING, P. I., KENDALL, G., AND SOUBEIGA, E. A hyperheuristic approach to scheduling a sales summit. In *Selected Papers from the Third International Conference on Practice and Theory of Automated Timetabling III* (2001), Springer-Verlag, pp. 176–190.

[30] DE JONG, K. A. *Evolutionary computation - a unified approach.* MIT Press, 2006.

[31] DEB, K., MOHAN, M., AND MISHRA, S. Evaluating the ϵ-domination based multi-objective evolutionary algorithm for a quick computation of pareto-optimal solutions. *Evolutionary Computation 13*, 4 (2005), 501–525.

[32] DEB, K., PRATAP, A., AGARWAL, S., AND MEYARIVAN, T. A fast elitist multi-objective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation 6* (2000), 182–197.

[33] DIMOPOULOS, C., AND ZALZALA, A. M. S. Investigating the use of genetic programming for a classic one-machine scheduling problem. *Advances in Engineering Software 32*, 6 (2001), 489–498.

[34] DORIGO, M., AND STÜTZLE, T. *Ant Colony Optimization*. MIT Press, MA, USA, 2004.

[35] EGUCHI, T., OBA, F., AND TOYOOKA, S. A robust scheduling rule using a neural network in dynamically changing job-shop environments. *International Journal of Manufacturing Technology and Management 14*, 3-4 (2008), 266–288.

[36] FERREIRA, C. Gene expression programming: a new adaptive algorithm for solving problems. *Complex Systems 13*, 2 (2001), 87–129.

[37] FOGEL, D. B. Phenotypes, genotypes, and operators in evolutionary computation. In *Proceedings of the IEEE International Conference on Evolutionary Computation* (1995), vol. 1, IEEE, pp. 193–198.

[38] FOGEL, G. B. Evolutionary programming. In *Handbook of Natural Computing* (2012), G. Rozenberg, T. Bäck, and J. N. Kok, Eds., Springer, pp. 699–708.

[39] FOGEL, L. J. *Intelligence Through Simulated Evolution: Forty Years of Evolutionary Programming*. Wiley, 1999.

[40] GAO, J., SUN, L., AND GEN, M. A hybrid genetic and variable neighborhood descent algorithm for flexible job shop scheduling problems. *Computers & Operations Research 35*, 9 (2008), 2892–2907.

[41] GARRIDO, P., AND RIFF, M. C. DVRP: a hard dynamic combinatorial optimisation problem tackled by an evolutionary hyper-heuristic. *Journal of Heuristics 16*, 6 (2010), 795–834.

[42] GEIGER, C., UZSOY, R., AND AYTUG, H. Rapid modeling and discovery of priority dispatching rules: an autonomous learning approach. *Journal of Scheduling 9*, 1 (2006), 7–34.

[43] GERE, JR., W. Heuristics in job shop scheduling. *Management Science 13*, 3 (1966), 167–190.

[44] GIBBS, J., KENDALL, G., AND ÖZCAN, E. Scheduling English football fixtures over the holiday period using hyper-heuristics. In *Proceedings of Parallel Problem Solving from Nature* (2010), Springer, pp. 496–505.

[45] GLOVER, F., AND LAGUNA, M. *Tabu Search*. Kluwer, Norwell, MA, USA, 1997.

[46] GONZÁLEZ-RODRÍGUEZ, I., VELA, C. R., PUENTE, J., AND HERNÁNDEZ-ARAUZO, A. Improved local search for job shop scheduling with uncertain durations. In *Proceedings of the 19th International Conference on Automated Planning and Scheduling* (2009), pp. 154–161.

[47] HANSEN, P., MLADENOVIĆ, N., AND PÉREZ, J. A. M. Variable neighbourhood search: methods and applications. *4OR 6*, 4 (2008), 319–360.

[48] HART, E., ROSS, P., AND CORNE, D. Evolutionary scheduling: a review. *Genetic Programming and Evolvable Machines 6* (2005), 191–220.

[49] HELD, M., AND KARP, R. M. The traveling-salesman problem and minimum spanning trees: Part II. *Mathematical Programming 1* (1971), 6–25.

[50] HILDEBRANDT, T., HEGER, J., AND SCHOLZ-REITER, B. Towards improved dispatching rules for complex shop floor scenarios: a genetic programming approach. In *Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation* (2010), pp. 257–264.

[51] HOLLAND, J. H. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence.* University of Michigan Press, 1975.

[52] HOLTHAUS, O., AND RAJENDRAN, C. Efficient dispatching rules for scheduling in a job shop. *International Journal of Production Economics 48*, 1 (1997), 87–105.

[53] HOLTHAUS, O., AND RAJENDRAN, C. Efficient jobshop dispatching rules: further developments. *Production Planning & Control 11*, 2 (2000), 171–178.

[54] INGIMUNDARDOTTIR, H., AND RUNARSSON, T. P. Supervised learning linear priority dispatch rules for job-shop scheduling. In *Proceedings of Learning and Intelligent OptimizatioN (LION 5)* (2011), pp. 263–277.

[55] JACKSON, J. An extension of Johnson's result on job-lot scheduling. *Naval Research Logistics Quarterly 3(3)* (1956), 201–204.

[56] JAKOBOVIĆ, D., AND BUDIN, L. Dynamic scheduling with genetic programming. In *Proceedings of the 9th European Conference on Genetic Programming* (2006), pp. 73–84.

[57] JAKOBOVIC, D., JELENKOVIC, L., AND BUDIN, L. Genetic programming heuristics for multiple machine scheduling. In *Proceedings of*

the 10th European Conference on Genetic Programming (2007), pp. 321–330.

[58] JAKOBOVIĆ, D., AND MARASOVIĆ, K. Evolving priority scheduling heuristics with genetic programming. *Applied Soft Computing 12*, 9 (2012), 2781–2789.

[59] JAYAMOHAN, M., AND RAJENDRAN, C. New dispatching rules for shop scheduling: a step forward. *International Journal of Production Research 38*, 3 (2000), 563–586.

[60] JOHNSON, S. Optimal two- and three-stage production schedules with setup times included. *Naval Research Logistics Quarterly 1* (1954), 61–68.

[61] JOHNSTON, M., LIDDLE, T., AND ZHANG, M. A relaxed approach to simplification in genetic programming. In *Genetic Programming*, A. Esparcia-Alczar, A. Ekrt, S. Silva, S. Dignum, and A. Uyar, Eds., vol. 6021 of *Lecture Notes in Computer Science*. Springer, 2010, pp. 110–121.

[62] JONES, A., AND RABELO, L. Survey of job shop scheduling techniques. Tech. rep., National Institute of Standards and Technology, Gaithersberg, 1998.

[63] KEIJZER, M., AND BABOVIC, V. Dimensionally aware genetic programming. In *Proceedings of the Genetic and Evolutionary Computation Conference* (1999), vol. 2, pp. 1069–1076.

[64] KENNEDY, J., AND EBERHART, R. C. Particle swarm optimization. In *Proceedings of IEEE International Conference on Neural Networks* (1995), pp. 1942–1948.

[65] KENNEDY, J., AND EBERHART, R. C. *Swarm Intelligence*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2001.

[66] KINZETT, D., JOHNSTON, M., AND ZHANG, M. Numerical simplification for bloat control and analysis of building blocks in genetic programming. *Evolutionary Intelligence 2*, 4 (2009), 151–168.

[67] KIRA, K., AND RENDELL, L. The feature selection problem: traditional methods and a new algorithm. In *Proceedings of the Association for the Advancement of Artificial Intelligence* (1992), vol. 2, pp. 129–134.

[68] KIRKPATRICK, S., GELATT, C. D., AND VECCHI, M. P. Optimization by simulated annealing. *Science 220*, 4598 (1983), 671–680.

[69] KOHAVI, R., AND JOHN, G. H. Wrappers for feature subset selection. *Artificial Intelligence 97*, 1 (1997), 273–324.

[70] KOZA, J. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA, 1992.

[71] KRAWIEC, K. Genetic programming-based construction of features for machine learning and knowledge discovery tasks. *Genetic Programming and Evolvable Machines 3*, 4 (2002), 329–343.

[72] LAND, A. H., AND DOIG, A. G. An automatic method of solving discrete programming problems. *Econometrica 28*, 3 (1960), pp. 497–520.

[73] LANGDON, W., AND POLI, R. *Foundations of Genetic Programming*. Springer, 2002.

[74] LAW, A. M., AND KELTON, W. D. *Simulation Modeling and Analysis*. McGraw Hill, 2000.

[75] LAWLER, E., LENSTRA, J., RINNOOY KAN, A., AND SHMOYS, D. Sequencing and scheduling: algorithms and complexity. In *Logistics of Production and Inventory*, S. C. Graves, A. H. G. Rinnooy Kan, and P. H. Zipkin, Eds., vol. 4 of *Handbooks in Operations Research and Management Science*. Elsevier, 1993, pp. 445–522.

[76] LIDDLE, T., JOHNSTON, M., AND ZHANG, M. Multi-objective genetic programming for object detection. In *IEEE Congress on Evolutionary Computation* (2010), IEEE, pp. 1–8.

[77] LIU, H., AND MOTODA, H. *Feature Extraction, Construction and Selection: A Data Mining Perspective*. Kluwer, Norwell, MA, USA, 1998.

[78] LUKE, S. *Essentials of Metaheuristics*, second ed. Lulu, 2013. Available for free at http://cs.gmu.edu/∼sean/book/metaheuristics/.

[79] MCCARTHY, J. History of LISP. In *History of programming languages I* (1978), ACM, pp. 173–185.

[80] MCKAY, R. I., HOAI, N. X., WHIGHAM, P. A., SHAN, Y., AND O'NEILL, M. Grammar-based genetic programming: a survey. *Genetic Programming and Evolvable Machines 11*, 3–4 (2010), 365–396.

[81] MICHALEWICZ, Z., AND FOGEL, D. *How to Solve It: Modern Heuristics*. Springer, 2004.

[82] MILLER, J. F., AND THOMSON, P. Cartesian genetic programming. In *Genetic Programming*. Springer, 2000, pp. 121–132.

[83] MIYASHITA, K. Job-shop scheduling with GP. In *Proceedings of the Genetic and Evolutionary Computation Conference* (2000), pp. 505–512.

[84] MKAOUER, M. W., KESSENTINI, M., BECHIKH, S., DEB, K., AND Ó CINNÉIDE, M. High dimensional search-based software engineering: finding tradeoffs among 15 objectives for automating software refactoring using NSGA-III. In *Proceedings of the 2014 Conference on Genetic and Evolutionary Computation* (2014), ACM, pp. 1263–1270.

[85] MONTANA, D. J. Strongly typed genetic programming. *Evolutionary Computation 3*, 2 (1995), 199–230.

[86] MUNI, D. P., PAL, N. R., AND DAS, J. Genetic programming for simultaneous feature selection and classifier design. *IEEE Transactions on Systems, Man, and Cybernetics, Part B 36*, 1 (2006), 106–117.

[87] NAWAZ, M., ENSCORE, E. E., AND HAM, I. A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *Omega 11*, 1 (1983), 91–95.

[88] NELSON, B. *Foundations and Methods of Stochastic Simulation: A First Course*. Springer Science & Business Media, 2013.

[89] NESHATIAN, K., AND ZHANG, M. Genetic programming for feature subset ranking in binary classification problems. In *Proceedings of the 12th European Conference on Genetic Programming* (2009), Springer, pp. 121–132.

[90] NESHATIAN, K., AND ZHANG, M. Pareto front feature selection: Using genetic programming to explore feature space. In *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation* (2009), ACM, pp. 1027–1034.

[91] NESHATIAN, K., ZHANG, M., AND ANDREAE, P. Genetic programming for feature ranking in classification problems. In *Simulated Evolution and Learning*, vol. 5361 of *Lecture Notes in Computer Science*. Springer, 2008, pp. 544–554.

[92] NESHATIAN, K., ZHANG, M., AND ANDREAE, P. A filter approach to multiple feature construction for symbolic learning classifiers using genetic programming. *IEEE Transactions on Evolutionary Computation 16*, 5 (2012), 645–661.

[93] NESHATIAN, K., ZHANG, M., AND JOHNSTON, M. Feature construction and dimension reduction using genetic programming. In *Proceedings of the 20th Australian Joint Conference on Artificial Intelligence* (2007), Springer, pp. 160–170.

[94] NGUYEN, S., ZHANG, M., JOHNSTON, M., AND TAN, K. C. A co-evolution genetic programming method to evolve scheduling policies for dynamic multi-objective job shop scheduling problems. In *Proceedings of the IEEE Congress on Evolutionary Computation* (2012), pp. 3332–3339.

[95] NGUYEN, S., ZHANG, M., JOHNSTON, M., AND TAN, K. C. A computational study of representations in genetic programming to evolve dispatching rules for the job shop scheduling problem. *IEEE Transactions on Evolutionary Computation 17*, 5 (2013), 621–639.

[96] NGUYEN, S., ZHANG, M., JOHNSTON, M., AND TAN, K. C. Dynamic multi-objective job shop scheduling: A genetic programming approach. In *Automated Scheduling and Planning*, A. S. Uyar, E. Ozcan, and N. Urquhart, Eds., vol. 505 of *Studies in Computational Intelligence*. Springer, 2013, pp. 251–282.

[97] NGUYEN, S., ZHANG, M., JOHNSTON, M., AND TAN, K. C. Hybrid evolutionary computation methods for quay crane scheduling problems. *Computers & Operations Research 40*, 8 (2013), 2083–2093.

[98] NGUYEN, S., ZHANG, M., JOHNSTON, M., AND TAN, K. C. Learning iterative dispatching rules for job shop scheduling with genetic programming. *The International Journal of Advanced Manufacturing Technology 67*, 1–4 (2013), 85–100.

[99] NGUYEN, S., ZHANG, M., JOHNSTON, M., AND TAN, K. C. Learning reusable initial solutions for multi-objective order acceptance and scheduling problems with genetic programming. In *Proceedings of the 16th European Conference on Genetic Programming* (2013), pp. 157–168.

[100] NGUYEN, S., ZHANG, M., JOHNSTON, M., AND TAN, K. C. Automatic design of scheduling policies for dynamic multi-objective job

shop scheduling via cooperative coevolution genetic programming. *IEEE Transactions on Evolutionary Computation 18*, 2 (2014), 193–208.

[101] NIE, L., GAO, L., LI, P., AND LI, X. A GEP-based reactive scheduling policies constructing approach for dynamic flexible job shop scheduling problem with job release dates. *Journal of Intelligent Manufacturing 24*, 4 (2013), 763–774.

[102] NIE, L., GAO, L., LI, P., AND ZHANG, L. Application of gene expression programming on dynamic job shop scheduling problem. In *Proceedings of the 15th International Conference on Computer Supported Cooperative Work in Design* (2011), pp. 291–295.

[103] NIE, L., SHAO, X., GAO, L., AND LI, W. Evolving scheduling rules with gene expression programming for dynamic single-machine scheduling problems. *The International Journal of Advanced Manufacturing Technology 50* (2010), 729–747.

[104] NOWICKI, E., AND SMUTNICKI, C. A fast taboo search algorithm for the job shop problem. *Management Science 42*, 6 (1996), 797–813.

[105] PANWALKAR, S., AND ISKANDER, W. A survey of scheduling rules. *Operations Research 25* (1977), 45–61.

[106] PARK, J., NGUYEN, S., JOHNSTON, M., AND ZHANG, M. Evolving stochastic dispatching rules for order acceptance and scheduling via genetic programming. In *Proceedings of the 26th Australasian Joint Conference on Artificial Intelligence* (2013), pp. 478–489.

[107] PICKARDT, C., BRANKE, J., HILDEBRANDT, T., HEGER, J., AND SCHOLZ-REITER, B. Generating dispatching rules for semiconductor manufacturing to minimize weighted tardiness. In *Proceedings of the Winter Simulation Conference* (2010), pp. 2504–2515.

[108] PICKARDT, C., HILDEBRANDT, T., BRANKE, J., HEGER, J., AND SCHOLZ-REITER, B. Evolutionary generation of dispatching rule

sets for complex dynamic scheduling problems. *International Journal of Production Economics 145*, 1 (2013), 67–77.

[109] PILLAY, N. Evolving hyper-heuristics for the uncapacitated examination timetabling problem. *Journal of the Operational Research Society 63*, 1 (2012), 47–58.

[110] PINEDO, M. *Scheduling: Theory, Algorithms, and Systems*, 3rd ed. Springer, 2008.

[111] PINEDO, M., AND SINGER, M. A shifting bottleneck heuristic for minimizing the total weighted tardiness in a job shop. *Naval Research Logistics 46*, 1 (1999), 1–17.

[112] PIRAMUTHU, S., RAMAN, N., AND SHAW, M. J. Decision support system for scheduling a flexible flow system: Incorporation of feature construction. *Annals of Operations Research 78* (1998), 219–234.

[113] POLI, R. Parallel distributed genetic programming. Tech. Rep. Technical Report CSRP-96-15, University of Birmingham, 1996.

[114] POLI, R., LANGDON, W., AND MCPHEE, N. *A Field Guide to Genetic Programming*. Published via `http://lulu.com` and freely available at `http://www.gp-field-guide.org.uk`, 2008. (With contributions by J. R. Koza).

[115] POTTER, M. A., AND DE JONG, K. A. Cooperative coevolution: An architecture for evolving coadapted subcomponents. *Evolutionary Computation 8*, 1 (2000), 1–29.

[116] POTTS, C., AND STRUSEVICH, V. Fifty years of scheduling: a survey of milestones. *Journal of the Operational Research Society 60*, S1 (2009), 41–68.

[117] RAJENDRAN, C., AND HOLTHAUS, O. A comparative study of dispatching rules in dynamic flowshops and jobshops. *European Journal of Operational Research 116*, 1 (1999), 156–170.

[118] RAMASESH, R. Dynamic job shop scheduling: a survey of simulation research. *Omega 18*, 1 (1990), 43–57.

[119] RAMIREZ, R., AND PUIGGROS, M. An evolutionary computation approach to cognitive states classification. In *Proceedings of the IEEE Congress on Evolutionary Computation* (2007), IEEE, pp. 1793–1799.

[120] REMDE, S., COWLING, P., DAHAL, K., AND COLLEDGE, N. Exact/heuristic hybrids using rVNS and hyperheuristics for workforce scheduling. In *Evolutionary Computation in Combinatorial Optimization*. Springer, 2007, pp. 188–197.

[121] SELS, V., GHEYSEN, N., AND VANHOUCKE, M. A comparison of priority rules for the job shop scheduling problem under different flow time and tardiness-related objective functions. *International Journal of Production Research 50*, 15 (2012), 4255–4270.

[122] SHA, D., AND HSU, C.-Y. A hybrid particle swarm optimization for job shop scheduling problem. *Computers & Industrial Engineering 51*, 4 (2006), 791–808.

[123] SHEN, X., AND YAO, X. Mathematical modeling and multi-objective evolutionary algorithms applied to dynamic flexible job shop scheduling problems. *Information Sciences 298* (2015), 198–224.

[124] SHIUE, Y.-R., GUH, R.-S., AND LEE, K.-C. Development of machine learning based real time scheduling systems: using ensemble based on wrapper feature selection approach. *International Journal of Production Research 50*, 20 (2012), 5887–5905.

[125] SMITH, M. G., AND BULL, L. Genetic programming with a genetic algorithm for feature construction and selection. *Genetic Programming and Evolvable Machines 6*, 3 (2005), 265–281.

[126] TAY, J. C., AND HO, N. B. Evolving dispatching rules using genetic programming for solving multi-objective flexible job-shop problems. *Computers & Industrial Engineering 54* (2008), 453–473.

[127] VAESSENS, R. J. M., AARTS, E., AND LENSTRA, J. Job shop scheduling by local search. *INFORMS Journal on Computing 8* (1994), 302–317.

[128] VAN LAARHOVEN, P., AARTS, E. H. L., AND LENSTRA, J. K. Job shop scheduling by simulated annealing. *Operations Research 40*, 1 (1992), 113–125.

[129] VAZQUEZ RODRIGUEZ, J. A., AND OCHOA, G. On the automatic discovery of variants of the NEH procedure for flow shop scheduling using genetic programming. *Journal of the Operational Research Society 62*, 2 (2011), 381–396.

[130] VEPSALAINEN, A., AND MORTON, T. Priority rules for job shops with weighted tardiness costs. *Management Science 33* (1987), 1035–1047.

[131] VO, S. Meta-heuristics: The state of the art. In *Local Search for Planning and Scheduling*, A. Nareyek, Ed., vol. 2148 of *Lecture Notes in Computer Science*. Springer, 2001, pp. 1–23.

[132] WANG, L., AND ZHENG, D.-Z. An effective hybrid optimization strategy for job-shop scheduling problems. *Computers & Operations Research 28*, 6 (2001), 585–596.

[133] WANG, Z., TANG, K., AND YAO, X. Multi-objective approaches to optimal testing resource allocation in modular software systems. *IEEE Transactions on Reliability 59*, 3 (Sept 2010), 563–575.

[134] WECKMAN, G. R., GANDURI, C. V., AND KOONCE, D. A. A neural network job-shop scheduler. *Journal of Intelligent Manufacturing 19*, 2 (2008), 191–201.

[135] WHIGHAM, P. A. Grammatically-based genetic programming. In *Proceedings of the Workshop on Genetic Programming: From Theory to Real-World Applications* (1995), Morgan Kaufmann, pp. 33–41.

[136] WHITLEY, D., AND SUTTON, A. M. Genetic algorithms — a survey of models and methods. In *Handbook of Natural Computing* (2012), G. Rozenberg, T. Bäck, and J. N. Kok, Eds., Springer, pp. 637–671.

[137] WONG, P., AND ZHANG, M. Algebraic simplification of GP programs during evolution. In *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation* (2006), pp. 927–934.

[138] XIA, W., AND WU, Z. An effective hybrid optimization approach for multi-objective flexible job-shop scheduling problems. *Computers & Industrial Engineering 48*, 2 (2005), 409–425.

[139] XUE, B., ZHANG, M., AND BROWNE, W. N. Single feature ranking and binary particle swarm optimisation based feature subset ranking for feature selection. In *Proceedings of the Australasian Computer Science Conference* (2012), vol. 122 of *CRPIT*, pp. 27–36.

[140] YIN, W.-J., LIU, M., AND WU, C. Learning single-machine scheduling heuristics subject to machine breakdowns with genetic programming. In *Proceedings of the IEEE Congress on Evolutionary Computation* (2003), vol. 2, pp. 1050–1055.

[141] ZHANG, B.-T., AND MÜHLENBEIN, H. Balancing accuracy and parsimony in genetic programming. *Evolutionary Computation 3*, 1 (1995), 17–38.

[142] ZHANG, G., SHAO, X., LI, P., AND GAO, L. An effective hybrid particle swarm optimization algorithm for multi-objective flexible

job-shop scheduling problem. *Computers & Industrial Engineering 56*, 4 (2009), 1309–1318.

[143] ZHANG, M., AND WONG, P. Genetic programming for medical classification: a program simplification approach. *Genetic Programming and Evolvable Machines 9*, 3 (2008), 229–255.

[144] ZHANG, Q., AND LI, H. MOEA/D: A multiobjective evolutionary algorithm based on decomposition. *IEEE Transactions on Evolutionary Computation 11*, 6 (2007), 712–731.

[145] ZHOU, H., FENG, Y., AND HAN, L. The hybrid heuristic genetic algorithm for job shop scheduling. *Computers & Industrial Engineering 40*, 3 (2001), 191–200.

[146] ZITZLER, E., LAUMANNS, M., AND THIELE, L. SPEA2: Improving the strength pareto evolutionary algorithm for multiobjective optimization. In *Evolutionary Methods for Design, Optimisation, and Control with Applications to Industrial Problems* (2002), pp. 95–100.