Abstract

This thesis covers several topics relevant to the design of fuzzy systems using evolutionary algorithms (EAs), with application to control problems and constrained multiobjective problems.

Encoding is a fundamental part of any EA. The solution encoding, and its interactions with the EA’s operators, should be designed to minimise arbitrary search bias. A multidimensional encoding suitable for fully specified fuzzy logic rulebases is investigated, and is shown to have better convergence than traditional single-dimensional encoding. By comparison with 2-point and uniform crossover, the improvement is attributed to the elimination of dimensional encoding bias.

The “curse of dimensionality” is the exponential growth of the search space as the number of decision variables increases. In particular, encoding a fully specified fuzzy logic rulebase can result in a prohibitively large search space. Cooperative coevolution and hierarchical fuzzy rulebases both mitigate the curse of dimensionality through modularity, for evolutionary algorithms and fuzzy systems respectively, and these techniques are shown to be highly compatible with one another. The evolutionary convergence, hierarchical design, and opportunity for parallel computation are analysed for the combined techniques.
Most real-world problems are characterised by multiple, conflicting objectives, and are subject to multiple constraints. **Multiobjective optimisation**, particularly constrained multiobjective optimisation, is investigated using control and function optimisation benchmark problems.

Two multiobjective diversity measures, hypervolume and distance-to-neighbours, are quantitatively analysed: hypervolume is found to more accurately identify the Pareto front, and distance-to-neighbours is found to distribute solutions more uniformly. The inverted pendulum is used as a case study to qualitatively investigate multiobjective design and optimisation of a control problem.

This thesis presents the reconciliation of objective optimisation and constraint satisfaction as the main challenge facing any constrained multiobjective optimisation algorithm, and identifies and investigates two strategies for reconciliation: extended dominance, and blended space.

A novel blended space algorithm – the Blended Rank Evolutionary Algorithm (BREA) – is proposed. BREA dynamically maintains trade-offs between objective optimisation, constraint satisfaction, and population diversity, in order to better identify the Pareto optimal set of solutions in difficult problems. BREA is very favourably compared to the extended dominance algorithm NSGA-II on the nonlinear crop-rotation problem, improving both solution quality and reliability.
Coevolution and Encoding of Fuzzy Systems, and Multiobjective Optimisation

Nicholas Young
B. Information Technology (Honours)

Thesis submitted for the Degree of Doctor of Philosophy
Faculty of Business and Informatics
Central Queensland University
February 2007
Contents

Chapter 1. Introduction ................................................................................................... 1
  1.1 Preliminaries ............................................................................................................. 1
  1.2 Scope ......................................................................................................................... 5
  1.3 Overview of thesis .................................................................................................... 6

Chapter 2. Introduction to Fuzzy Systems ................................................................... 10
  2.1 Fuzzy logic .............................................................................................................. 10
    2.1.1 Fuzzy control ................................................................................................... 11
  2.2 Fuzzy sets and membership functions .................................................................... 11
    2.2.1 Trapezoidal and triangular membership functions .......................................... 13
    2.2.2 Gaussian membership functions ...................................................................... 13
    2.2.3 Sigmoidal membership functions .................................................................... 13
  2.3 Fuzzy system ........................................................................................................... 14
  2.4 Fuzzy if-then rules .................................................................................................. 14
  2.5 Rulebase .................................................................................................................. 15
    2.5.1 Local vs. global semantics ............................................................................... 15
    2.5.2 Unstructured vs. structured .............................................................................. 16
  2.6 Inference engine ...................................................................................................... 17
    2.6.1 Consequent ....................................................................................................... 17
    2.6.2 Matching degree ............................................................................................... 18
    2.6.3 Interpolation ..................................................................................................... 18
    2.6.4 Defuzzification ................................................................................................. 19
    2.6.5 Mamdani inference engine ............................................................................... 20
    2.6.6 Takagi-Sugeno-Kang (TSK) inference engine ................................................ 20
    2.6.7 Standard Additive Model (SAM) inference engine ......................................... 21
  2.7 Hierarchical rulebases ............................................................................................. 21
    2.7.1 Topologies ........................................................................................................ 22
    2.7.2 Intermediate outputs ......................................................................................... 23
    2.7.3 Limitations of hierarchical rulebases .................................................................. 24
  2.8 Knowledgebase discovery ...................................................................................... 24
    2.8.1 Extracting human expertise .............................................................................. 25
    2.8.2 Automatic learning ........................................................................................... 25
  2.9 Summary ................................................................................................................. 26

Chapter 3. Introduction to Evolutionary Algorithms ................................................. 28
  3.1 Search algorithms .................................................................................................... 28
    3.1.1 Single solution search algorithms ..................................................................... 29
    3.1.2 Population-based search algorithms ................................................................. 30
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.2 Evolutionary algorithms</td>
<td>30</td>
</tr>
<tr>
<td>3.2.1 Types of evolutionary algorithms</td>
<td>31</td>
</tr>
<tr>
<td>3.2.2 A simple evolutionary algorithm</td>
<td>32</td>
</tr>
<tr>
<td>3.2.3 Encoding</td>
<td>33</td>
</tr>
<tr>
<td>3.2.4 Fitness</td>
<td>34</td>
</tr>
<tr>
<td>3.2.5 Selection for reproduction</td>
<td>35</td>
</tr>
<tr>
<td>3.2.6 Reproduction</td>
<td>36</td>
</tr>
<tr>
<td>3.2.7 Selection for survival</td>
<td>39</td>
</tr>
<tr>
<td>3.2.8 Diversity</td>
<td>39</td>
</tr>
<tr>
<td>3.2.9 Convergence</td>
<td>40</td>
</tr>
<tr>
<td>3.2.10 Epistasis</td>
<td>41</td>
</tr>
<tr>
<td>3.2.11 EA design</td>
<td>42</td>
</tr>
<tr>
<td>3.3 Parallelism</td>
<td>43</td>
</tr>
<tr>
<td>3.3.1 Parallel supercomputers</td>
<td>44</td>
</tr>
<tr>
<td>3.3.2 EAs and parallelism</td>
<td>47</td>
</tr>
<tr>
<td>3.4 Cooperative coevolution</td>
<td>50</td>
</tr>
<tr>
<td>3.4.1 Curse of dimensionality</td>
<td>51</td>
</tr>
<tr>
<td>3.4.2 Modularity</td>
<td>52</td>
</tr>
<tr>
<td>3.4.3 Apportionment of credit</td>
<td>53</td>
</tr>
<tr>
<td>3.4.4 Decomposition</td>
<td>54</td>
</tr>
<tr>
<td>3.5 Multiobjective evolutionary algorithms</td>
<td>56</td>
</tr>
<tr>
<td>3.5.1 Approaches to multiobjective optimisation</td>
<td>57</td>
</tr>
<tr>
<td>3.5.2 Pareto methods</td>
<td>58</td>
</tr>
<tr>
<td>3.6 Constrained multiobjective evolutionary algorithms</td>
<td>65</td>
</tr>
<tr>
<td>3.6.1 Handling constraints</td>
<td>66</td>
</tr>
<tr>
<td>3.6.2 Constraint space</td>
<td>67</td>
</tr>
<tr>
<td>3.7 Summary</td>
<td>68</td>
</tr>
<tr>
<td>Chapter 4. Evolutionary Learning of Fuzzy Logic Systems</td>
<td>70</td>
</tr>
<tr>
<td>4.1 Targets for optimisation</td>
<td>70</td>
</tr>
<tr>
<td>4.2 Learning membership functions</td>
<td>71</td>
</tr>
<tr>
<td>4.3 Learning rules</td>
<td>72</td>
</tr>
<tr>
<td>4.4 Learning membership functions and rules</td>
<td>75</td>
</tr>
<tr>
<td>4.5 Learning structure</td>
<td>77</td>
</tr>
<tr>
<td>4.6 Summary</td>
<td>78</td>
</tr>
<tr>
<td>Chapter 5. Inverted Pendulum Problem</td>
<td>81</td>
</tr>
<tr>
<td>5.1 Introduction</td>
<td>81</td>
</tr>
<tr>
<td>5.2 Model</td>
<td>82</td>
</tr>
<tr>
<td>5.3 Fuzzy system</td>
<td>82</td>
</tr>
<tr>
<td>5.4 Simulation</td>
<td>84</td>
</tr>
<tr>
<td>5.4.1 Success and failure</td>
<td>84</td>
</tr>
<tr>
<td>5.5 Fitness evaluation and weights</td>
<td>85</td>
</tr>
<tr>
<td>5.6 Test-cases</td>
<td>87</td>
</tr>
<tr>
<td>5.7 Summary</td>
<td>88</td>
</tr>
<tr>
<td>Chapter 6. Multidimensional Encoding</td>
<td>89</td>
</tr>
</tbody>
</table>
List of Figures

Figure 2.1. Membership functions for a fuzzy set and crisp set ................................................. 12
Figure 2.2. Three overlapping fuzzy sets together covering the entire x domain ......................... 12
Figure 2.3. Trapezoidal membership function ................................................................. 13
Figure 2.4. Gaussian membership function ............................................................................. 13
Figure 2.5. Sigmoidal membership function ............................................................................. 13
Figure 2.6. Fuzzy system diagram ......................................................................................... 14
Figure 2.7. Single-layer rulebase ............................................................................................ 22
Figure 2.8. Two-layer rulebase ............................................................................................... 22
Figure 2.9. Hierarchical rulebase ............................................................................................. 22
Figure 3.1. 2-point crossover .................................................................................................. 38
Figure 3.2. EA design framework ............................................................................................ 42
Figure 3.3. Cooperative coevolution framework ....................................................................... 51
Figure 3.4. Visualisation of domination .................................................................................... 60
Figure 5.1. State variables and control force of the inverted pendulum problem ..................... 81
Figure 5.2. Membership functions for input $x_1$ ................................................................. 83
Figure 5.3. Membership functions for input $x_2$ ................................................................. 83
Figure 5.4. Membership functions for input $x_3$ ................................................................. 83
Figure 5.5. Membership functions for input $x_4$ ................................................................. 83
Figure 5.6. Centroids of membership functions for output $y$ .................................................... 84
Figure 6.1. 2D rulebase ........................................................................................................... 92
Figure 6.2. 3D rulebase ........................................................................................................... 92
Figure 6.3. Table form of a 2D rulebase .................................................................................... 93
Figure 6.4. Design of an EA, distinguishing between encoding variables and structure .............. 94
Figure 6.5. A schema with 5 specified decision-variables .......................................................... 96
Figure 6.6. 1-point crossover as a string .................................................................................... 99
Figure 6.7. 2-point crossover as a string .................................................................................... 99
Figure 6.8. 2-point crossover as a loop .................................................................................... 100
Figure 6.9. 3D crossover along dimension $x_2$ ....................................................................... 116
Figure 6.10. “Fedora” function ............................................................................................... 120
Figure 6.11. Fedora function – Fitness of best solution ............................................................ 121
Figure 6.12. Train data – Fitness of best solution ....................................................................... 123
Figure 6.13. Inverted Pendulum – Fitness of best solution ...................................................... 126
Figure 7.1. Some different possible topologies for a 4-input fuzzy system ................................. 133
Figure 7.2. Cooperative coevolution framework ....................................................................... 135
Figure 7.3. Scope for parallelism ............................................................................................. 141
Figure 7.4. Centroids of membership functions for output $y$ ................................................... 144
Figure 7.5. Previous topology for inverted pendulum problem ............................................... 145
Figure 7.6. Topology for inverted pendulum problem used in this study. ..................... 145  
Figure 7.7. Centroids of membership functions for output $y_A$. ....................................... 147  
Figure 7.8. Membership functions covering input $y_A$. ............................................................ 147  
Figure 7.9. Fitness of best solution – by generations. .......................................................... 155  
Figure 7.10. Fitness of best solution – by fitness evaluations. ........................................... 156  
Figure 7.11. State and control for inverted pendulum. ................................................... 160  
Figure 7.12. State and control for inverted pendulum. ................................................... 160  
Figure 8.1. 4D visualisation of non-dominated front...................................................... 182  
Figure 8.2. 4D visualisation of non-dominated front. View from the right.................... 182  
Figure 8.3. 4D visualisation of non-dominated front. View from above....................... 183  
Figure 8.4. 4D visualisation of non-dominated front. View from the front.................... 183  
Figure 8.5. Graph features illustrating different trade-offs ............................................. 184  
Figure 8.6. Visualisation of domination ................................................................. 186  
Figure 8.7. Distance-to-neighbours metric ..................................................................... 189  
Figure 8.8. Hypervolume metric ..................................................................................... 190  
Figure 8.9. Left: Pareto front and an approximation set of 10 points. Centre:  
Piecewise linear approximation. Right: Dominance approximation. ....................... 193  
Figure 8.10. Left: Pareto front and an approximation set of 10 points. Centre:  
Worst 2 points removed, by distance diversity score. Right: Worst 2 points  
removed, by hypervolume diversity score. .............................................................. 194  
Figure 8.11. Pareto front and approximation sets (left: selected by distance; right:  
selected by hypervolume). ....................................................................................... 194  
Figure 8.12. Progress of constraint satisfaction of best solution during early  
generations. ........................................................................................................... 207  
Figure 8.13. Random wander of best solution through objective space during early  
generations. ........................................................................................................... 207  
Figure 8.14. Non-dominated solutions found after 80 (left), 110 (centre), and 1000  
(right) generations ................................................................................................ 207  
Figure 8.15. One typical run each of hypervolume and distance diversity metrics....... 209  
Figure 8.16. Cumulative solution set from 10 runs each of hypervolume and  
distance diversity metrics ..................................................................................... 211  
Figure 8.17. Only non-dominated solutions from the cumulative solution set ...... 212  
Figure 9.1. A two-objective, two-constraint problem based on CTP ............................. 233  
Figure 9.2. NSGA-II vs. BSEA on CTP-6. .................................................................... 242  
Figure 9.3. NSGA-II vs. BSEA on CTP-7 .................................................................... 243  
Figure 9.4. NSGA-II vs. BSEA on CTP-8 .................................................................... 243  
Figure 9.5. NSGA-II on the restricted CTP-8 ................................................................. 244  
Figure 9.6. BSEA on the restricted CTP-8. .................................................................... 244  
Figure 9.7. The final populations after 1000 generations of NSGA-II and BSEA,  
on the restricted CTP-8. ....................................................................................... 245  
Figure 9.8. BSEA vs. NSGA-II over 2,000 generations .................................................. 248  
Figure 9.9. BSEA vs. NSGA-II over 200,000 generations ............................................. 249  
Figure 9.10. Detail of non-dominated fronts from BSEA ............................................. 250  
Figure 9.11. Invisible creep caused by allowing constraint satisfaction to be traded  
for objective optimisation ..................................................................................... 256
Figure 9.12. Invisible creep caused by allowing diversity alone to differentiate the non-dominated rank ................................................................. 258
Figure 9.13. The diversity, constraint, and objective weights as functions of $\alpha$ .......... 265
Figure 9.14. BREA vs. BSEA vs. NSGA-II over 2,000 generations............................ 271
Figure 9.15. BREA vs. BSEA vs. NSGA-II over 200,000 generations....................... 272
Figure 9.16. NSGA-II over 200,000 generations vs. NSGA-II over 2,000 generations ......................................................................................................................... 276
Figure 9.17. BSEA over 200,000 generations vs. BSEA over 2,000 generations ...... 277
Figure 9.18. BREA over 200,000 generations vs. BREA over 2,000 generations ...... 278
Figure 9.19. BREA with 100 solutions over 2,000 generations vs. NSGA-II with 100 solutions over 20,000 generations vs. NSGA-II with 800 solutions over 2,000 generations ................................................................. 280
List of Tables

Table 5.1. Test cases ................................................................. 88
Table 6.1. Crossover operator characteristics ........................................ 117
Table 6.2. Algorithm parameters ......................................................... 119
Table 6.3. Fedora function – Fitness of best solution ......................... 122
Table 6.4. Analysis of variance P values, at 200 generations ............... 122
Table 6.5. Train data – Fitness of best solution .................................... 124
Table 6.6. Analysis of variance P values, at 500 generations ............... 124
Table 6.7. Fitness of best individual ...................................................... 126
Table 6.8. Analysis of variance P values, at 200 generations ............... 126
Table 6.9. Inter-node send/receive times ................................................ 150
Table 7.1. Algorithm parameters ......................................................... 153
Table 7.2. Mutation schedule ......................................................... 153
Table 7.3. Algorithm parameters ......................................................... 155
Table 7.4. Analysis of variance P values, for final solutions .................. 155
Table 7.5. Mean and standard deviation of final solutions ...................... 157
Table 7.6. Analysis of variance P values, at 25,000 evaluations ............ 157
Table 7.7. Analysis of variance P values, at 50,000 evaluations ............ 157
Table 7.8. Analysis of variance P values, at 100,000 evaluations .......... 157
Table 7.9. Rulebase A consequents, grouped by $x_1$ then $x_2$ (row-major) ................................................................. 161
Table 8.1. Test cases ................................................................. 178
Table 8.2. Relevant Pareto relationships ............................................... 199
Table 8.3. Algorithm parameters ......................................................... 206
Table 8.4. Comparison metrics ......................................................... 214
Table 9.1. Algorithm Parameters ......................................................... 240
Table 9.2. CTP Parameters .............................................................. 241
Table 9.3. Algorithm parameters ......................................................... 269
Table 9.4. Mean and standard deviation of hypervolume metric ............ 271
Table 9.5. Analysis of variance P values .............................................. 271
Table 9.6. Mean and standard deviation of hypervolume metric ............ 272
Table 9.7. Analysis of variance P values .............................................. 272
Table 9.8. Mean and standard deviation of hypervolume metric ............ 276
Table 9.9. Analysis of variance P values .............................................. 276
Table 9.10. Mean and standard deviation of hypervolume metric .......... 277
Table 9.11. Analysis of variance P values .............................................. 277
Table 9.12. Mean and standard deviation of hypervolume metric .......... 278
Table 9.13. Analysis of variance P values .............................................. 278
Table 9.14. Mean and standard deviation of hypervolume metric .......... 280
Table 9.15. Analysis of variance P values .............................................. 280
Acronyms

GA – Genetic Algorithm
EA – Evolutionary Algorithm
CMEA – Constrained Multiobjective Evolutionary Algorithm
CTP – Constrained Test Problem [31]
NSGA-II – Nondominated Sorting Genetic Algorithm II [28]
BSEA – Blended Space Evolutionary Algorithm (a novel algorithm)
BREA – Blended Rank Evolutionary Algorithm (a novel algorithm)
Acknowledgement

This work owes much to the guidance, support, and motivation from my supervisor, Dr. Russel J. Stonier. Thanks also to my co-supervisors, Mr. Stephen Smith and Dr. Peter Thomas, for their advice.

Thanks to my fellow students and friends, especially Sarah, who have provided distraction and inspiration in roughly equal measure. Thanks to my family for their support and patience! Thanks most of all to my mother, who shared the experience of working towards a PhD, but who can share our experiences no more.

And of course, thanks to the examiners of this thesis for their recommendations, and to the wider research community from which so many ideas are drawn.
Declaration

This thesis contains no material which has been accepted for the award of another degree. Furthermore, this thesis does not contain any material previously published or written by another person, except where due reference and acknowledgement is made in the text.

....................................

Nicholas Young
This research has been published in:

1. Young, N. and Stonier, R.J., Co-evolutionary learning and hierarchical fuzzy control for the inverted pendulum. in *International Congress on Evolutionary Computation*, (Canberra, 2003), IEEE-Press, 467 - 473.

2. Young, N. and Stonier, R.J., Multi-dimensional encoding to reduce bias in fuzzy knowledge-bases. in *7th Asia-Pacific Conference on Complex Systems*, (Cairns, 2004), ISBN 1 87667496 2, 196-206.

3. Young, N., Blended Ranking to Cross Infeasible Regions in Constrained Multiobjective Problems. in *Computational Intelligence for Modelling Control and Automation*, (Vienna, Austria, 2005), IEEE-Press.

4. Young, N. and Stonier, R.J., Blended Rank Evolutionary Algorithm for the Constrained Multiobjective Crop Rotation Problem. in *Computational Intelligence for Modelling Control and Automation*, (Sydney, Australia, 2006), IEEE-Press.
Chapter 1. Introduction

This thesis presents research into three evolutionary algorithm (EA) techniques relevant to the application of EAs to fuzzy systems: multidimensional encoding; cooperative coevolution; and constrained multiobjective optimisation. In the process, this thesis will also discuss hierarchical fuzzy rulebases and EA parallelism.

The most significant original contributions of this thesis are in the final chapters on constrained multiobjective optimisation. This area was so promising in its own right that chapter 9 focuses on constrained multiobjective EAs (CMEAs) to the exclusion of fuzzy systems, and a novel CMEA called Blended Rank Evolutionary Algorithm is presented.

1.1 Preliminaries

Fuzzy logic and evolutionary computation belong to the broad field of “soft computing” which uses computational intelligence techniques to exploit tolerances for imprecision and uncertainty to solve problems for which rigorous, exact solutions are impractical to obtain.

Fuzzy systems

Fuzzy systems [101] are essentially function approximators, and as such may be applied to a wide range of problems, in which it might be difficult to express an appropriate function using any other technique. These problems include: control, signal processing, pattern recognition and classification, management and decision-making. As function
approximators, fuzzy systems are comparable to soft computing techniques such as neural networks, classifier systems, and expert systems, or to statistical techniques such as regression analysis and curve fitting.

Fuzzy logic, based around the concept of fuzzy sets, was introduced by Lotfi Zadeh [111, 112] to handle inherent uncertainty and imprecision in knowledge. A fuzzy system uses rules relating fuzzy sets to describe input-output mappings. The fuzziness of the sets allows the output to be interpolated between multiple rules, resulting in a smooth, continuous output from a discrete set of rules.

**Evolutionary algorithms**

Evolutionary algorithms [38, 65] are essentially function *optimisers* – search algorithms – and as such may be applied to a similarly wide range of problems, in which a function is given but the values for decision-variables that optimise the function are unknown. These problems include: engineering optimisation, scheduling, bioinformatics, evolvable hardware, and even art. EAs, like most soft computing techniques, are usually inferior to specialised techniques where such exist, but are generally robust performers where no other technique exists – typically for complex, multimodal, non-linear problems. As search algorithms/function optimisers, EAs are comparable to soft computing techniques such as simulated annealing, tabu search, and swarm intelligence, or to mathematical techniques such as gradient descent and simplex method.

EAs are a class of related search algorithms that all maintain a population of trial solutions to a problem. Darwinian “survival of the fittest” selection, combined with sexual reproduction and mutation, causes the population to evolve towards fitter solutions. In its simplest form, the fitness function is the same as the function to optimise.
EAs have diversified to cover a wide range of problems with specific characteristics. These characteristics include: multimodal problems, dynamic problems, non-traditional representations, and (importantly to this thesis) constrained and multiobjective optimisation.

**Evolutionary fuzzy systems**

Function approximation by fuzzy systems and function optimisation by evolutionary algorithms can effectively be combined in many ways [63]. This work uses them in two: function approximation problems and control problems.

In **function approximation**, the fuzzy system approximates a function and the EA optimises the internal parameters of the fuzzy system in order to minimise the error between the fuzzy system’s approximation and a target function. For example, this work evolves a fuzzy system to model an unknown physical process (train axle unloading) from observed data.

In **control**, the fuzzy system implements a control function and, as above, the EA optimises the fuzzy system. Unlike function approximation, there is no target function to compare the control function to; instead there is an evaluation function which is a function of the control function. The EA evolves a control function (implemented in a fuzzy system) to optimise that evaluation function. For example, this work evolves a fuzzy system to control the inverted pendulum, where a freely swinging pole atop a cart must be balanced upright by controlling the movement of the cart.
Advanced techniques

As evolved fuzzy systems have such broad applicability, improvements to the basic algorithms are valuable. There are many models for fuzzy systems and many different evolutionary algorithms designed to improve different aspects of performance.

On the fuzzy systems side, hierarchical rulebases [69] are intended to improve the modularity of fuzzy systems and reduce the number of rules required, making the fuzzy system easier to implement.

On the evolutionary algorithms side, cooperative coevolution [75] is intended to exploit the modularity present in many systems to effectively reduce the size of the search space, making evolution faster.

Evolutionary algorithms are inherently trivial to implement using parallel computation [2] to reduce the run-time of an algorithm. Cooperative coevolution is particularly well suited to parallelisation.

Multiobjective optimisation [27], in the broadest sense, is a way of formulating problems that more naturally and effectively defines what is desired of a solution, by specifying multiple independent objective functions to be optimised. These objectives may conflict with one another, making it impossible to optimise all functions with a single solution. The conflicting objective functions define a trade-off surface of potential solutions, called the Pareto front.

EAs are well suited to multiobjective optimisation due to their population-based search, allowing it to search for multiple solutions that optimise different objective functions simultaneously. The goal of a Pareto-based multiobjective EA is to find solutions distributed across the Pareto front trade-off surface.
Constrained multiobjective optimisation adds constraints that must be satisfied, in addition to optimising the objective functions. Constraints determine whether a trial solution is feasible or not. Traditionally constraints have been formulated as penalties applied back to fitness function, but this is a crude way to manage multiple constraints and multiple objectives. Highly constrained, multiobjective problems require a more sophisticated strategy.

1.2 Scope

The two main problems used in this thesis are the inverted pendulum problem, and the nonlinear crop rotation problem. The inverted pendulum problem is solved by evolving a fuzzy controller, and used to investigate the evolution of fuzzy systems. The nonlinear crop rotation problem is solved by a constrained multiobjective EA, and used to investigate constrained multiobjective optimisation independently of fuzzy systems. However, this work is not about solving particular application problems. Rather, this work investigates the intersection of fuzzy systems, evolutionary algorithms, and multiobjective optimisation by applying a range of evolutionary computation techniques to a range of fuzzy systems.

This work should mainly be appreciated from the evolutionary computation field, as the majority of observations and innovations are on the evolutionary algorithm side, not the fuzzy systems side. Indeed, the final research chapter of this thesis is concerned solely with constrained multiobjective EAs, and fuzzy systems do not appear.

However, it would be overgeneralising to discount the significance of fuzzy systems to this work. Many of the evolutionary techniques investigated and innovated upon are done
so specifically in the context of fuzzy systems, and those results do not necessarily
generalise to all evolutionary algorithms.

1.3 Overview of thesis

This thesis is broadly organised into introductory chapters and research chapters. The
introductory chapters (2-5) provide the background knowledge to put the research
chapters (6-9) into context. Each research chapter also includes additional background
knowledge specific to that chapter.

Chapter 1.

This chapter gives some preliminary information on the use of fuzzy systems,
evolutionary computation, and multiobjective optimisation, and the intersection of these
topics. It gives this overview and identifies the relationships between subsequent
chapters.

Chapter 2.

Chapter 2 introduces fuzzy systems, describing types of membership functions, rulebases,
and inference engines. A specific rulebase structure, the hierarchical rulebase, is
introduced. Methods for discovering appropriate fuzzy systems are briefly described for
completeness, though this work uses EAs exclusively.

Various fuzzy systems are used in the research chapters 6-8, as problems to be optimised
by EAs.

Chapter 3.

Chapter 3 introduces evolutionary algorithms. After placing evolutionary algorithms in
the context of heuristic search algorithms, the chapter describes the operation of
traditional EAs, including common choices for operators. The concepts of diversity, convergence, and epistasis are introduced. Further attention is given to the topics of parallelism, cooperative coevolution, multiobjective optimisation, and constrained multiobjective optimisation.

Various EAs are used or referred to in every subsequent chapter.

Chapter 4.

Chapter 4 discusses the application of EAs to discovering fuzzy systems. It identifies three targets for optimisation: membership functions, rules, and structure. This work uses EAs to search for rules only; chapters 6-8 use various EAs to evolve rulebases for various fuzzy systems.

Chapter 5.

Chapter 5 introduces the inverted pendulum problem as a fuzzy control system to be optimised by EA. The chapter gives: a mathematical model of the problem; a simulation to evaluate potential controllers; a basic fuzzy controller implementation (which is modified in chapter 7); and a function for measuring fitness for evolution (which is modified in chapter 8).

The inverted pendulum problem is used in chapters 6-8 to investigate various fuzzy and EA techniques.

Chapter 6.

Chapter 6 researches multidimensional encoding and crossover. It investigates sources of bias – both desirable and undesirable – that exist in the encoding and are expressed through crossover, in the context of fuzzy rulebases for function approximation and
control problems. Specifically, it investigates the effect of unwanted dimensional bias and useful epistatic bias, and proposes a multidimensional encoding and crossover intended to remove dimensional bias whilst exploiting epistatic bias. The multidimensional approach is compared to traditional vector-encoding and \( n \)-point crossover, and uniform crossover. Multidimensional encoding and crossover is adopted for subsequent research chapters.

**Chapter 7.**

Chapter 7 researches the combination of hierarchical fuzzy rulebases and cooperative coevolution. One unresolved difficulty for both hierarchical rulebases and cooperative coevolution is determining an appropriate problem decomposition; this chapter proposes that both techniques should naturally use the same decomposition, as both techniques require that the degree of interdependence, or epistasis, should be minimised between modules.

Using the inverted pendulum problem, the chapter tests several hypotheses about cooperative coevolution; investigates the role of intermediate variables in hierarchical rulebase decomposition; and investigates the inherent parallelism of EAs in general and cooperative coevolution in particular.

**Chapter 8.**

Chapter 8 researches the application of constrained multiobjective optimisation to the inverted pendulum problem. In doing so, it qualitatively investigates the diverse range of possible control strategies for the inverted pendulum problem, and quantitatively investigates the performance and other characteristics of two diversity measures for multiobjective EAs.
Chapter 8 also acts as an extended introduction to chapter 9, which continues the research into constrained multiobjective optimisation.

**Chapter 9.**

Chapter 9 researches the strategies used by constrained multiobjective EAs to reconcile the often conflicting goals of constraint satisfaction and objective optimisation. In particular, a novel algorithm based on a “blended space” strategy is proposed and contrasted with a popular algorithm based on the “extended dominance” strategy. Unlike previous chapters, chapter 9 does not use fuzzy systems to investigate the EAs. Instead it uses two benchmark problems specifically designed to illustrate the behaviour of constrained multiobjective EAs (as opposed to multiobjective EAs). As such, chapter 8 also serves as a bridge between the previous chapters, which all investigate EAs in the context of fuzzy systems, and chapter 9, which is focussed solely upon constrained multiobjective EAs.

**Chapter 10.**

Chapter 10 summarises the findings from the research chapters 6-9 and identifies future avenues of research.
Chapter 2. Introduction to Fuzzy Systems

2.1 Fuzzy logic

Fuzzy logic is most narrowly defined as an extension of classical two-valued logic to allow reasoning under uncertainty. Fuzzy logic replaces crisp sets, to which an element either certainly belongs or certainly does not, with fuzzy sets, to which an element belongs with some degree of certainty. Fuzzy logic also provides all the “fuzzy” operations required to work with these fuzzy sets. It is therefore very appropriately applied to problems that inherently involve imprecise data or ill-defined concepts – fuzzy logic used to handle uncertainty.

However, fuzzy logic is also used to approximate precise systems when other models are unacceptable. In some problems no more precise model exists; in other problems more precise models are too costly. Industry has very often found it beneficial to introduce imprecision (in the form of fuzzy logic) into otherwise precisely specified problems as a way to reduce cost. Fuzzy logic can provide an acceptably precise solution to a problem for an acceptable cost. It is this second use of fuzzy logic, as an approximator, that is of relevance to this work – fuzzy logic used to handle complexity.

It has been well established that fuzzy systems are universal function approximators, capable of approximating a continuous function to any arbitrary precision [11, 59, 102]. As such, they have been applied to a wide variety of fields including control, signal
processing, pattern recognition and classification, management and decision-making [105].

2.1.1 Fuzzy control
This work uses fuzzy logic primarily for control. A control problem can be stated as a system of equations governing a number of state variables, one (or more) of which is a control variable: an independent variable able to take any value (within constraints).

A fuzzy logic controller, therefore, approximates a control function, in which the control variable(s) value is a function of the state variables at any given time. The task of the controller is to implement a control function that brings each state variable to some desired value.

Chapter 5 describes a particular control problem – the inverted pendulum problem – that is used in chapters 6-8.

2.2 Fuzzy sets and membership functions
The essence of fuzzy logic is the fuzzy set [112]. A fuzzy set is a set that contains values with some degree of certainty, or membership. A membership function, \( \mu_i(x) \) gives the degree of membership for each element, \( x \), in belonging to the set \( i \). It is a continuous function in the range \([0,1]\). 1 indicates certain membership; 0 indicates certain non-membership; values in between indicate the degree of certainty. The terms fuzzy set and membership function are often used synonymously.

Figure 2.1 is a simple comparison of a crisp set and a fuzzy set – more accurately, a comparison of the membership function describing a crisp set and a fuzzy set.
Multiple fuzzy sets are used to partition the entire domain of a crisp variable. Fuzzy sets may overlap one another, so that an element has some positive degree of membership in multiple sets. Figure 2.2 shows three fuzzy sets (A, B, C) such that the element $p$ has a membership of 0.75 in set $A$, 0.25 in set $B$, and 0.0 in set $C$.

When fuzzy logic is used in problems with inherently vague concepts, membership sets are often given linguistic names, such as {hot, cold} or {small, medium, large}. Naming the membership sets makes it easier to capture and express human knowledge.

There are many ways of defining the membership function for a fuzzy set. Figure 2.3 shows a trapezoidal membership function, defined by four parameters ($a$, $b$, $c$, $d$). Membership functions can also be triangular, Gaussian, and sigmoidal, among others.

When designing a fuzzy system, it is not obvious a) what the best function type is for a given fuzzy set; and b) what the best parameters are for a given function type for a given membership set. Following are short descriptions of the most commonly used membership function types.
2.2.1 Trapezoidal and triangular membership functions
Trapezoidal membership functions are defined by four parameters (a, b, c, d). Triangular membership functions are a special case of trapezoidal functions where b = c. These functions are chosen for their simplicity and computational efficiency.

![Trapezoidal membership function](image)

**Figure 2.3.** Trapezoidal membership function.

2.2.2 Gaussian membership functions
Gaussian membership functions are defined by (m, σ) where m is the centre of the normal distribution and σ is the standard deviation. A Gaussian function, which continuously covers the entire variable domain, may improve the accuracy of a fuzzy system.

![Gaussian membership function](image)

**Figure 2.4.** Gaussian membership function.

2.2.3 Sigmoidal membership functions
Sigmoidal membership functions are defined by (a, c) where c is the value at which membership is 0.5, and a controls the abruptness of the transition. Sigmoidal membership functions can be used for sets covering the extremes of the variable domain.

![Sigmoidal membership function](image)

**Figure 2.5.** Sigmoidal membership function.
2.3 Fuzzy system

Fuzzy logic at its simplest is the logic of fuzzy sets. However, for the remainder of this work, “fuzzy system” will be used to refer to a system of if-then rules expressed using fuzzy sets. These systems of fuzzy if-then rules, together with a model for interpreting the rules, are used to capture human knowledge or approximate complex functions. These fuzzy logic systems are the most visible use of fuzzy logic due to their wide range of successful applications.

The components of a fuzzy logic system are:

- A knowledge base, which is a rulebase of if-then rules, and also all of the fuzzy sets used by the rules.

- An inference engine, which specifies the form of the consequents; how matching degrees are calculated; and how a final crisp output is calculated.

![Fuzzy system diagram.](image)

2.4 Fuzzy if-then rules

A fuzzy if-then rule captures one piece of knowledge (or approximates one section of a function). It is an input-output mapping.

A fuzzy if-then rule has a condition, or antecedent (input), and a consequent (output). A single rule has the form:

if antecedent then consequent

or in more detail:
if \( x_1 \) is \( A_1 \) and \( x_2 \) is \( A_2 \) ... and \( x_N \) is \( A_N \) then \textit{consequent}

where \( x_1, ... , x_N \) are input variables and \( A_1, ... , A_N \) are fuzzy sets. The definition of the consequent depends on the \textit{inference engine} used, but is typically defined using either fuzzy sets or a linear function of the input.

The nature of fuzzy sets means that a given input will not always have a degree of membership of exactly 1.0 in any rule’s fuzzy sets, and a given input may have some nonzero degree of membership in many different rules. The inference engine combines the input’s degrees of membership, \( \mu_{a_1}(x_1), \mu_{a_2}(x_2), ... , \mu_{a_N}(x_N) \) in all sets \( A_1, A_2, ... , A_N \) of a rule, to determine the overall relevance, or \textit{matching degree}, of the rule.

### 2.5 Rulebase

A \textit{rulebase} is a collection of fuzzy if-then rules. With sufficient rules and membership sets, the rulebase can provide a complete input-output mapping. This allows a fuzzy system to be a universal function approximator, with a precision limited by the number of rules and fuzzy sets [11, 59, 102].

A rulebase is \textit{complete} if, for any given possible combination of input values, it contains at least one rule with non-zero matching degree. A rulebase should have overlapping rules – rules whose antecedent fuzzy sets overlap, to ensure a smoothly varying output [105].

#### 2.5.1 Local vs. global semantics

A single rule, \( R_i \), in the rulebase is commonly written ([101]) as:

\[ R_i: \text{ if } x_1 \text{ is } A_{i,1} \text{ and } x_2 \text{ is } A_{i,2} \text{ ... and } x_N \text{ is } A_{i,N} \text{ then consequent} \]

where \( x_1, ... , x_N \) are input variables, and \( A_{i,1}, ... , A_{i,N} \) are fuzzy sets.
The fuzzy sets in the rulebase (of \( L \) rules), \( A_{i,j} \) for all \( i = 1, \ldots, L \) and \( j = 1, \ldots, N \), may all be unique, in which case the rulebase has **local semantics**. That is, fuzzy sets are defined locally to each rule, and are entirely independent from fuzzy sets used in other rules. Alternatively, the same fuzzy sets may be reused between rules, in which case the rulebase has **global semantics**. That is, a set of fuzzy sets is defined to cover an input’s domain, and all rules reference one of those commonly defined fuzzy sets to describe that input.

A rule using global semantics can be rewritten as:

if \( x_1 \) is \( F_{1,i} \) and \( x_2 \) is \( F_{2,j} \) and \( x_N \) is \( F_{N,k} \) then consequent

where \( x_1, \ldots, x_N \) are input variables; \( F_n \) is the set of fuzzy sets covering the \( n^{th} \) input; and \( F_{n,j} \) is the \( i^{th} \) fuzzy set covering the \( n^{th} \) input. In this notation, all rules reference the same sets of fuzzy sets \( (F_n) \), it is only the index \((i, j, \text{and } k, \text{above})\) into the set of fuzzy sets that changes from rule to rule. This global semantics notation is used for the remainder of this work, particularly in chapter 6.

### 2.5.2 Unstructured vs. structured

An **unstructured rulebase** is composed of rules with local semantics, or that do not necessarily have an antecedent for every input variable. A rule that has no antecedent for a particular input variable is a generalised rule that matches all values of that input variable (since the value of that variable will not contribute anything to the rule’s matching degree).

An unstructured rulebase can require fewer rules, because local semantics allows very broad fuzzy sets to be used for certain rules, and unspecified antecedents allows one rule
to match all values of a certain input. However, there is no guarantee of completeness (there is not necessarily at least one rule that will match every possible combination of input values), and rulebase discovery is made more difficult by the non-uniform rules (see section 2.8).

A structured rulebase (also fully specified rulebase) is composed of rules with global semantics, and that have exactly one antecedent per input variable. There is exactly one rule for every possible combination of antecedent fuzzy sets; this is only possible with global semantics, which implies a finite number of unique combinations.

The uniform rule structure in a fully specified rulebase guarantees completeness, and makes automatic rulebase discovery easier.

However, the number of rules in a fully specified rulebase is typically large: \[ \prod_{n=1}^{N} |F_n| \]
where \( F_n \) is the set of fuzzy sets covering the \( n^{th} \) input, and \( |F_n| \) is the cardinality of the set.

The size of the rulebase increases exponentially with the number of inputs. Fortunately, extremely large rulebases can be mitigated by hierarchical rulebase structures (see section 2.7).

### 2.6 Inference engine

The choice of inference engine determines:

#### 2.6.1 Consequent

The form of the consequent is typically either a fuzzy set:

\[ R_i: \text{if } x_1 \text{ is } A_{i,1} \text{ and } x_2 \text{ is } A_{i,2} \ldots \text{ and } x_n \text{ is } A_{i,n} \text{ then } y \text{ is } B_i \, . \]

or a linear function of the input variables:
$R_i$: if $x_1$ is $A_{i,1}$ and $x_2$ is $A_{i,2}$ ... and $x_n$ is $A_{i,n}$ then $y$ is $f_i(x_1, x_2, \ldots, x_n)$.

### 2.6.2 Matching degree

The matching degree, $\alpha_i$, of a rule, $R_i$, can be either the minimum of the input degrees of membership:

$$\alpha_i = \min (\mu_{A_{i,1}}(x_1), \mu_{A_{i,2}}(x_2), \ldots, \mu_{A_{i,n}}(x_n)),$$

or the product of the input degrees of membership:

$$\alpha_i = \mu_{A_{i,1}}(x_1) \cdot \mu_{A_{i,2}}(x_2) \cdot \ldots \cdot \mu_{A_{i,n}}(x_n).$$

### 2.6.3 Interpolation

Fuzzy systems produce smooth approximations of a function, from a finite rulebase of discrete rules, by interpolating the consequents of all matching rules. (This is why a rulebase should have overlapping rules; so that multiple rules match any given input [105].) The interpolation depends upon the form of the consequents.

If the consequents are linear functions, then the functions are evaluated to give crisp consequents. The interpolation is a normalised weighted sum of the crisp consequents:

$$y = \frac{\sum_{i=1}^{n} \alpha_i \cdot f_i(x_1, x_2, \ldots, x_n)}{\sum_{i=1}^{n} \alpha_i} \quad (2.1)$$

If the consequents are fuzzy sets, then ultimately the fuzzy consequents must be defuzzified to give a final crisp output. There are two approaches: addition and fusion.
**Addition – defuzzify then combine**

All consequent fuzzy sets are defuzzified to give crisp consequents. Then, similarly to the linear functions, the interpolation is a normalised weighted sum of the crisp consequents:

$$y = \frac{\sum_{i=1}^{m} \alpha_i \cdot d(B_i)}{\sum_{i=1}^{m} \alpha_i}$$  \hspace{1cm} (2.2)

where $d$ is the defuzzification function used.

**Fusion – combine then defuzzify**

All consequent fuzzy sets are “weighted” by either clipping or scaling the sets by their matching degrees, then the sets are fused by superposition, and finally the resultant fuzzy set is defuzzified. (See [106] for details on clipping/scaling and superposition operations on fuzzy sets.)

### 2.6.4 Defuzzification

There are two common defuzzification functions: mean of maximum and centroid.

The **mean of maximum** of fuzzy set $B$ is the mean all values of $x$ that maximise $\mu_B(x)$.

The **centroid** of fuzzy set $B$ is the centre of the area under the membership function.

The mean of maximum method is faster to compute than the centroid method, but the centroid method is more accurate. However, when additive interpolation is used instead of fusion interpolation, the consequent fuzzy sets can be defuzzified offline, since the crisp values of the singleton consequent fuzzy are constant. For additive interpolation, the computational cost of centroid defuzzification is irrelevant.
2.6.5 Mamdani inference engine
The Mamdani inference engine, by E. H. Mamdani [64], was used for the first fuzzy logic controller, as well as for the first application of fuzzy rules to function approximation. In light of the previous discussion, the Mamdani inference engine can be characterised as:

- Consequents are fuzzy sets.
- Matching degree can be either minimum or product. This inference engine is often called the Mamdani product inference engine, specifying that the product operator is used here.
- Interpolation is by fusion.
- Defuzzification is often mean of maximum, as centroids are expensive to compute for fused fuzzy sets.

2.6.6 Takagi-Sugeno-Kang (TSK) inference engine
The TSK inference engine, by T. Takagi, M. Sugeno (and later K. T. Kang) [100] is capable of giving a more accurate function approximation than the Mamdani inference engine, for the same number of rules, or alternatively reduces the number of rules required for complex high-dimensional problems. It can be characterised as:

- Consequents are linear functions of input variables.
- Matching degree can be either minimum or product.
- Interpolation is additive: a normalised weighted sum of the consequent function evaluations.
- Defuzzification is not required, as the consequents are not fuzzy sets.

The drawback to the TSK inference engine is the lack of interpretability due to the use of linear functions as consequents.
2.6.7 Standard Additive Model (SAM) inference engine

The SAM inference engine, by B. Kosko [58], combines elements of the Mamdani and TSK inference engines. It is characterised as:

- Consequents are fuzzy sets.
- Matching degree is the product operator.
- Interpolation is additive: similar to TSK, a normalised weighted sum of the defuzzified consequent fuzzy sets.
- Defuzzification is centroid, as the consequent fuzzy sets can be defuzzified offline for use in the additive interpolation.

2.7 Hierarchical rulebases

The size of a single, fully specified rulebase covering many input variables with many fuzzy sets can be prohibitively large. For example, a rulebase with 10 inputs each covered by 3 fuzzy sets has $3^{10} = 59049$ rules. The exponential increase in the size of the system as inputs are added is known as the “curse of dimensionality” (which afflicts many areas besides fuzzy systems, including evolutionary algorithms as seen in later chapters).

Very large rulebases are impractical to learn using any method, so alternative rulebase structures that reduce the number of rules are useful.

One way of mitigating the size of a high-input fully specified rulebase is to use a hierarchical fuzzy system [80] in which the inputs are divided amongst multiple smaller rulebases. A single monolithic rulebase (Figure 2.7), covering all inputs, grows exponentially with the number of inputs; a collection of smaller rulebases (Figure 2.8), for each of which the number of inputs is restricted, grows only linearly with the number of inputs.
For example, a hierarchical system can have 2 rulebases each with 5 inputs, which would have \(3^5 + 3^5 = 486\) rules.

![Diagram of single-layer rulebase](Figure 2.7. Single-layer rulebase)

![Diagram of two-layer rulebase](Figure 2.8. Two-layer rulebase)

![Diagram of hierarchical rulebase](Figure 2.9. Hierarchical rulebase)

### 2.7.1 Topologies

The individual smaller rulebases each produce an **intermediate output**. All intermediate outputs must be aggregated in some way to produce a single **final output** from the fuzzy system. The design of the rulebase structures and their links to one another is called the rulebase **topology**.

One simple approach is to have each of the individual rulebases output a “best guess” intermediate output that is an approximation of the final output, and to use a linear combination (e.g., take the mean) of the intermediate outputs to produce a final output.

The aggregation may be more sophisticated than a linear function; it could be another fuzzy rulebase (the linear increase in rules from the addition of another rulebase will usually still be less than the exponential increase avoided by this technique). The additional rulebase adds another **layer** to the topology. Figure 2.8 shows the intermediate output from two first-layer rulebases, A and B, being aggregated by second-layer rulebase C to produce a final output.

Another approach is to feed inputs into the system at different layers, rather than into different rulebases all at the first layer. Figure 2.9 shows this approach. A special case of this approach, in which there is exactly one rulebase per layer, each of which has exactly...
two inputs (one input being the intermediate output from the previous layer and the other input being an input variable) guarantees a linear growth as inputs increase, and the minimum number of rules to cover all inputs. [101]

The former approach described above (Figure 2.8) is often described as a **multilayered fuzzy system**, and the latter approach (Figure 2.9) is often described as a **hierarchical fuzzy system**. However, within both approaches there are a number of different possible topologies, which overlap to some degree. This work uses the terms “multilayered” and “hierarchical” synonymously, and specifies the topology where relevant.

### 2.7.2 Intermediate outputs

When intermediate outputs are aggregated by a higher-layer rulebase, their values are no longer necessarily approximations of the final output. The intermediate outputs can contain any arbitrary information that is used by the second-layer rulebase to produce a final output, potentially improving the accuracy and flexibility of the fuzzy system (compared to a mean of the intermediate outputs). For example, in [67] a multilayer rulebase was used to control a mobile robot. The first layer took just the robot’s position as input and produced a desired direction as intermediate output. The direction was just one piece of information influencing the final output, and was not an approximation of the final control.

Furthermore, intermediate outputs need not even have a physical meaning defined by the practitioner. Learning algorithms (such as evolutionary algorithms, used to learn another robot controller in [97, 113]) produce rulebases that make use of intermediate outputs in any way that is found to work.
2.7.3 Limitations of hierarchical rulebases

A hierarchical rulebase can still be fully specified, in the sense that every possible combination of input values is still covered by rules in the rulebase. The completeness of the hierarchical rulebase is guaranteed if each single rulebase is complete, and a hierarchical fuzzy system is still a universal function approximator [103]. However, it necessarily sacrifices some expressiveness. It is no longer possible for there to be an independent final consequent for every possible combination of input antecedents, since inputs that contribute to an intermediate output can no longer independently affect the final output. The topology design imposes interdependence upon certain groups of input variables. With a suitable topology, this is not necessarily a problem; a good topology should reflect the natural interdependence between inputs such that the imposed interdependence does not restrict the behaviour of the fuzzy system (to an unacceptable degree).

The difficulty in designing a hierarchical or multilayered fuzzy system lies in determining the best topology. For example, [97, 113] studied the effectiveness of different topologies for the inverted pendulum control problem, which is described in chapter 5 and used in chapters 6-8. Chapter 7 uses a hierarchical rulebase to facilitate cooperative coevolution.

2.8 Knowledgebase discovery

This section gives a very brief description of knowledgebase discovery techniques, because chapter 4 examines knowledgebase discovery in the context of evolutionary algorithms, in greater detail.
2.8.1 Extracting human expertise
Fuzzy logic was invented with the goal of bringing human expertise into engineering systems in a systematic, efficient, and interpretable way [101]. To that end, fuzzy systems are still designed by interrogating human experts and attempting to capture their knowledge in the fuzzy system. This is something of an art, but the success of this approach laid the foundations for fuzzy logic applications. For example, the first fuzzy controller for a full-scale industrial process was for a cement kiln, and the knowledgebase was derived from an operator’s manual [47].

2.8.2 Automatic learning
The knowledgebase of a fuzzy system comprises the rulebase and the membership functions definitions. Global semantics allows the clean separation of membership functions from rulebase [63], allowing learning algorithms to target either or both for optimisation.

Learning can broadly be classified as online or offline learning [101].

Online (also called adaptive) learning, in which a controller can adapt its internal input-output mapping, is used for control problems with a highly variable, changing environment. Online learning may also be appropriate when the environment contains unknown variables not included in the original model that the controller was designed around.

Online learning requires a reference model that can identify the desired system state at each point in time, without necessarily knowing the output required to reach that desired state. The online learning algorithm modifies the controller’s knowledgebase in real-time.
to minimise the error between the desired state and the actual attained state. The existence of an initial knowledgebase is usually assumed.

Offline (also called non-adaptive) learning is used for approximation problems in general (including control problems) to produce a knowledgebase, where none already exists, or to tune a knowledgebase, where one exists.

Offline learning requires a system model of the environment to control (i.e., a simulation) or of the function to approximate (i.e., input-output data). This allows the fuzzy system to be evaluated and trained off-line, much faster than real-time and without real-world consequences. Offline learning, by evolutionary algorithms, is the focus of this work.

2.9 Summary

This chapter has given a brief introduction to fuzzy systems as universal function approximators in general, and as controllers in particular. Fuzzy sets and fuzzy logic are used to handle imprecision or uncertainty. Fuzzy rules are input-output mappings, and fuzzy rulebases are sets of overlapping rules that together cover the domain of input space. An inference engine is used to interpolate the output from overlapping rules in order to produce a crisp output in response to a crisp input.

Rulebases can be specified using local or global semantics: in local semantics the fuzzy sets are defined independently for every rule; in global semantics a set of fuzzy sets is defined for each input and reused between rules. Global semantics requires fully specified rulebases, with exactly one rule for every possible combination of input fuzzy sets. Hierarchical rulebases are used to reduce the size of fully specified rulebases by allowing a modular design.
Rules and membership functions can be identified by human experts or by automatic learning. Automatic learning can be online or offline: online learning is used in highly dynamic environments in real-time; offline learning is used with simulations before being implemented. Evolutionary algorithms are well suited to offline learning of fuzzy systems.

In the remainder of this work, various fuzzy systems are used as test problems to investigate evolutionary algorithm techniques.

Chapter 5 describes a fuzzy controller for the inverted pendulum problem, which is used in chapters 6, 7, and 8. Chapter 6 additionally uses fuzzy systems for function approximation.

In all cases, the fuzzy systems used in this work have the following characteristics:

- Triangular fuzzy sets (see section 2.2.1).
- Global semantics (see section 2.5.1).
- Fully specified, structured rulebase (see section 2.5.2).
- Standard Additive Model inference engine (see section 2.6.7)

Additionally, chapter 7 makes use of hierarchical rulebases. In all cases, the problem is to discover the rulebase consequents.
Chapter 3. Introduction to Evolutionary Algorithms

3.1 Search algorithms

Evolutionary algorithms (EAs) are one class of search algorithm. Search algorithms attempt to optimise an objective function:

\[
\text{minimise } f(x) \quad (3.1)
\]

where \( x \in \mathbb{R}^n \); \( x \) is a vector of \( n \) decision variables. \( \mathbb{R}^n \), which is often bounded, is referred to as the search space.

We assume minimisation throughout this work, and for this discussion we assume real-valued decision variables, though the techniques described can (and will, in later chapters) trivially be applied to integer-valued decision variables.

Search algorithms are used when it is not possible to analytically deduce an optimal solution, \( x^* \), from the structure of the function \( f \) – for example when \( f \) is a complex non-linear function. Search algorithms attempt to find \( x^* \), an optimal solution, by iteratively evaluating multiple trial solutions before accepting a solution close to the global optimum. In general search algorithms do not need any knowledge of \( f \), making them problem independent, but they do not guarantee global optimality (since usually they have no way of knowing if a solution is the global optimum or not).
Search algorithms differ in how they select trial solutions. Besides finding an optimal solution, the main goal of search algorithms is to minimise the number of trial solution evaluations in otherwise prohibitively large search spaces.

The function graph of \( f \) is called the fitness landscape. A search algorithm must search the fitness landscape for the global minimum, without becoming stuck in local minima.

There are several well known algorithms to search for an optimal solution to a non-linear optimisation problem \([60]\).

### 3.1.1 Single solution search algorithms

Gradient descent algorithms \([35]\) begin with an initial trial solution, \( x' \), and follow the gradient of steepest descent on the fitness landscape at the point \( x' \) to give the next trial solution. This iterative process continues until the gradient reaches 0. Gradient descent algorithms rely on some method of determining the gradient of the function \( f \) at any given point, and easily become trapped in local minima where the gradient is 0.

Tabu search algorithms \([37]\) move iteratively from one solution \( x \) to the next trial solution \( x' \) in the neighbourhood of \( x, N(x) \). The main feature of tabu search is the tabu list, which modifies \( N(x) \) to exclude certain solutions, such as those recently visited or those that possess undesirable heuristic properties.

Simulated annealing \([61]\) is a search algorithm inspired by the process of annealing metal, in which heated metal is slowly cooled to remove defects. Similarly to tabu search, it iteratively selects the next trial solution, \( x' \), from the local neighbourhood of the current solution, \( x \). The probability of selecting \( x' \) increases with the difference \( f(x) - f(x') \), tempered by a global temperature parameter, \( T \). That is, solutions with better (lower) \( f(x') \) evaluations are always more probable to be selected than solutions with worse (higher)
\( f(x') \), but a higher temperature, \( T \), makes the difference in probability less. At highest temperature, all neighbouring solutions have equal selection probabilities; at lowest temperature, the best solution is certain to be selected. The temperature parameter is slowly decreased over the course of the algorithm.

### 3.1.2 Population-based search algorithms

The previously described search algorithms iteratively evaluate one trial solution at a time. EAs use a population of trial solutions, which are initially scattered randomly across the fitness landscape. A suite of operators, which differ between evolutionary algorithms, are then iteratively applied to the population of solutions, causing the population to “evolve” to better solutions. The wide range of operators available for EAs has made them applicable to many problems.

Particle swarm optimisation [6] is another population-based search algorithm, inspired by the swarming, flocking, and schooling behaviour of animals. Each trial solution has a “velocity” through search space, and is aware of the location of the best solution in the population. At each iteration, each trial solution’s velocity is modified by the direction of the best solution and by some random perturbation, and in the next iteration the solution is replaced by a trial solution along the updated velocity vector. The result is a swarm of particles exploring the search space, biased towards following the best known solution.

### 3.2 Evolutionary algorithms

The defining features of EAs are selection and reproduction, which together support the tenets of Darwinian natural selection: that fitter entities have a greater probability of reproducing, and that offspring resemble their parents. These two properties are all that
are required for any system to evolve [32], and all evolutionary algorithms support them, as a bare minimum:

1. Fitter solutions (better $f(x)$) have a greater probability of reproducing (selection operator).
2. Offspring are similar, but not identical, to their parents (mutation operator).

In addition, most EAs support:

3. Sexual reproduction to exchange information between multiple solutions (crossover operator)

There are many other less common EA operators, some of which are inspired by biological evolution and some of which are not.

Literature in the field of EAs often uses the term individual (or chromosome) to refer to a solution, gene to refer to a decision-variable, and allele to refer to a decision-variable value. In this work we use the generic terms solution and decision-variable, to emphasise that many of the techniques investigated here are generalisable to different search algorithms.

3.2.1 Types of evolutionary algorithms
There are many types of EAs that all use Darwinian natural selection as inspiration but differ by the operators used.

Genetic algorithms were the first EAs used for problem solving. They were characterised by binary encoding, bit-flip mutation, and crossover [38]. Genetic algorithms developed into real-encoded genetic algorithms [65], which are commonly simply called EAs and which are the focus of this thesis. When an algorithm is referred to as an “evolutionary
algorithm” it is usually understood that a real-encoded genetic algorithm is meant, but that the particular operators used are not of importance. Such is the case in this work. Early EAs were largely differentiated by their choice of operators and encoding. For example:

- **Genetic algorithms** use a binary string encoding, where binary bases encode decision variable values analogously to DNA bases encoding alleles.

- **Differential evolution** uses a continuous real-number search-space; operators are designed for the continuous domain and are self-adjusting.

- **Genetic programming** encodes programs as tree structures, with nodes representing atoms (constants and variables) and operations and functions. Nodes can be mutated or entire branches crossed over.

- **Evolution strategies** are stripped-down EAs driven by simple selection operators.

The low-level operators and encoding schemes that differentiate these traditional EAs are largely interchangeable, and recently have taken a back seat to research in higher-level architectural differences between EAs, such as co-evolution between multiple populations, constrained optimisation, and multiobjective optimisation.

### 3.2.2 A simple evolutionary algorithm

The following algorithm describes a simple EA using generational replacement. Any reproduction selection, crossover, and mutation operators may be used.
1. generate \( n \) solutions with random decision variable values, to become population \( P(0) \)

2. evaluate fitness of all solutions in population \( P(0) \)

3. repeat for \( t = 1 \) to \( g \) generations:
   i. repeat to generate \( n \) children:
      a) select two parents, \( p_1 \) and \( p_2 \) from population \( P(t - 1) \) using selection for reproduction
      b) copy \( p_1 \) and \( p_2 \) to give two children, \( c_1 \) and \( c_2 \)
      c) perform mutation on \( c_1 \) and \( c_2 \)
      d) perform crossover on \( (c_1, c_2) \)
      e) add \( c_1 \) and \( c_2 \) to \( P(t) \)
   ii. evaluate fitness of all solutions in population \( P(t) \)

4. return the best solution from \( P(g) \)

### 3.2.3 Encoding

Solutions must be encoded in such a way that they can be manipulated by the EA. The encoding provides the “genetic material” that EAs manipulate to evolve better solutions.

Initially, genetic algorithms manipulated bit strings that encoded the solution’s decision variables as a long string of 1’s and 0’s (analogously to the a-t-g-c encoding of DNA). However, it is now more common for EAs to use operators designed to directly manipulate a vector of decision variables.

The encoding of the solution is most relevant to the generation of new solutions during reproduction (ie, usually crossover and mutation). It is usually not relevant to the selection process.
Common encoding schemes are, for example:

- **Bit string** – One long bit string encodes the decision variables as a sequence. The advantage is that the rest of the algorithm can be entirely agnostic to the actual decision variables, but bit string encoding is no longer commonly used due to representational accuracy and the destructiveness of crossover and mutation on arbitrary bits in the string.

- **Real encoded** – A vector of real-valued decision variables. This also covers the specific case of integer-valued decision variables. The main concern is the position of each decision variable in the vector, as position is often relevant to crossover. Real encodings typically specify upper and lower bounds for each decision variable.

- **Tree encoded** – Used for genetic programming. A tree represents a program by using leaf nodes to represent atoms (constants and variables) and branch nodes to represent operations and functions.

Chapter 6 investigates the use of a multidimensional encoding scheme designed to reduce bias in crossover.

### 3.2.4 Fitness

The fitness function used to evaluate trial solutions is *usually* the function $f$ to be optimised, so that evolution towards better fitness values corresponds to optimising the objective function. Exceptions to this rule occur when some additional terms are added onto the objective function to form the fitness function. For example, a penalty term may be added to reflect constraints that must be satisfied (see section 3.6), or a diversity term may be added to promote population diversity.
In general though, the most important term in the fitness function is the objective function, and so the two names are often used interchangeably, and in this work unless explicitly stated otherwise, $f = \text{objective function} = \text{fitness function}$.

Note that in our discussions of EAs, we refer to “better” and “worse” fitness scores to avoid confusion over minimisation and maximisation problems.

Fitness evaluation refers to the act of computing a solution’s fitness score from the fitness function. This computation may be expensive, depending on the problem, and given the typically high number of trials, and therefore evaluations, required by EAs, the goal of many EAs is to minimise the number of fitness evaluations required (whilst still producing high quality solutions!).

A solution’s fitness determines its probability for selection. The particular interaction between the way the selection operators work and the way a solution’s fitness is evaluated determines the selection pressure on the population that guides evolution.

### 3.2.5 Selection for reproduction

Selection for reproduction supports the Darwinian tenet that fitter individuals have greater probability of reproducing. Usually two solutions are selected from the previous generation to become parents, producing two children that are added to the current generation (note that newly generated children, in the current generation, are usually not candidates for selection as parents, from the previous generation).

Selection for reproduction is repeated to produce the desired number of children per generation. This may be some fraction of the total population size, up to and including generating a number of children equal to the population size.
The actual selection operator selects one solution from the population, and is used once independently for each parent. The probability of selecting a solution should correlate with its fitness. The selection operator must balance exploitation with exploration: it favours the best known solutions, but gives all solutions a chance to contribute to the search. For this reason, selection for reproduction is almost always stochastic; not entirely deterministic and not entirely random.

**Roulette selection** [38], also known as proportional selection, allocates a probability, \( p \), directly proportional to a solution’s fitness relative to all other solutions in the population, \( P \), (assuming maximisation of \( f(x) \)): \[
p(x) = \frac{f(x)}{\sum_{y \in P} f(y)}
\]

**Tournament selection** returns the solution with the highest fitness from a random sample of solutions from the population. The most common is 2-tournament, where the random sample is 2 solutions. Selection pressure can be increased by increasing the sample size; the larger the sample size, the better the probable fitness of the best solution in the sample.

### 3.2.6 Reproduction

Once two solutions are selected as parents, the reproduction scheme is used to generate two children. Reproduction usually involves two operators: crossover and mutation. The goal of crossover is to shuffle large chunks of existing genetic material through the population. The goal of mutation is to introduce new genetic material into the population. Both operators are designed to suit the encoding scheme.
3.2.6.1 Crossover
Crossover is the essence of sexual reproduction; it allows large amounts of genetic material to be exchanged between two parents. This is what makes the population-based approach of EAs so effective; the population is not a collection of trial solutions in isolation, but a pool of genetic material that is reshuffled into new solutions in continually better combinations.

There are many crossover operators, which are usually designed to operate on two input solutions, for example:

- **n-point** crossover – designed for string or vector encodings. The two solution strings (or vectors) are cut at \( n \) randomly selected positions (the same positions for both strings), producing \( n+1 \) segments per solution. Then, every second segment is exchanged between the two strings. For example, in 1-point crossover each string is bisected at the same point, and the second segments are exchanged.

- **Uniform** crossover – suitable for most encodings. Every decision variable (or bit in a bit string) has an independent probability (usually 0.5) of being exchanged between the two solutions.

- **Arithmetic** crossover – designed for real encodings. For each decision variable, the real values of that variable for both solutions are combined in a weighted sum to give the value of that variable for a single child (arithmetic crossover produces one child from two parents). The weight on each solution can be random for each operation, set as a parameter, or influenced by fitness.

The effect of crossover often depends on the encoding scheme used. For example, with n-point crossover the closer together two decision variables are in the vector, the greater the probability that they will remain together after crossover. This leads to the rule of thumb
that closely related decision variables should be close together in the decision variable vector, since it is more often more detrimental than beneficial to change variables that depend closely upon one another. (see section 3.2.10)

Many EAs employ a crossover probability, applicable to all crossover operators, which is simply the probability of the crossover operator being applied at all, for each reproduction.

\[
\begin{array}{cccccccc}
A & B & C & D & E & F & G & H \\
I & J & K & L \\
\end{array}
\]

to child 1

\[
\begin{array}{cccccccc}
A & B & C & D & E & F & G & H \\
I & J & K & L \\
\end{array}
\]

to child 2

\[
\begin{array}{cccccccc}
A & B & C & D & E & F & G & H \\
I & J & K & L \\
\end{array}
\]

to child 1

**3.2.6.2 Mutation**

Crossover only reshuffles existing genetic material, and over time genetic material will be lost as low fitness solutions are passed over for selection. Mutation counters this loss of genetic diversity by continually introducing new genetic material (decision variable values) into the population, ensuring that the population cannot become inescapably stuck in local optima.

There are many mutation operators, which are usually designed to operate on each decision variable (or bit) independently, for example:

- **Bit inversion** – designed for binary string encodings. Each bit in the string has an independent probability of being flipped from a 1 to a 0 or vice versa. In a binary encoding, this has a high probability for dramatic changes to the magnitude of the variable if a high-order bit is mutated.

- **Gaussian mutation** – designed for real encodings. The probability of the decision variable mutating to any given value is given by a Gaussian distribution centred
on the current value; smaller mutations are more probable than larger mutations. The standard deviation of the Gaussian distribution is given as a parameter.

Mutations that exceed the decision variable’s bounds are set to the bound. The mutation probability is the independent probability of each decision variable being mutated.

3.2.7 Selection for survival
EAs typically have a fixed population size. Reproduction will exceed the population size, and a limited number of solutions are selected to survive to the next generation. There are many ways of selecting the next generation from the previous generation + children:

- **Generational replacement** – a number of children equal to the population size are produced, and these children replace the previous generation.

- **Greedy replacement** – the best-fitness solutions from the previous generation and the children, combined, survive to the next generation.

- **Stochastic replacement** – a stochastic selection operator, such as roulette or tournament, is used to select the next generation from the previous generation and the children, combined.

In addition, **elitism** may be used in conjunction with any of the above: A given number of the best-fitness solutions from the previous generation are guaranteed to survive to the next generation. The remainder of the population is selected some other way.

3.2.8 Diversity
Some measure of **diversity** may also be incorporated in an attempt to maintain useful genetic material in the population throughout the search. This is intended to avoid local optima, and to find all global optima in multi-modal problems.
Diversity (also called **niching**) is a measure of how similar a solution is to the rest of the population, and usually modifies the fitness score to influence selection indirectly [39]. Diversity may also influence selection directly when implemented in **restricted mating** schemes [30] (in which genetically similar parents are selected), or when implemented in **crowding** [24] (in which children replace genetically similar solutions from the previous generation).

In traditional (single-objective) EAs, diversity is measured in decision-variable space, and is referred to as **genetic diversity**. In multiobjective EAs, diversity is usually measured in objective space (see section 3.5.2.4).

### 3.2.9 Convergence

The search behaviour of an EA is a balance between exploration of new solutions and exploitation of good solutions. Convergence is the process of the population evolving from initially random solutions to just a few similar good solutions. Optimal convergence is a vague concept. Convergence should not be too fast, lest the population converge to sub-optimal solutions, but convergence needs to be fast enough to find good solutions within a reasonable timeframe.

The choice of operators and parameters affects the convergence behaviour of the EA. Some established rules of thumb are:

- Higher tournament sizes increase selection pressure, speeding convergence.
- Higher mutation rates and mutation magnitudes increase population diversity, slowing convergence.
- Greedy replacement and elitism increase selection pressure, speeding convergence.
Diversity measures maintain genetic diversity in the population, slowing convergence.

Complicating the issue is the observation that convergence is not strictly necessary to find near optimal solutions; the optimal solution need only be found once to be useful at the end of the algorithm’s run, it needn’t fill the entire population. This is especially relevant when elitism is used to preserve the best solution from generation to generation; in this case elitism, which by itself speeds convergence, enables the use of other techniques to increase genetic diversity and slow convergence.

3.2.10 Epistasis

In biology, epistasis is the effect of one gene upon the phenotypical expression of another. Analogously, in evolutionary computation, epistasis is the effect that one decision-variable has upon the contribution of another decision-variable to the fitness evaluation [38]. Intuitively, it is a coupling between two (or more) decision-variables that is significant to the performance of the solution.

Changing just one of a pair of highly epistatic decision-variables will have a dramatic effect on the contribution of the unchanged decision-variable to the fitness evaluation, making it difficult to optimise epistatic decision-variables independently of one another. Also recall that evolution fundamentally depends upon offspring being similar to their parents; if a small change results in a very large difference between parent and offspring, steady evolution can be difficult. This suggests that in general, pairs (or groups) of mutually epistatic decision-variables should be preserved, or changed, together.

This exploitation of epistasis is a valuable heuristic implemented in many EAs. In its simplest form, it is the rule of thumb that closely related decision variables should be
located close to one another in the solution encoding so that crossover is more likely to keep them together. Chapter 6 investigates epistasis in the context of crossover, and chapter 7 investigates epistasis in the context of cooperative coevolution.

3.2.11 EA design

![Figure 3.2. EA design framework.](image)

The problem domain is given to the EA practitioner. The operators are usually selected from the suite of common operators in the literature. It is the task of the practitioner to design an encoding and fitness evaluation that will interface between the problem domain and the established EA operators. This is the *application* of an EA to a problem.

The practitioner may find that no satisfactory application can be devised to interface between the conceptual solution given and the operators existing in the literature. In such a case the practitioner may be inspired to develop a new operator to suit the application, rather than the other way around. Well-known examples of this are genetic-programming mutation and crossover operators designed to suit tree-structured encodings (see section 3.2.1), and Pareto-based selection operators designed to suit multiobjective fitness evaluations (see section 3.5).
This work modifies the crossover operator in chapter 6, to support multidimensional encodings; the fitness evaluation and selection in chapter 7, to support cooperative coevolution; and the fitness evaluation and selection in chapter 9, to support constrained multiobjective optimisation.

### 3.3 Parallelism

Parallel processing is commonly used for two reasons: to reduce the real-world computation time required by a problem, and to solve problems that require more data than a single computer can keep in main memory. (see [33])

Although it may conceivably be the case that an EA is used to solve a problem with memory requirements exceeding a single computer, it is more commonly the case that an EA searching a very large and complex search-space, or with a very computationally expensive fitness function, may simply not be able to produce an acceptable solution in an acceptable time-frame. In such a case, parallel processing can be used to reduce the real-world computation time, and this work only considers this use of parallel processing.

Most algorithms are expressed sequentially, and must be reworked to support parallel processing. Those computations that can be performed without (or with minimal) dependence upon other computations are identified and implemented in parallel. Different algorithms will be more or less amenable to parallelisation. In general, Ahmdal’s Law imposes a limit on the speedup possible from parallelisation. It describes the execution time of a program in terms of sequential code, parallel code, and communications overhead:

\[
t = s + p/n + on
\]

(3.2)
where $t$ is the total execution time, $s$ is the sequential code execution time, $p$ is the parallel code execution time, $o$ is the communications overhead time, and $n$ is the number of parallel processors. $s$ is constant, placing a lower limit on the total execution time. $p$ decreases towards zero with additional processors, but $o$ increases (here linearly, but sometimes exponentially) with additional processors. There is therefore a different optimal parallel architecture for different problems, and often the solution is not to simply add more processors, as this can increase the total execution time!

Generally, the ratio of parallel computation, $p$, to communication overhead, $o$, in an algorithm is described as its granularity. The parallelised algorithm should be designed and implemented to maximise the amount of parallel computation whilst minimising the amount of communication – thereby maximising the granularity. The greater the granularity, the easier it is to find a parallel processor architecture that will result in a worthwhile speedup.

Different problems exhibit differing degrees of inherent granularity. For example, astrophysics simulations with n-body interactions are inherently fine-grained (every object influences every other object in the simulation), but evolutionary algorithms are inherently coarse-grained (no object – i.e., an individual fitness evaluation – is influenced by any other object).

### 3.3.1 Parallel supercomputers

The following definitions are used:

**Process** – a single executing program. A process can have multiple threads executing (conceptually) in parallel, which all share the process’ address space. A parallelised
algorithm may be implemented as multiple processes, or multiple threads within a process.

**Processor** – a single processor, capable of a single thread of execution at a time.

**Node** – the smallest unit of hardware that can be allocated to a single process. A node may have multiple processors, in which case a process must be multi-threaded to take advantage of the node’s full capacity.

Supercomputing facilities can be broadly classified by how tightly coupled their processing nodes are. Generally speaking, tightly coupled architectures, with high-speed and low-latency communication between nodes, are used for fine-grained problems with high communication needs. Tightly coupled architectures are also generally the most expensive so, where possible, coarse-grained problems are run on cheaper, loosely-coupled architectures.

For more detail, see [23]. The more common architectures are:

**Centralised multiprocessor**

A single computer with multiple, tightly coupled nodes. Each node may have its own memory (nonuniform memory access, NUMA) or all nodes may share the same pool of memory (uniform memory access, UMA, also called symmetric multiprocessors, SMP). Regardless of whether the architecture is NUMA or UMA, all nodes have access to the same physical address space, and different processes can share data simply by reading and writing to memory. Centralised multiprocessors therefore use a **shared-memory** architecture.
**Cluster**

A collection of individual, loosely coupled nodes, each of which has its own processor(s) and memory, connected by a communications network. Each node is typically a stand-alone computer, but nodes are typically **homogenous** – all use the same hardware. Clusters may be proprietary, coming pre-assembled in racks, or may be custom made from off-the-shelf computers.

Although the network is typically very fast relative to other communications networks, inter-node communication cannot approach the high speeds and low latency of a tightly coupled multiprocessor computer, in which nodes communicate over a local system bus. This relatively slow inter-node communication makes a shared-memory architecture infeasible; each node can only address its own local memory, and processes must share data by explicitly sending and receiving **messages** over the communications network. This work assumes parallelism by message-passing, which can also be used on shared-memory architectures.

**Computing grid**

A very loosely coupled collection of nodes. Computing grids typically differ from clusters in the following ways:

- Nodes are **heterogeneous**, with different hardware and operating systems.
- The communications network is large-scale, decentralised, may be heterogeneous, and may be slow even relative to other communications networks.
- Different parts of the grid may be owned and controlled by different organisations or individuals, necessitating complex access control.
3.3.2 EAs and parallelism
Evolutionary algorithms are well known for their “trivial” parallelism (see for example the reviews [1, 2, 8]). The trivial parallelism is due to the important role fitness evaluations play in EAs: fitness evaluations are often the most computationally expensive part of an EA, and each fitness evaluation can be calculated independently of every other fitness evaluation, making it trivial to distribute a population across multiple processors to process this stage in parallel.

The simplest way of parallelising an EA is the master-slave model (see [9]). One process, the master, is responsible for the evolutionary logic – selection, mutation, crossover, etc. – and each generation it distributes the population out to slave processes to perform the fitness evaluations. This is trivially parallel, because each fitness evaluation is independent from all others, and the only communication required is sending the solutions out to the slave processes at the beginning of each generation, and receiving the fitness scores back at the end of each generation.

Typically the master-slave model is synchronous, where all solutions must be evaluated every generation. In this case the essential algorithm is unchanged: the synchronous model is implemented without affecting the output of the algorithm. However, the implementation may be inefficient: when each slave process finishes evaluating all its solutions, it must block until the slowest slave process finishes and the next generation can be distributed out to the slave processes. The implementation may be complicated by heterogeneous processor nodes and fitness evaluations with variable computational cost, both of which make efficient distribution of solutions to slave processes unpredictable.
Rarely, the master-slave model is **asynchronous**, where not all solutions must be evaluated before the next generation begins. Therefore not all solutions are candidates for selection and reproduction. This is similar to EAs that use a generational gap rather than full replacement, but because the “generation gap” is caused by nondeterministic factors such as processor speed and network, and by nonlinear factors such as variable fitness evaluation costs, the parallelism of the implementation affects the results from the asynchronous model. The advantage is that the asynchronous model can avoid excessive blocking. However, the improved efficiency may not be worth the uncertainty of the results, and as the asynchronous model is not commonly used, for the rest of this work master-slave refers to the synchronous master-slave model.

Another approach is to structure the population. The **cellular** model arranges solutions spatially on a grid (commonly 2-dimensional). The grid structure introduces locality to solutions, such that mate selection is restricted to the immediate neighbourhood of grid positions. The algorithm is parallelised by assigning different regions of the grid to different processes. As mate selection is local, this minimises inter-process communication. Furthermore, the dimensionality of the grid and the neighbourhood radius can be tailored to match the parallel computer architecture – this is most relevant to NUMA multiprocessor architectures, where communication costs are non-uniform and depend on the distance between physical processors.

The **island** model divides the population into subpopulations (islands), which evolve mostly independently [16]. Mate selection is restricted to the same island, but occasionally a solution “migrates” from one island to another. The migration policy
determines which solutions are selected for migration (e.g. selecting the best solution aids convergence; a random solution aids diversity), and to which islands they can migrate. The algorithm is parallelised by assigning different islands to different processes. The migration policy minimises inter-process communication, similarly to the restricted neighbourhood in the cellular model.

The cellular model is usually described as fine-grained and the island model as coarse-grained, though of course the neighbourhood radius and migration policy determine the actual granularity of each.

Any of the models can be combined to form a hierarchical model. For example, the island model can use subpopulations that arrange their solutions in a cellular grid, or the cellular model can use the master-slave model for parallelism. Different combinations suit different aims: for example, finer granularity of parallelism, or smaller niching. See [8, 41].

Both the cellular model and the island model change the essential algorithm, as does the asynchronous master-slave model. However, these changes to the algorithm may be claimed to be improvements to the algorithm, and the structured populations are sometimes implemented without parallelising the algorithm. The structured populations can work similarly to niching in traditional EAs, controlling premature convergence by maintaining genetic diversity through restricted mate selection.

When investigating sequential algorithms, the implementation code and computing hardware used to obtain the results can be abstracted away; they are irrelevant to the
results. However, when using any model of parallelism that affects the outcome of the algorithm, results must be interpreted with caution because the algorithm’s implementation and the parallel computing architecture used can affect the results by changing the sequence of execution.

Chapter 7 discusses the parallelism of EAs in greater detail, in the context of coevolutionary algorithms (which are particularly suited to parallel implementations).

3.4 Cooperative coevolution

Coevolution, broadly, is the use of multiple populations evolving largely independently to solve a problem. Cooperative coevolution is contrasted with competitive coevolution [44, 83] and island coevolution [16]. Competitive coevolution is the use of two populations each evolving independent solutions that are in competition with one another – for example, opposing game-playing strategies or solutions vs. test-cases – such that an “evolutionary arms race” is encouraged to progressively drive the evolution of both populations to greater sophistication. Island coevolution is the use of multiple populations that rarely exchange genetic material, with the aim of protecting niches and preserving genetic diversity. This work does not use competitive coevolution or island coevolution.

Cooperative coevolution [75] is a framework for dividing the total number of decision variables amongst multiple, (largely) independent EAs. The task of each EA is to optimise only its own assigned decision variables. Each EA operates on its own population of individuals, each of which encodes only the decision-variables assigned to that EA.
In cooperative coevolution, an individual is only a partial solution. Exactly one individual from each population is required to form a complete solution. These “representative” individuals are selected and combined by a representative manager.

Figure 3.3 illustrates the cooperative coevolutionary framework. The number of populations and the decision variables assigned to each population are arbitrary. There is no requirement that decision variables be assigned in contiguous blocks; in fact the encoding of the solution is unimportant. However, there is a relationship between epistasis, solution encoding, and coevolutionary design, which is discussed below and further in chapters 6 and 7.

3.4.1 Curse of dimensionality
EAs suffer from the “curse of dimensionality”: as the number of decision variables (dimensions) to be optimised increases, the search space grows exponentially. This exponential growth can be mitigated by decomposing the entire problem domain into smaller problems. For example, instead of a single 10-dimensional search space, it may be possible to search two 5-dimensional search spaces. This gives a decrease in the total
search space, as the number of potential solutions in each search space are summed rather than multiplied.

### 3.4.2 Modularity

EAs are applied to increasingly more complex problem domains, in which modularity is desirable. It may also be the case that a more modular search process would aid evolution. The practitioner can of course design solution modularity into the encoding, but this does not allow the EA to exploit the modularity during the search process (except indirectly through the crossover operator, as investigated in chapter 6).

Cooperative coevolution [75] explicitly introduces modularity into the EA search process. Each population evolves just one module of problem. This allows the evolutionary process to optimise each module independently of the overall, possibly quite complex, problem.

There are two difficulties. The first is the **apportionment of credit** problem: how can the fitness of a single module be evaluated, independently of other modules upon which it depends? Cooperative coevolution provides a framework for the recombination of modules into a whole solution, in such a way that the quality of an individual module can be assessed.

The second difficulty is the **decomposition** of the problem domain into relatively independent modules. In some problem domains the most appropriate decomposition will be obvious. In other problem domains the decomposition will be very ambiguous. Due to the diversity of problem domains, there is as yet no standard method of decomposition
and practitioners must find or design one suited to their particular problem domain. Chapter 7 proposes a problem decomposition suitable for some fuzzy systems.

3.4.3 Apportionment of credit

It is usually impossible to evaluate the fitness of an individual (a partial solution) on its own. (If it is possible to do so, then a standard EA can be used and cooperative coevolution is not required – this work assumes it is impossible.) When an individual is evaluated, it must be part of a complete solution. The difficulty is determining how much the individual contributes to the overall solution quality – how much credit should be apportioned to the individual.

To evaluate the fitness of individuals in population $n$, the EA retrieves a representative individual from each of the other populations. All individuals in population $n$ are then combined with the same representatives to form complete solutions, such that the solutions differ only by the decision-variables assigned to population $n$. These solutions are evaluated, and any differences in fitness may be attributed to the individual and not the representatives (which are the same for all individuals). The fitness of the individual is simply assigned to be the fitness of the solution it formed.

This arrangement is not perfect. It may be the case that an individual, $a$, performs better than another individual, $b$, when combined with one set of representatives, but that $a$ performs worse than $b$ when combined with a different set of representatives. This can happen when there is a strong epistatic relationship between decision-variables in the individual and decision-variables in the representatives. For this reason, the selection of representatives has an effect on the search process.
[76] describes two methods for selecting representatives, which are investigated in chapter 7:

- **Exploitative**: Select the individual with the best fitness in its population to be the representative of that population. As the final solution returned from the cooperative coevolutionary algorithm will be composed of the best individuals from each population, this selection method guides the search towards individuals that perform well in this context, thereby directly optimising the final solution.

- **Explorative**: Select a random individual from each population to be the representative of that population. This favours modularity, since individuals combined with random representatives must be robust to consistently attain a good fitness, and increased modularity may indirectly improve the final solution.

Both of these selection methods can be used together (take the better of the two fitness scores as the individual’s fitness), thereby obtaining the benefits of both at the cost of increased computational cost.

As stochastic search algorithms, EAs are relatively tolerant of factors that may occasionally cause an inferior individual to be selected over a superior individual, as long as the selection pressure favours superior individuals in general. Therefore the apportionment of credit does not need to be especially precise, as long as it stochastically – and unbiasedly – identifies superior solutions. Careful selection of representative individuals can fulfil this requirement.

### 3.4.4 Decomposition

Cooperative coevolution requires the original problem to be decomposed in some way such that the individuals of a given population encode only a subset of all decision-
variables. There are many ways of decomposing the original problem, and most are domain-specific.

The above discussion showed that high epistasis between decision-variables in different modules makes the apportionment of credit problem difficult. Therefore, the decomposition should be designed to minimise the epistatic links between modules.

Cooperative coevolution was first used in [76] to optimise function variables, using an extreme decomposition of one decision-variable per population. Decision-variables were encoded as bit-strings, so crossover could be used even though each individual encoded just one decision-variable.

Cooperative coevolution was used in [78] to learn rules for a rule-based robot controller. Each individual encoded an entire rulebase, but the rulebase was unstructured such that multiple individuals (i.e., representatives) could be concatenated to form a larger rulebase. Therefore it was trivial to decompose the system into arbitrarily large sets of rules that could be merged together to form a complete system. In this particular case, the system was decomposed into 2 populations of rules and each population was seeded with expert knowledge covering different areas of behaviour.

Cooperative coevolution was applied to inventory control in [34], comparing an extreme decomposition (as in [76]) to an intuitive decomposition that minimised epistasis between modules. The intuitive decomposition grouped together into the same module, decision-variables that related to the same inventory item. The intuitive decomposition proved better than the extreme decomposition.
A dynamic decomposition was investigated in [77], which started with one module and new modules – new populations – were created as required. To do this, the system used Cascade Neural Networks, which allow layers of neurons to be added one at a time without losing the knowledge already in the network. Each population was assigned one layer to be encoded by its individuals; to add a layer to the neural network a new population was created for that layer. A new layer was added after a number of generations had elapsed or evolution had stagnated.

In contrast to starting with one module and dynamically adding modules as necessary, [104] started with an extreme decomposition (as in [76]) and analysed the current decomposition for signs of high epistasis between modules. If a high epistatic link was identified between two modules, those modules were merged into one (the two populations were merged).

### 3.5 Multiobjective evolutionary algorithms

Traditional, single-objective EAs use one objective function to measure the quality of a solution and to differentiate it from other solutions in the population. In real-world problems there are usually multiple objectives to optimise, which may conflict with one another – for example, one conflict might be the quality of a solution vs. the cost of its implementation (and even “quality” itself might be characterised by multiple, possibly conflicting, objectives). Problems with conflicting objectives are characterised by “trade-offs”: when searching for an “optimal” solution, a search algorithm might trade off performance on one objective for better performance on another.

The general multiobjective optimisation problem becomes:

$$\text{minimise } f_i(x), i = 1, \ldots, M$$

(3.3)
The difficulty with multiple objectives is that an EA requires a single fitness score, and it is not immediately obvious how a single fitness function can be obtained from multiple objective functions.

The simplest way of evaluating multiple objectives is aggregate fitness evaluation, whereby the fitness scores for all individual objectives are summed, usually with a constant weight associated with each term to reflect its relative importance. This is common practice in evolutionary computation.

The problem with this approach is two-fold: first, it is often unclear what the constant weights should be, especially if the objective functions are non-commensurable, and second, it does not allow for trade-off between conflicting objectives, as the weights lock the search to a certain region of the trade-off surface.

Multiobjective EAs optimise multiple objective functions at the same time. Coello [14] gives a summary of multiobjective EA research.

### 3.5.1 Approaches to multiobjective optimisation

There are two broad approaches to multiobjective optimisation, as identified in [27]. The first approach, called the **ideal** approach, is to generate a large set of candidate solutions – the final solution set – all with unique advantages (and disadvantages). The final decision-maker must then subjectively select one solution from the set, using higher-level qualitative considerations that were impossible or inappropriate to include in the original problem description. The ideal approach is taken in this thesis.

An alternative approach, called the **preference-based** approach, is to identify beforehand the relative preference of each objective in the multiobjective problem. The weighted-
sum method of evaluating multiple objectives, mentioned above, is an example of preference-based multiobjective optimisation.

### 3.5.2 Pareto methods

Most methods taking the ideal approach to multiobjective optimisation are Pareto based. Vilfredo Pareto presented the concept of a Pareto-optimal solution in the context of economics in 1896 [72]. Given resources to be allocated to individuals, a Pareto-optimal allocation is one in which the allocation cannot be changed to make any individual better-off without simultaneously making another individual worse-off. This concept is generalised to multiobjective problems. A Pareto-optimal solution is a solution such that the decision-variables cannot be changed to improve any objective without simultaneously worsening another objective.

Pareto-based methods of multiobjective optimisation are based on the assumption that the scores on different objective functions are non-commensurable in the broadest sense: that a score on one objective function cannot be compared, in any way, to a score on any other objective function.

Non-commensurable usually refers to different unit types. For example, cost in $ cannot be compared to acceleration in m/s^2. Non-commensurable in the Pareto sense goes beyond unit type, however. If the same unit is used to measure different qualities, it is still non-commensurable. For example, vehicle acceleration in m/s^2 cannot be compared to braking in m/s^2.

Pareto optimisation assumes that all objectives measure non-commensurable qualities. Therefore, a solution can only be better than another solution if it is better on all objectives, in which case it is said to dominate the other. If neither solution dominates
the other, they are **non-dominated**. These boolean, binary relationships are the only ways to compare solution quality in Pareto-based multiobjective optimisation.

### 3.5.2.1 Objective space and Pareto front

The use of multiple objective scores creates another search space in addition to decision variable space: **objective space**. A solution’s objective scores can be plotted in this objective space.

In a minimisation problem (assuming a minimum of 0), the “closer” (taking into account non-commensurability) a solution is to the origin of objective space, where all objectives are 0, the better it is. However, almost all multiobjective problems are characterised by conflicting objectives, where improvement in one objective is impossible without worsening in another, making it impossible to even *formulate* (let alone implement) a solution that lies at the origin of objective-space.

These interrelated, conflicting objective functions define the **Pareto front** in objective space. The conflicting objective functions make it impossible to formulate any solution that has better objective scores, for all objectives, than any point on the Pareto front. The Pareto front is a surface in objective space, separating the unattainable origin from the searchable region of space.

It is the goal of all Pareto-based multiobjective EAs to find a set of solutions that approximate the Pareto front. A good approximation set will contain solutions that are close to the Pareto front and that are distributed evenly across its surface.
3.5.2.2 Pareto dominance
EAs traditionally used the single fitness score to compare solutions. As there is no single fitness score in Pareto-based multiobjective EAs, another method of comparing solutions is necessary.

Pareto dominance is a means of comparing the quality of two solutions in objective space. One solution, \( a \), dominates another solution, \( b \), written as \( a > b \):

\[
a > b \quad \text{iff} \quad \left( \forall i \in \{1, \ldots, M\}, f_i(a) \leq f_i(b) \right) \land \left( \exists i \in \{1, \ldots, M\} : f_i(a) < f_i(b) \right)
\]  

That is, \( a \) is at least as good as \( b \) on all objectives, and is better on at least one. If neither \( a > b \) nor \( b > a \) then \( a \sim b \), that is \( a \) and \( b \) are non-dominated.

![Figure 3.4. Visualisation of domination. Points a and b are non-dominated with respect to one another. Point c is dominated by b but not a.](image)

Pareto dominance can be used directly in tournament selection, which removes the need for a single fitness score as only a binary comparison is used, or can be used to organise the population into dominance ranks, which provides a single fitness score.

3.5.2.3 Dominance ranking
Pareto dominance ranking [38] is one way of assigning a single fitness score to a solution in a multiobjective problem. It is an iterative process: those solutions in the population that are non-dominated are assigned to rank 1. Rank 1 solutions are then removed from consideration, and those solutions that previously were dominated only by rank 1 solutions become the new non-dominated solutions and are assigned to rank 2.
Rank 2 solutions are then removed, and the process continues until all solutions in the population are ranked. A solution’s fitness is simply its rank.

Not all Pareto-based multiobjective EAs use dominance ranking (though this work does). For example:

In PESA-II [21] dominance ranking is not required because only globally non-dominated solutions (i.e., only the first rank in dominance ranking) are candidates for selection.

In SPEA2 [115] a solution has a strength, which is the number of solutions in the population that it dominates. A solution’s fitness is the sum of the strengths of all solutions that dominate it. Therefore all non-dominated solutions have the best fitness, 0. SPEA2’s fitness is interesting in that it reflects both proximity to the Pareto front (the closer a solution is to the front, the fewer solutions will dominate it, and the lower the sum of strengths will be) and diversity (the more isolated a solution is, the lower the strength of solutions that dominate it, and the lower the sum of strengths will be).

3.5.2.4 Diversity
Dominance ranking provides the selection pressure to move through objective space towards the Pareto front, but provides no pressure to spread out along the Pareto front. It is combined with some measure of diversity so that the population clearly and smoothly identifies the Pareto front, giving the final decision maker the most diverse set of solutions possible.

Diversity in multiobjective optimisation serves a different purpose to diversity in traditional single-objective EAs. In a traditional EA, diversity (also called niching, crowding, or fitness sharing) is intended to maintain a high decision-variable-space
diversity in the population, to avoid local optima and find all global optima in multi-modal problems. In a multiobjective EA, diversity is intended to maintain a high objective-space diversity, to provide the most diverse solution set to the final decision-maker.

Diversity is usually considered to be secondary in importance to dominance. The rationale is that, given two solution sets $A$ and $B$ such that every solution in $B$ is dominated by at least one solution in $A$ there will always be a superior solution choice in $A$ regardless of either set’s diversity. Therefore, the use of diversity will usually never cause a dominated solution to be more fit than a solution that dominates it. Either the diversity measure modifies the fitness by an amount less than the difference caused by domination, or the EA’s selection operators are modified to compare first on dominance and second on diversity. Assuming a tournament selection operator, the end effect is the same.

NSGA-II [28] measures the distance to a solution’s neighbours over all objective dimensions. Each objective dimension is normalised. For each dimension, the distance between a solution’s two neighbours (one on either side) is added to that solution’s diversity measure. Solutions at the extremes of each dimension are assigned an arbitrary diversity. This method is only accurate for ranks of mutually non-dominated solutions, since it assumes that solutions lie on the same surface. This means that the diversity measure can only meaningfully differentiate solutions within the same rank, but this is not a problem when diversity is secondary to dominance.
LAHC [56] measures the hypervolume in objective space that a solution uniquely dominates. It is similar to, but arguably more accurate than, NSGA-II’s diversity measure. It is insensitive to the scale of objective dimensions, but is more computationally expensive. Like NSGA-II, it is only accurate for ranks of mutually non-dominated solutions, but since selection is only from non-dominated solutions this is not a problem.

SPEA2 [115] measures the norm-2 (Euclidean) distance from a solution to every other solution in the population, and uses the $k^{th}$ smallest distance as its diversity measure (where $k$ is a user-defined parameter). This is a global measure of diversity, but in SPEA2 diversity is still secondary to dominance.

The selection of $k$ is problematic. When $k$ is low, niches are vulnerable to extinction. For example in nearest-neighbour, when $k=1$, if one previously highly diverse solution is mutated slightly to produce a very similar solution, both solutions will suddenly have a very low diversity, measured to one another. Whilst nearest-neighbour ($k=1$) distributes points excellently across continuous spaces, it leaves small discrete feasible regions, with no room to spread out, vulnerable to extinction. As $k$ increases, the vulnerability to extinction decreases, but it becomes less useful in measuring individual diversity.

PESA-II [21] uses a histogram method that counts the number of other solutions inside the same objective-space box. This is a global measure of diversity, but in PESA-II selection is only from non-dominated solutions, so it is in effect secondary to dominance. The number of objective-space boxes must be user-defined, after which the boundaries of the boxes are scaled to the range of each objective dimension.
3.5.2.5 Examples of Pareto-based multiobjective EAs

MOGA [36] was an early multiobjective genetic algorithm that implemented Pareto dominance ranking. A final fitness score was assigned to individuals by interpolating between a predefined best score for rank 1 and a predefined worst score for the last rank. Fitness sharing was used on objective-space to spread individuals across the Pareto front.

NSGA [90] also used Pareto dominance ranking. However, after ranking, all individuals within a rank were assigned a dummy fitness value which was then modified by fitness sharing on decision-variable-space (not objective-space like MOGA), and sharing only took place inside ranks, not between ranks. The dummy fitness value and sharing ensured that even after sharing, all individuals from one rank had a fitness score higher than the all individuals from lower ranks.

NPGA [50] used Pareto dominance directly inside tournament selection, rather than using Pareto dominance ranking to derive fitness scores as in MOGA and NGSA. The tournament is between two individuals, but to compute the winner a larger sample of individuals is used. The dominance of the two candidates relative to the sample is calculated. Of the two candidates, if one is non-dominated and the other is dominated, the non-dominated individual wins. Else, if both are non-dominated or dominated the winner is the one most dissimilar to the sample in objective-space.

NSGA-II [28] improved upon NGSA to reduce the computational cost of sorting, introduce elitism, and remove the need for explicit sharing parameters. Recent versions of NSGA-II also include constraint-handling capabilities.

PAES [55] used an evolution strategy – a type of EA with no population in which a single parent generates one or more children, and if a superior solution is found it replaces the parent. Pareto dominance was used to compare children to the parent; this
was a direct and simple application of Pareto dominance to an EA. An archive (which did not participate in evolution) of all currently found non-dominated solutions was kept to provide the final solution set.

**PESA-II** [21] was a novel approach in which the basic unit of reproductive selection was not the solution, but the hyperbox – a dynamically assigned regular partitioning of objective-space. The hyperbox’s fitness (minimise), for the purposes of selection, was the number of solutions inside it. Once a hyperbox was selected, a random solution from inside it was used for the actual reproduction. This novel selection scheme was intended to promote diversity in objective space.

**SPEA2** [115] used a Pareto dominance count, as described above. SPEA2’s fitness is interesting in that it reflects both proximity to the Pareto front and diversity using just Pareto dominance.

More recently, multiobjective EAs have begun to use archiving schemes to record the spread of individuals across the Pareto front [20, 22]. There can be an arbitrarily large number of non-dominated individuals by the end of an evolutionary run, and the archiving scheme must keep the best distribution of solutions.

### 3.6 Constrained multiobjective evolutionary algorithms

In addition to being multiobjective, most real-world problems are constrained. The general constrained multiobjective optimisation problem is:

\[
\begin{align*}
\text{minimise } & f_i(x), & i = 1, \ldots, M \\
\text{subject to } & g_j(x) \leq 0, & j = 1, \ldots, N
\end{align*}
\]
It is common to reformulate equality constraints as inequality constraints (e.g. as $g_j(x) \equiv |h_j(x)| \leq 0$ where $h_j(x) = 0$ is an equality constraint), and so we consider only inequality constraints in this work.

A constraint score greater than 0 means the constraint is violated. If any constraints are violated, then the solution is infeasible.

Constrained multiobjective evolutionary algorithms (CMEAs) have received growing attention recently; see for example the survey [13], and specific CMEAs [26, 81, 82].

### 3.6.1 Handling constraints

CMEAs can handle constraints and infeasible solutions in many ways:

- Variation operators (crossover and mutation) can be written to avoid creating infeasible solutions.
- Introduce an additional operator to repair infeasible solutions.
- Infeasible solutions can be discarded.
- Infeasible solutions can be retained in the population and used in the search process.

In many problems it may be necessary to retain infeasible solutions for use in the search process, as finding any feasible solution may be difficult.

When infeasible solutions are retained, the algorithm needs some way of comparing infeasible solutions for selection.

Of CMEAs that do retain infeasible solutions, some algorithms do not treat constraint scores as a multidimensional space and simply sum or min-max constraint violations (collapsing the space into one dimension) or apply penalties back onto objective scores.
(distorting the objective space). Other algorithms do treat constraint space as a space in its own right, in which Pareto dominance ranking can be performed.

### 3.6.2 Constraint space

The use of multiple constraint scores adds another search space: **constraint space**. The vast majority of constraint space represents infeasible solutions. Only the origin of constraint space, where every constraint score is 0, represents feasible solutions. Therefore the search for feasible solutions is a search for the origin of constraint space.

CMEAs need to work with both constraint space and objective space. There are multiple ways for the spaces to interact.

#### 3.6.2.1 Extended dominance

Many algorithms use some form of an **extended dominance relation** (see chapter 9 for a full discussion and references) that replaces the basic Pareto dominance relation to incorporate constraints as well as objectives, thereby bridging the two spaces.

The extended dominance relation is:

1. If both solutions are feasible, dominance is measured in objective space.
2. If one solution is feasible and the other is infeasible, the feasible solution dominates. (3.6)
3. If both solutions are infeasible, dominance is measured in constraint space.

The extended dominance relation, as a replacement to the Pareto dominance relation, can be used in tournament selection or ranking. The effect on ranking is to ensure that a rank is either all feasible or all infeasible, and that all feasible ranks are better than all infeasible ranks. NSGA-II [28] uses this extended dominance relation, but uses a sum of normalised constraint violations rather than Pareto dominance in constraint space.
3.7 Summary

This chapter has given a brief introduction to basic evolutionary algorithms and some advanced topics relevant to this work. EAs are heuristic, population-based search algorithms, with a wide range of genetic operators, encoding schemes, and selection methods that differentiate the different varieties of EAs.

EAs are inherently trivially parallel due to the independence of fitness evaluations. However, different EAs offer coarser- or finer-grained parallelism, and this may influence the choice of algorithm and hardware implementation.

Cooperative coevolution is one such coarse-grained EA, but cooperative coevolution offers advantages to the search algorithm as well as to the parallel implementation. It addresses the curse of dimensionality by decomposing the total search space into smaller spaces that can be searched (largely) independently. However, this introduces the problems of apportionment of credit and problem decomposition.

Constrained multiobjective optimisation is a way of simultaneously optimising multiple, conflicting objectives, subject to multiple constraints. EAs are well suited to Pareto-based multiobjective optimisation due to the population-based search, which can represent an entire trade-off surface of alternative solutions.

In the remainder of this work, chapters investigate the application of these EA techniques to fuzzy systems.

Chapter 6 investigates the relationship between epistasis and encoding in the context of crossover. The relationship between epistasis and encoding – or more generally, application design – is a recurring theme echoed in chapter 7, in the context of coevolutionary problem decomposition and hierarchical rulebases.
Chapter 7 also investigates both parallelism and cooperative coevolution in greater detail.

Chapter 8 investigates diversity measures in multiobjective optimisation, and uses a multiobjective EA to explore controllers for the inverted pendulum problem.

Chapter 9 continues the multiobjective optimisation theme, focussing on constrained multiobjective optimisation, concluding this work by presenting a novel constrained multiobjective evolutionary algorithm.
Chapter 4. Evolutionary Learning of Fuzzy Logic Systems

4.1 Targets for optimisation

A fuzzy system comprises several parts: rulebases; fuzzy sets; and an inference engine. The choice of inference engine is usually a choice between several pre-existing inference engines (for example, Mamdani, TSK, or SAM) and is not considered here. The choices of fuzzy sets (which includes the shape of the membership set and number of sets) and rulebase (which includes rulebase structure and the individual rules therein) are usually not obvious and are targets for optimisation.

The number of fuzzy sets per variable determines how fine-grained the fuzzy system will be, as the fuzzy sets discretise input-space. Increasing the number of fuzzy sets increases the number of rules in a fully specified rulebase.

The distribution and shape of the membership functions describing the fuzzy sets affects the precision over certain areas of input-space; more fuzzy sets may be concentrated around areas of input-space that require more precise approximation.

The structure of the rulebase affects the number of total rules required to cover the given number of input variables and fuzzy sets. The structure may also affect the overall accuracy of the controller [69]. Choices of structure include flat, hierarchical (in which case the hierarchical structure must be defined), and unstructured.
The choice of fuzzy sets and structure of the rulebase determine the “control points” in input-space. The rules define the behaviour of the system, interpolated across those control points.

4.2 Learning membership functions

Fuzzy sets may be defined using local or global semantics [63]. For local semantics, every fuzzy set in every rule has its own membership function definition. For global semantics, membership function definitions are reused between rules. Global semantics gives a clean separation of membership functions from rulebase, allowing learning algorithms to target either or both for optimisation.

The simplest approach to using evolutionary computation to improve fuzzy systems is to start with an existing system and optimise, or tune the shape and position of membership functions (the number of membership functions is fixed, as changing the number of membership functions would change the rules as well). Tuning was one of the earliest applications of evolutionary computation to fuzzy logic systems, perhaps because of its simplicity and perhaps because of the large number of pre-existing fuzzy systems that could be further tuned in this way.

A statistical method was used in [98] to initially estimate membership functions and rules from training data, and then used a genetic algorithm to modify the shape and position of membership functions. The encoding scheme encoded all points for all membership functions separately (e.g., lower bound, centre, upper bound for a triangular function). This allowed any number of sets to overlap, which made the system difficult to interpret and may conflict with some hardware implementations, and so a penalty was applied if the number of sets overlapping any given point exceeded a given constant.
A genetic algorithm was used in [70] to optimise a controller for the truck-back-upper problem with a given rulebase and given input membership functions, derived from human expert knowledge. The GA optimised the membership functions for the output control variable to improve the overall fuzzy system’s performance. The GA encoded the areas of overlap between adjacent membership sets (rather than encoding all points for all sets), which both reduced the search space of the GA (a more compact encoding) and gave a more intuitive fuzzy system overall.

A real-coded evolutionary algorithm was used in [43] to optimise membership functions. The above examples used global semantics – common membership function definitions for all uses of the same variable. This example used local semantics – membership functions were defined on a per-rule basis, not on a per-variable basis. Therefore, the number of rules in the rulebase was variable (as there is no need to “cover” all combinations of input sets), but the search space for the EA was much larger as membership functions were not reused between rules. (The larger search space prompted the use of a real-coded EA rather than a binary bit-string GA.)

4.3 Learning rules

The next simplest use of evolutionary computation is to learn a good set of rules, given membership function definitions.

A rulebase is either structured or unstructured:

- A **structured rulebase** has rules with exactly one antecedent per input variable (“fully specified” rules), and there is exactly one rule for each possible combination of antecedent fuzzy sets. Structured knowledge-bases are generally more accurate, but less interpretable due to the increased size.
• An **unstructured rulebase** has rules that may omit input variables from their antecedents ("partially specified" rules), in which case the rules are satisfied regardless of the value of the unspecified antecedents. This removes the requirement for exactly one rule for each possible combination of antecedent fuzzy sets. Unstructured rulebases might not cover every possible combination of input values, even with general rules. Unstructured rulebases are generally smaller and more interpretable but less accurate.

Further, there are generally two ways of encoding a rulebase:

• The **Pittsburgh approach** is generally used for structured rulebases. Because the number of antecedents, and fuzzy sets per antecedent, are defined, the number of rules necessary to cover every possible combination of antecedent values is fixed. Only the consequents need to be learned. The rules are enumerated and a single solution encodes the consequents to the entire rulebase.

• The **Michigan approach** is only used for unstructured rulebases. There may be any number of rules, and rules may have any number of antecedents. Both the antecedents and the consequents to every rule must be learned. A single individual encodes the antecedents and consequents of a single rule (so an individual is no longer synonymous with a solution). The entire population, collectively, represents a single rulebase (the population becomes the solution).

There are some exceptions, such as variable-length encodings of entire unstructured rulebases which would fall under the Pittsburgh approach, and the iterative rule learning approach (see for example [18]) which would fall under the Michigan approach.
A straightforward GA implementation of the Pittsburgh approach was used in [68], encoding solutions to a single, flat rulebase. In [93, 94] the authors subsequently considered a hierarchical fuzzy system in which a GA was used to learn just a second rulebase that extended the first hierarchically. The first rulebase was either already given or evolved separately, so the task of the GA itself was still to learn a single rulebase. Finally, in [91], an entire hierarchical rulebase was encoded as a single solution. (Chapter 7 continues this research using cooperative coevolution.)

In [45], a hierarchical rulebase (the structure of which was predetermined by the experimenters) was encoded by concatenating the strings encoding each rulebase together into one string; thus each solution encoded an entire hierarchical rulebase. One concern was that the intermediate variables between layers in the hierarchy had no defined meaning, such that two knowledge-bases that produce identical output could do so using different linguistic tags for the hidden variables. This would allow crossover between two phenotypically similar individuals to be destructive. A reorder operator therefore preceded crossover, whereby the linguistic tags for the output from the first rulebase, and for the input to the second rulebase, were changed so as to preserve the functionality but make it more compatible with another rulebase for crossover. The changes were made statistically to maximise the similarity between the two individuals. (Chapter 7 also discusses the problem of the intermediate variables.)

In [46], a variable-length, unstructured Pittsburgh-style algorithm was used. Rules could omit parts of the antecedent (in which case they matched regardless of the omitted variable’s value) and duplicate some parts (in which case duplicate variables were
ignored). Rules that were more specific overrode rules that were less specific, but that would otherwise match.

The algorithm in [42] differed from the above examples significantly by using a Michigan-style algorithm to evolve an unstructured, local-semantics rulebase. An individual did not encode an entire rulebase; it encoded a single rule, so the population as a whole was considered the solution (Michigan-approach). An individual encoded not only a rule’s consequents but also its antecedents (unstructured). Finally, rather than encoding a linguistic tag, an individual encoded the membership set definitions explicitly on a per-rule basis (local semantics). Encoding membership function definitions explicitly would result in a much larger search space (as definitions are not reused between rules), but this was offset by the fact that a single individual encoded only a single rule.

One commonly cited reason for using an unstructured rulebase is interpretability: an unstructured knowledge base can have fewer, more general rules. In [71], interpretability was encouraged by using an encoding scheme for a structured rulebase that allowed consequents to take on a null value, indicating that the rule was unimportant. However, an unstructured approach still has the advantage of allowing more general rules.

4.4 Learning membership functions and rules

When using a rulebase with local semantics, membership sets and rules are inseparable and must be learnt together. Alternatively, global semantics allows membership functions to be learnt independently of rules and vice versa – or, both together. There are two ways of learning both membership sets and rules for global semantics rulebases: either serially in staged learning, or concurrently in an aggregate encoding scheme.
Staged learning is the simpler approach, and it amounts to first learning the rulebase using some straight-forward evolutionary technique such as described above, and then taking the rulebase that results and tuning it, also using one of the evolutionary techniques described above. MOGUL [18] is an example of staged learning. MOGUL uses iterative rule learning, a variation on the Michigan approach.

An aggregate encoding scheme was used in [48], whereby both the rulebase and the membership function definitions were encoded in a single individual. The rulebase was structured and encoded using the Pittsburgh approach. Only the widths of the membership sets were encoded – their shape (isosceles triangle) and centre were fixed. Compared to staged learning, this type of algorithm evolves rules and membership functions definitions concurrently, which increases the search space but has a better chance of finding a perfect “match” between rules and membership sets.

A similar aggregate encoding scheme was used in [74], but encoded both the position and width of membership sets. In [89], all 4 points for full trapezoidal membership functions were encoded.

Cooperative coevolution was used in [5] to accomplish concurrent learning of rules and membership sets. Cooperative coevolution uses multiple distinct populations, each population having individuals encoding one particular aspect of the complete solution. In [5], one population evolved individuals encoding an entire hierarchical rule-base using the Pittsburgh approach. A further three populations evolved individuals encoding membership function definitions – one population for each of three membership shapes (trapezoidal, exponential, and gaussian). The best rulebase and best membership function definitions were combined to form a complete solution. Note that this is different from
staged learning, as all populations are evolved concurrently and impact on each others’ fitness.

4.5 Learning structure

A structured rulebase covers every possible combination of antecedent sets, and as such can grow prohibitively large as the number of inputs to the system and the number of sets per input increase. It is possible to limit the exponential growth of a structured rulebase by using a hierarchical structure of multiple rulebases instead of one monolithic rulebase. A hierarchical structure requires far fewer rules because the number of rules required to cover two subsets of input variables are added together, rather than multiplied. Furthermore, a hierarchical structure may represent the knowledge contained in the system more intuitively and may permit multiple systems to interact more easily.

Usually the topology of a hierarchical rulebase will be determined by the practitioner, using expert knowledge about the problem domain. This is a limiting factor, as the practitioner’s topology design may not be optimal. The difference in performance resulting from different topologies for the same problem (control of the inverted pendulum) was shown in [97, 113].

Learning structure is a harder problem than learning membership definitions or rules. Changing the structure of the rulebase will usually invalidate existing rules. If learning the number of fuzzy sets per variable (rather than just the shape and position of a given number of membership sets) is also attempted, a change in the number of fuzzy sets may also invalidate existing rules and membership function definitions.

There is currently no proposed method of learning hierarchical, fixed structures. There has been some success in learning less rigidly defined rulebase structures.
Island-model coevolution was used in [73] to evolve different rulebase structures in parallel. The rulebases differed in the number of rules they contained. The rules and rulebase had no fixed structure, so the number of rules in any given rulebase was arbitrary (and was specified a priori by the practitioner). Using a Pittsburgh style encoding meant that individuals encoding different numbers of rules were not compatible for crossover. Therefore, in the island-model there were multiple genetically distinct populations ("islands") of individuals and each island evolved rulebases with the same number of rules. A special migration operator allowed a few individuals to move from one population to another by carefully duplicating or removing rules to make them genetically compatible whilst maintaining as closely as possible their functionality.

In [19], the number of fuzzy sets (and therefore the number of rules required to cover them) was variable. They used a hierarchical definition for fuzzy sets. This approach appears to mainly address the problem of globally defined membership functions, as all variables within each level use common membership function definitions. It does not address the problem of learning the type of hierarchical rulebases described above.

4.6 Summary

This chapter has introduced multiple methods by which evolutionary algorithms can be applied to the learning of fuzzy systems. EAs can optimise any combination of membership function definitions, rules, and, to a limited extent, knowledgebase structure. Membership function definitions can be tuned by EAs to improve the accuracy of an existing rulebase. Membership functions can use local or global semantics: in local semantics every instance of a membership function is unique and can be tuned independently; in global semantics rules share common membership function definitions,
reducing the number of decision-variables needed to be optimised and guaranteeing rulebase completeness.

Rulebases can be either structured or unstructured, and can be optimised using either the Pittsburgh approach or the Michigan approach: the Pittsburgh approach typically optimises structured rulebases by encoding a rulebase as a fixed number of consequents with each individual in the population representing an entire rulebase; the Michigan approach typically encodes both the antecedents and consequents of rules, with each individual representing just one rule and the population as a whole representing an entire rulebase.

This the remainder of this work investigates only the evolutionary learning of rules, not membership function definitions or rulebase structures. However, it does consider the design of membership function definitions and rulebase structures, in the context of hierarchical rulebases, in chapter 7.

Chapter 8 discusses the feasibility of learning hierarchical rulebases using Pareto-based multiobjective optimisation.

This work uses the Pittsburgh approach because that is consistent with the equivalence of individuals and solutions, and is therefore compatible with the widest range of EAs that operate on that assumption. This allows the findings from this work to be applied to a wider range of problem domains than just fuzzy systems. In contrast, the Michigan approach is applicable only to rule-based problem domains.

Specifically:
• Chapter 6 investigates a multidimensional encoding, which is applicable to any problem domain with regularly structured multidimensional solutions, but that could not be investigated using the Michigan approach or unstructured rulebases.

• Chapter 7 investigates cooperative coevolution, which requires a representative individual from each population (no single individual can be representative of a Michigan-approach population), and structured hierarchical rulebases, which cannot be represented by the unstructured Michigan approach.

• Chapters 8 and 9 investigate Pareto-based multiobjective evolutionary algorithms, in which the population is a collection of solutions representing the Pareto trade-off surface, which could not be represented by the Michigan approach where the population represents a single solution.

That is not to claim that the Michigan approach is necessarily inferior to the Pittsburgh approach, but it does change the established relationship between the *individual* and the *solution* that many EA techniques are based upon, making the Michigan approach incompatible with the topics investigated in later chapters.
5.1 Introduction

This chapter describes the inverted pendulum problem, a common benchmark control problem. It is described in the context of fuzzy systems and evolutionary algorithms, as it will be used in this context in subsequent chapters.

The inverted pendulum problem is a control problem with 4 state variables: the position of a cart on a railway, $x_1$; the velocity of the cart, $x_2$; the angle of a rigid pole balanced atop the cart, $x_3$; and the angular velocity of the pole, $x_4$. The controller produces one output, $u$, which is a force to modify the cart’s velocity ($x_2$). The controller is to bring all four state variables to 0 — that is, to balance the pole upright at the centre point of the railway.

![Figure 5.1. State variables ($x_1$, ..., $x_4$) and control force ($u$) of the inverted pendulum problem.](image)
The controller is implemented as a fuzzy controller, similarly to [95]. As a fuzzy system, the controller’s rulebase – specifically, the consequents – must be learnt, and this work uses various evolutionary algorithms to do so.

5.2 Model

The model describing the inverted pendulum is taken from [3].

The state-space of the four state variables is bounded, arbitrarily, by the state-space bounds:

\(-1 < x_1 < 1; \quad -1 < x_2 < 1; \quad -\pi/6 < x_3 < \pi/6; \quad -3 < x_4 < 3;\)

where \(x_1\) is the position of the cart, \(x_2\) is cart velocity, \(x_3\) is pole angle, and \(x_4\) is pole angular velocity.

The equations governing the state-variables are:

\[
\begin{align*}
\dot{x}_1 &= x_2 \\
\dot{x}_2 &= u + ml(sin(x_1)x_3^2 - \dot{x}_4 cos(x_3))/(M + m) \\
\dot{x}_3 &= x_4 \\
\dot{x}_4 &= \frac{g sin(x_3) + cos(x_3)(u - mlx_3^2 sin(x_3))/(M + m)}{l/\left((\pi^2 - m cos(x_3)^2)/(M + m)\right)}
\end{align*}
\]  

(5.1)

where \(M\) is cart mass; \(m\) is pole mass; \(l\) is pole length; \(g\) is gravitational acceleration; and \(u\) is the control force on the cart. Units are metres, seconds, radians, and kilograms, respectively.

This work uses \(M = 1\) kg; \(m = 0.1\) kg; \(l = 0.5\) m; \(g = 9.81\) ms\(^{-2}\).

5.3 Fuzzy system

The fuzzy system to control the inverted pendulum uses a fully-specified fuzzy rulebase and the Standard Additive Model inference engine, both described in chapter 2.
There are 4 inputs (the 4 state-variables), each of which is covered by 5 fuzzy sets, resulting in a $5\times5\times5\times5$ rulebase of 625 rules.

All membership functions are triangular, and the highest and lowest membership functions have their upper and lower bounds set to the upper and lower bounds of state-space, respectively, ensuring full coverage of state-space.

The output is covered by 13 fuzzy sets, but as the Standard Additive Model uses additive interpolation, only the centroid of each membership function is required.
The membership sets were not tuned (optimised), as developing an optimal inverted pendulum controller for its own sake was not the aim of this research.

This fuzzy system design is modified in chapter 7, in which a hierarchical rulebase structure is used to facilitate cooperative coevolution.

5.4 Simulation

The above model is initialised with state-variables from one of the test cases in section 5.6, below. The model is then used in a simulation that is run for 20 simulated seconds, updated at 0.02 second time-steps, giving $T_{\text{max}}=1000$ timesteps.

The state-variables are updated using the fourth-order Runge-Kutta method. At every timestep, the current state-variables are fed as input into the fuzzy system, and the fuzzy system’s output is the force $u$ fed back into the simulation model.

5.4.1 Success and failure

The goal of the controller is to bring the state to the origin of state-space $(0,0,0,0)$ and remain stable about this position.

If the controller causes any state-variable to break the bounds of state-space, the simulation is terminated immediately (recorded as timestep $T$) and the controller is deemed to have failed. Because it is impossible for the state to reach the exact origin in finite time, a goal region around the origin is defined by the goal region bounds:
If the controller maintains control of the state (does not break any state-space bounds) for the duration of the simulation, but does not bring the state into the goal region, the controller is still deemed to have failed.

If the controller maintains control of the state and brings the state into the goal region and the state remains inside the goal region at completion of the simulation, then the controller is deemed to have succeeded.

5.5 Fitness evaluation and weights

Evolutionary algorithms require fine-grained fitness evaluations capable of differentiating a wide population of solutions. In this case, a simple success/failure is too coarse-grained to guide an evolutionary search.

The goal of the controller is to bring all four state-variables to a stable position about 0 as quickly as possible. The controller’s ability to accomplish this task can be measured by calculating the mean absolute value of each state-variable, over the duration of the simulation. A running sum of the absolute value of each state-variable is kept, and the mean is calculated when the simulation terminates. A low mean indicates that the state-variable was brought to about 0 quickly, and maintained there for the duration of the simulation, which is desirable. A high mean can indicate either that the state-variable was brought to 0 slowly, or that the state-variable was brought to 0 quickly, but was not maintained about that position - both of which are undesirable.

Note that means, rather than running sums, are used so that the measures can meaningfully compare solutions that terminated at different times. The running sum of a
very poor solution that terminates very early may be less than the running sum of an optimal solution that lasted the entire duration. However, the mean will be greater.

This gives 4 fitness measures, \( f_1, \ldots, f_4 \), measuring the controller’s performance over the 4 state-variables:

\[
f_i = \frac{1}{T} \sum_{t=0}^{T} |x_i^t| \quad \text{for } i = 1, \ldots, 4,
\]

where \( T \) is the time-step that the simulation terminated at (may be less than \( T_{\text{max}} \) if the controller failed early) and \( x_i^t \) is the state variable \( x_i \) at time-step \( t \).

Additionally, the time remaining until the end of the simulation is recorded, as a fraction of the total time allocated to the simulation:

\[
f_5 = \frac{(T_{\text{max}} - T)}{T_{\text{max}}}
\]

If the simulation runs for the entire duration without the state breaking the state-space bounds, \( f_5 \) will be 0. If the simulation terminates early due to breaking state-space bounds, \( f_5 \) will reflect how long the controller was able to maintain control.

This gives 5 fitness measures that must be incorporated into a single fitness function.

The fitness function incorporates an additional penalty if the controller “fails”. At completion or termination of the simulation, the penalty adds 0.1 for each state variable outside the goal region:

\[
p = \sum_{i=1}^{4} \begin{cases} 0, & \text{if } x_i^T \text{ inside goal region} \\ 0.1, & \text{if } x_i^T \text{ outside goal region} \end{cases}
\]

where \( x_i^T \) is the state variable \( x_i \) at termination of the simulation.
The fitness function used in the evolutionary algorithm, to evaluate the fitness of a controller, \( c \), is:

\[
f(c) = \omega_1 f_1(c) + \omega_2 f_2(c) + \omega_3 f_3(c) + \omega_4 f_4(c) + \omega_5 f_5(c) + p(c),
\]

where \( \omega_{1..5} \) are weights that can be varied to evolve controllers with different control behaviours. The weights used in these experiments were:

\[
\omega_1 = 1.0; \quad \omega_2 = 0.25; \quad \omega_3 = 1.75; \quad \omega_4 = 1.0; \quad \omega_5 = 5.0;
\]

which can be interpreted as prioritising: firstly, remaining inside the state-space bounds; secondly, keeping the pole upright \((x_3)\); thirdly, keeping the cart centred \((x_1)\) and minimising the pole’s angular velocity \((x_4)\); and finally, minimising the cart’s velocity \((x_2)\).

This fitness evaluation is modified in chapter 8, in which Pareto-based multiobjective optimisation is used to remove the requirement for weights and to investigate the range of all possible strategies for “successful” controllers.

## 5.6 Test-cases

Typically the controller will be evaluated over multiple test-cases to give an averaged, overall fitness score. Each test-case specifies different starting values for the state-variables. In all cases, the goal of the controller is the same: to bring all state variables to 0 and remain stable inside the goal region.
Table 5.1. Test cases.

<table>
<thead>
<tr>
<th>Test case</th>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
<th>$x_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.25</td>
<td>0.25</td>
<td>0.1</td>
<td>0.1</td>
</tr>
<tr>
<td>2</td>
<td>0.25</td>
<td>0.25</td>
<td>0.1</td>
<td>-0.1</td>
</tr>
<tr>
<td>3</td>
<td>0.25</td>
<td>0.25</td>
<td>-0.1</td>
<td>0.1</td>
</tr>
<tr>
<td>4</td>
<td>0.25</td>
<td>-0.25</td>
<td>-0.1</td>
<td>0.1</td>
</tr>
<tr>
<td>5</td>
<td>0.25</td>
<td>-0.25</td>
<td>0.1</td>
<td>-0.1</td>
</tr>
<tr>
<td>6</td>
<td>0.25</td>
<td>-0.25</td>
<td>0.1</td>
<td>0.1</td>
</tr>
<tr>
<td>7</td>
<td>0.25</td>
<td>-0.25</td>
<td>-0.1</td>
<td>0.1</td>
</tr>
<tr>
<td>8</td>
<td>0.25</td>
<td>-0.25</td>
<td>-0.1</td>
<td>-0.1</td>
</tr>
<tr>
<td>9</td>
<td>0.25</td>
<td>0</td>
<td>0.26</td>
<td>0.2</td>
</tr>
<tr>
<td>10</td>
<td>0.25</td>
<td>0</td>
<td>0.26</td>
<td>-0.2</td>
</tr>
<tr>
<td>11</td>
<td>0.25</td>
<td>0.5</td>
<td>-0.26</td>
<td>0.2</td>
</tr>
<tr>
<td>12</td>
<td>0.25</td>
<td>0.5</td>
<td>-0.26</td>
<td>-0.2</td>
</tr>
<tr>
<td>13</td>
<td>0.25</td>
<td>-0.5</td>
<td>0.26</td>
<td>0.2</td>
</tr>
<tr>
<td>14</td>
<td>0.25</td>
<td>-0.5</td>
<td>0.26</td>
<td>-0.2</td>
</tr>
<tr>
<td>15</td>
<td>0.25</td>
<td>0</td>
<td>-0.26</td>
<td>0.2</td>
</tr>
<tr>
<td>16</td>
<td>0.25</td>
<td>0</td>
<td>-0.26</td>
<td>-0.2</td>
</tr>
</tbody>
</table>

5.7 Summary

This chapter has described the inverted pendulum problem, which will be used throughout the remainder of this work to investigate the application of evolutionary algorithms to fuzzy control systems.

The inverted pendulum problem is used in chapter 6 to investigate appropriate encodings for evolving fuzzy systems using evolutionary algorithms.

Chapter 7 uses the inverted pendulum problem to investigate the cooperative coevolution of hierarchical fuzzy rulebases.

Finally, chapter 8 uses a Pareto-based multiobjective evolutionary algorithm to investigate the range of all possible strategies for “successful” controllers.
Chapter 6. Multidimensional Encoding

6.1 Introduction

This chapter investigates encoding fully-specified fuzzy rulebases into a representation suitable for use in an evolutionary algorithm. Specifically, the structural encoding and its associated crossover operator are examined. The encoding is a potential source of bias in the search, and this chapter identifies two forms of encoding bias: epistatic bias, which is usually desirable, and dimensional bias, which is usually undesirable.

The aim is to find an encoding, and associated crossover operator, for fuzzy rulebases that exploits epistatic bias avoids whilst avoiding arbitrary dimensional bias.

This chapter uses the inverted pendulum problem introduced in chapter 5 as a test problem, and the proposed multidimensional encoding is used for the EAs in chapter 7 and chapter 8.

6.2 Background

6.2.1 Rulebases

This work uses fully specified rulebases that have $N$ inputs and one output. Each input, $x_n$, is covered by a set of membership sets, $M_n$, in which $M_{n,i}$ is the $i^{th}$ membership set covering the $n^{th}$ input. The output, $y$, is covered by the set of membership sets $M_y$.

This allows different inputs to have differing degrees and distributions of fine-grained classification, whilst re-using membership set definitions between different rules. This is the global semantics discussed in section 2.5.
For ease of reference, each membership set may be identified by a linguistic label (e.g. one of Very Low, Low, Medium, High, Very High). (For convenience, these labels may be reused for each input, in which case the same label may actually refer to multiple different membership sets, since membership sets are defined separately for each input. Thus a label is only meaningful in context of a particular input.)

The rulebase is fully specified, so that there is exactly one rule for every possible combination of input membership sets. The number of rules is therefore:

\[
\prod_{n=1}^{N} |M_n|,
\]

where \( N \) is the number of inputs and \( |M_n| \) is the cardinality of the set of the set \( M_n \).

Each rule has the form:

if \( x_1 \) is \( M_{1,j} \) and \( x_2 \) is \( M_{2,j} \) ... and \( x_N \) is \( M_{N,k} \) then \( y \) is \( M_{y,l} \)

where \( x_1, \ldots, x_N \) are input variables and \( M_{1,j}, \ldots, M_{N,k} \) are input membership sets, and \( y \) is the output variable and \( M_{y,l} \) is an output membership set. The “if” part comprises the antecedents; the “then” part comprises the consequent.

(In this work, the terms antecedent and input membership set, and the terms consequent and output membership set, are used interchangeably in context.)

As the rulebase is fully specified, the number of rules and the antecedents of all rules are fixed; they are predetermined by the number of inputs, \( N \), and all the input membership sets, \( M \), used to cover the inputs. All rulebases with the same \( N \) and \( M \) will have the same antecedent structure. Only the consequents of the rules within the rulebase remain to be determined.
Therefore, learning a rulebase (for example, using an evolutionary algorithm) amounts to learning a fixed number of consequents. When set as an optimisation problem, the rule consequents are the decision-variables that must be optimised.

This type of rulebase may be naturally represented as an orderly $N$-dimensional structure, $S$. Each input variable, $x_n$, is one dimension of the structure. The structure is $|M_n|$ units long in dimension $n$ (each unit being one membership set). The structure is thus partitioned into $\prod_{n=1}^{N} |M_n|$ discrete units of space, or cells, with each cell being the one rule that covers that unique combination of input membership sets. Each cell contains one consequent – the output membership set.

The rest of this work, as in the preceding paragraphs, uses the following notation to describe fully specified rulebases, which is also generalised to describe any similarly orderly multidimensional structure of regular cells:

$S$ is an $N$-dimensional structure.

$n$ is an integer $1 \leq n \leq N$ representing a dimension.

$M_n$ is the set of all indices into the $n$th dimension of $S$. So,

$|M_n|$ is the number of indices into the $n$th dimension, or in other words, the width of the structure in the $n$th dimension.

$i_n$ is an index into the $n$th dimension. $i_n \in M_n$. So,

$S_{i_1,i_2,\ldots,i_N}$ is a single cell in the structure. For example, $S_{1,1,1}$ would be the cell at the origin of a 3D structure; or the first rule in a 3-input rulebase.
In the preceding paragraphs, the indices referred membership sets; this notation generalises indices as integers, capable of describing any similar multidimensional structure.

(Membership sets, when not given linguistic labels, are commonly given integer labels. This facilitates mutation in an evolutionary algorithm, for example.)

For example, a two-input rulebase can be visualised as a two-dimensional grid (figure 6.1). Cells in the grid represent rules, and the value of the cell represents the consequent of that rule. A three-input rulebase could be visualised as a cube subdivided into regular smaller cubes (figure 6.2), each of which is a cell, or rule. Further dimensions are more difficult to visualise but in summary, all are naturally multidimensional structures.

One way of displaying a rulebase is in table form. This is useful for higher-dimension rulebases that cannot be printed or visualised in their natural form. In table form, there are \(N+1\) columns (\(N\) columns for inputs; 1 column for output), and one row for each rule. For...
example, figure 6.3 shows the table form of the rulebase displayed in its natural form in figure 6.1.

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>X₁</td>
<td>X₂</td>
<td>Y</td>
</tr>
<tr>
<td>VL</td>
<td>VL</td>
<td>A</td>
</tr>
<tr>
<td>VL</td>
<td>L</td>
<td>B</td>
</tr>
<tr>
<td>VL</td>
<td>M</td>
<td>C</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>VH</td>
<td>H</td>
<td>X</td>
</tr>
<tr>
<td>VH</td>
<td>VH</td>
<td>Y</td>
</tr>
</tbody>
</table>

Figure 6.3. Table form of a 2D rulebase, showing relationship to the natural form of 2D rulebase, above. First two columns are inputs. Third column is output. There would be 25 rows in this table.

Note that in table form, entries (rows) in the table are ordered firstly by antecedent for the first input, secondly by antecedent for the second input, and so on for however many inputs there are (in this case just two). (This is referred to as a **row-major encoding**, since in figure 6.1, the first input forms rows in the grid.) Since the combination of antecedents uniquely identifies a rule, the result is a fixed ordering of rules in the table. This means that an entire rulebase can be **encoded** as just the final column of consequents, since as long as the order of consequents is preserved, the antecedents can be reconstructed and the rule that each consequent belongs to can be determined.

### 6.2.2 Encoding
Genetic Algorithms, the first evolutionary algorithms, originally adapted solutions to the algorithm by means of an **encoding**. The algorithm’s operators were designed to work
with a bit-string, and it was the responsibility of the encoding to transform the solution’s
decision-variables, whatever they were, into this form.

Since GAs, many other EAs have been created with different operators designed to
perform better on different encodings, which in turn may better represent the solution.

A common encoding, suitable for many problems that do not have highly specific
requirements, is the vector of real-valued decision-variables. This is eminently suitable
for all problems where the solution is naturally a vector, but it has also proven popular
and effective for countless problems where the solution is not a vector, but a loosely-
coupled collection of decision-variables. It is so common that the vector of decision-
variables, along with its associated crossover operator, is often regarded as a defining
feature of evolutionary algorithms (specialised applications such as genetic programming
and VLSI layout notwithstanding).

6.2.2.1 Structural encoding

Conceptually, we can distinguish between encoding the solution’s structure – e.g., a
vector – and encoding the solution’s decision-variables – e.g., real-valued. This
distinction is illustrated in a revision of the diagram from chapter 3:

Figure 6.4. Design of an EA, distinguishing between encoding variables and structure.
For the remainder of this work, encoding refers to the encoding of the solution’s structure.

6.2.2.2 Epistasis
In biology, epistasis is the effect of one gene upon the phenotypical expression of another. Analogously, in evolutionary computation, epistasis is the effect that one decision-variable has upon the contribution of another decision-variable to the fitness evaluation [38]. Intuitively, it is a coupling between two (or more) decision-variables that is significant to the performance of the solution.

The relevance of epistasis to encoding is that, in many cases, encoding a solution as a vector is an arbitrary design choice. A vector is often chosen to encode solutions that are collections of loosely-coupled decision-variables. In this case, the designer must decide where in the vector each decision-variable is encoded. Knowledge of epistatic links between decision-variables often guides this design.

Knowledge of epistasis can be used to guide the evolutionary search. Changing just one of a pair of highly epistatic decision-variables will have a dramatic effect on the contribution of the unchanged decision-variable to the fitness evaluation, making it difficult to optimise epistatic decision-variables independently of one another. This suggests that in general, pairs (or groups) of mutually epistatic decision-variables should be kept, or changed, together.

6.2.2.3 Exploiting epistasis
Exploiting epistasis is the heuristic of reducing the probability of changing highly epistatic decision-variables independently of one another. It is a conservative, or greedy, heuristic in that it commits to a partial solution easily.
Exploiting epistasis attempts to serve two purposes: first, it reduces dramatic, unpredictable changes in fitness evaluation, making the search process more reliable (recall that a tenet of evolution is that children are similar to their parents); and second, it frees up the exploratory search to focus on other (fewer) decision-variables. It exemplifies the “building blocks” view of evolutionary algorithms, wherein solutions improve incrementally, keeping those parts that have been proven to work well.

Exploiting epistasis is in opposition to open exploration. However, the absence of exploration does not imply the presence of exploitation. Exploiting epistasis requires actively directing the search away from mutually-epistatic decision-variables, and towards non-epistatic decision-variables. The absolute rate of exploration is peripheral to the heuristic, but the rate of exploration, relative to degree of epistasis, is central.

6.2.2.4 Schemata
Schemata are used to describe the “building blocks” of the evolutionary search [38]. They are conceptual entities that can be used to understand how the population evolves. As conceptual entities, schemata are not manipulated explicitly by EAs.

A schema, at its most general, is a template for a complete solution, in which certain decision variables have specified values and remaining decision variables have unspecified values.

\[
\begin{array}{cccccccc}
A & * & * & * & F & * & * & * & K & * & * & P & * & * & * & U & * & * & * \\
\end{array}
\]

Figure 6.5. A schema with 5 specified decision-variables, corresponding to \( x_2 = VL \) in Figure 6.3

Since the vast majority of EAs use a vector encoding, schemata are usually also represented as vectors corresponding to the encoding, with one additional possible value for each decision-variable: the wildcard, which indicates that that particular decision-variable is unimportant in that particular schema.
Simple schemata, specifying only a few decision variables, are useful for understanding the effects of epistasis on the search, and how the EA operators – specifically crossover – can exploit it. This work uses schemata to understand epistasis, and so **schemata exploitation** is synonymous with exploitation of epistasis, and **schemata exploration** means exploration of new schemata without (or with little) regard for epistasis.

### 6.2.3 Crossover

Of all the operators of an EA, only crossover is coupled with the structural encoding. Selection operates on fitness evaluation, to which the encoding is invisible. Mutation does operate on encoding, but only as far as the encoding of independent decision-variables (i.e., bit-string, real-valued (continuous or integer), symbolic).

#### 6.2.3.1 Crossover vs. mutation

Both crossover and mutation modify solutions to generate new children. However, mutation is usually disregarded as significantly affecting the exploitation of epistasis, for several reasons:

- **The probability** of mutation is very low, typically set to produce just one expected mutation per child. In contrast, the probability of crossover is very high.

- **The scope** of a single mutation, when it occurs, affects just one decision-variable. In contrast, crossover, when it occurs, can affect large numbers of decision-variables (depending on crossover points).

- **The magnitude** of mutation is typically low, at least for real-coded decision variables that are typically mutated using a Gaussian distribution. In contrast, the magnitude of the change wrought by crossover is limited only by the genetic diversity present in the population.
For these reasons, although it is possible for mutation to interact with epistasis by mutating just one of a pair (or group) of mutually epistatic decision-variables, we focus on the effect of crossover on the exploitation of epistasis.

6.2.3.2 Disruption
Let us assume that a given schema specifies a group of two or more decision-variables that have a high degree of mutual epistasis. Disruption occurs when one (or more) of the decision-variables in the schema is changed independently of the others. For the purposes of this chapter we assume that this disruption is caused by crossover, not mutation, and that the two parents have different values for the decision-variables so that crossover does actually produce a change of value in the schema.

The defining length, $L$, of a schema is the distance (in the encoding, and assuming a vector) between the first and the last specified decision-variable.

The probability of disruption, $P_D$, is the probability that crossover will change some, but not all, of the decision-variables in the schema. $P_D$ depends on the defining length and the number of crossover points.

If schemata are used to represent groups of mutually epistatic decision-variables, then avoiding disruption is exploiting epistasis. A low $P_D$ promotes schemata exploitation; a high $P_D$ promotes schemata exploration.

6.2.3.3 Disruption and crossover points
With 1-point crossover, in which a single cut is made randomly anywhere along the vector, it is obvious that the greater $L$, the greater the $P_D$: there are more chances of the cut falling between decision-variables in the schema, swapping one but not all of them with the other parent. The same is true of any crossover with an odd number of cuts.
However, if there are an even number of cuts between the schema’s decision-variables, only the unspecified decision-variables between the even cuts will be swapped out, and the schema’s specified decision-variables will be swapped together and there will be no disruption.

It was shown in [25] that $P_D$ always initially increases as $L$ increases, and with 1-point (or any odd-numbered) crossover, $P_D$ continues to increase as $L$ increases. However, with even-point crossover $P_D$ initially increases with $L$, but reaches a turning point where $P_D$ begins to decrease as $L$ becomes large enough to probably contain all crossover cuts.

Another way of looking at odd-point crossover vs. even-point crossover is as strings vs. loops. In odd-point crossover, the first and last decision-variables on the vector will always be disrupted no matter which crossover points are chosen, like two endpoints on a string cut in the middle.

In even-point crossover, the first and last decision-variables on the vector will never be disrupted no matter which crossover points are chosen, they will always go to the same child.
In even-point crossover the vector, with first and last decision variables effectively adjacent and inseparable, becomes more like a loop that has had segments cut out of it with even numbers of cuts.

![Diagram of 2-point crossover as a loop](image)

Even-point crossover, on a loop, has a lower total $P_D$ because the greatest possible defining distance, $L$, between any 2 decision-variables becomes the distance to the opposite side of the loop, halfway around the loop; what would be only halfway along the open string. The greatest possible $L$, for two decision-variables, in even-point crossover is effectively half that of odd-point crossover. Supporting this view, the turning point for $P_D$ is when $L$ equals half the length of the decision vector. (Even-point $L$ may be larger than half the vector when the schema has more than 2 decision-variables, but will still be less than (or equal to, in a worst-case) odd-point $L$.)

For the remainder of this work, $L$ is taken to mean the effective $L$; that is, taking into account the conceptual loop encoding when even-point crossover is used.

These observations suggest that even-point crossover is preferable to odd-point crossover, since total $P_D$ over all $L$ is reduced. They also suggest that fewer crossover points are preferable to more crossover points, since the more crossover points there are, the more
chances there are of one falling inside the schema. This is why 2-point crossover is so common: it is the lowest even-point crossover.

6.2.4 Relationship of crossover and encoding, to disruption and epistasis

Exploiting epistasis requires a relatively lower probability of disruption, $P_D$, for mutually epistatic decision-variables. It does not require a lower $P_D$ over all possible schemata; just epistatic schemata. Indeed, disruption outside of epistatic schemata is a powerful exploratory mechanism in the evolutionary search. This interplay is central to the heuristic of exploiting epistasis.

A schema’s $P_D$ due to any $n$-point crossover, varies with its defining length, $L$, in the encoding. Therefore, if it is possible to design the encoding such that epistatic schemata have a low $L$ (and that the more epistatic a schema is, the lower its $L$), then the above criteria are met: disruption of epistatic decision-variables is avoided (thereby exploiting epistasis) whilst disruption of other variables is encouraged (thereby exploring schemata). Therefore, it is a general rule of thumb in designing encodings that the greater the epistasis between two decision-variables is, the closer together they should be in the encoding (which is usually a vector). In other words, the encoding should minimise $L$ for epistatic schemata.

In summary: The heuristic of exploiting epistasis can be implemented by deliberately introducing bias into the encoding, with the understanding of how that bias will affect the effect of crossover.
6.2.5 Uniform crossover

In uniform crossover [99], each decision-variable has the same independent probability of being crossed over. This probability, $P_U$, is typically 0.5 but can vary from 0.0 to 0.5. $P_U > 0.5$ is effectively equivalent to $1-P_U$. $P_U$ can also be interpreted as the probability of any given decision variable in parent 1 going to child 2, in which case $1-P_U$ would be the probability of any given decision variable in parent 1 going to child 1. Since both children are generated and neither has any further association with either parent, $P_U$ is effectively equivalent to $1-P_U$.

The independent probability for each decision-variable of being crossed over has led to the idea that uniform crossover is excessively disruptive: closely encoded epistatic decision-variables are just as probable to be disrupted as are distantly encoded non-epistatic decision-variables. Assuming that the encoding has been designed to minimise $L$ for epistatic schemata, it is true that uniform crossover has a “higher” probability of disrupting epistatic schemata, relative to non-epistatic schemata. (Contrast this with $n$-point crossover, with appropriate encoding, in which epistatic schemata have a lower $P_D$ relative to non-epistatic schemata.)

However, [25] showed that the overall $P_D$ of uniform crossover need not be higher than $n$-point crossover. $P_D$ is directly proportionate to $P_U$ (up to 0.5, as explained above). Very low $P_U$ means that almost all decision-variables will be crossed over together (i.e., almost all decision-variables from parent 1 will go to child 2), resulting in low overall disruption. As $P_U$ approaches 0.5, it becomes equally likely that a decision variable will be crossed over or not (i.e., roughly half will go to child 1 and half will go to child 2), resulting in high overall disruption. Therefore, in uniform crossover, overall disruption can be directly controlled by $P_U$.  

102
6.2.5.1 Uniform crossover vs. n-point crossover

Another way of looking at it is that in uniform crossover, $P_D$ is controlled by $P_U$. In contrast, in $n$-point crossover, $P_D$ is controlled by $L$ and $n$. Overall $P_D$ can be controlled; it is not the main difference between uniform and $n$-point crossover.

The main difference is that uniform crossover’s $P_D$ does not depend on encoding (i.e., $L$), whereas $n$-point crossover’s $P_D$ does. This is both an advantage and a disadvantage, depending on the problem.

The advantage is that it may be difficult or impossible to design an encoding that satisfactorily minimises $L$ for epistatic schemata. With uniform crossover, the encoding is irrelevant; all encoding bias is removed.

The disadvantage is that, because uniform crossover removes all encoding bias, it fails to meet the criteria of exploiting epistasis. In uniform crossover $P_D$ is the same for all schemata (with the same number of specified decision-variables), whereas in $n$-point crossover $P_D$ can differ between schemata (ideally corresponding to epistasis by using an appropriate encoding design, and leading to exploitation of epistasis).

However, it should be noted that regardless of $P_U$, with uniform crossover it is possible to copy any possible schema (with probability depending on the number of specified decision-variables and $P_U$, of course). With $n$-point crossover it is impossible to copy some schemata due to the fixed number of crossover points.

6.2.6 Encoding bias and alternative encodings/crossovers

In the preceding sections, it has been taken for granted that it is possible to design an encoding that satisfactorily minimises $L$ for epistatic schemata. Depending on the problem and the encoding structure (or more accurately, the encoding structure that the crossover operator expects), this may not be true.
Typically, designing an encoding that minimises $L$ for epistatic schemata would mean ordering decision-variables appropriately in a vector encoding. This may work satisfactorily if the decision-variables are loosely coupled and unstructured. However, it may be difficult or impossible to map a naturally multidimensional solution structure onto a single-dimensional structure (i.e., a vector) whilst maintaining relationships between decision-variables in all dimensions equally. (We assume that in naturally multidimensional problems, decision-variables closer to one another in the solution structure tend to be more epistatic.)

In such a case, an inappropriate encoding would bias the search by increasing the probability of disruption, $P_D$, between some epistatic decision-variables relative to some other, equally epistatic, decision-variables.

For example, consider mapping a 2-dimensional grid-type solution structure onto a 1-dimensional vector-type encoding using a simple column-major encoding (or row-major, to equivalent effect). In this case, decision-variables that are vertical neighbours in the grid will have a low defining distance, $L$, in the encoding, but decision-variables that are horizontal neighbours in the grid will have a high $L$ in the encoding. However, there might be no reason to assume that vertical neighbours are more epistatic than horizontal neighbours. (See for example figure 6.1 (a 2-dimensional grid-type solution structure) and figure 6.3 (a row-major encoding of figure 6.1).)

This results in an undesirable, and unintended, encoding bias (contrast with the heuristic, deliberate encoding bias intended to exploit epistasis).

To avoid this undesirable dimensional encoding bias, the designer might choose uniform crossover, for which $P_D$ does not depend on $L$. However uniform crossover removes all
encoding bias, both desirable (e.g. exploitation of epistasis) and undesirable (e.g., dimensional bias). Another option is to consider encodings and crossovers that are not vector-based, with the intention of avoiding unintended encoding bias whilst retaining deliberate encoding bias.

6.2.6.1 2-d crossover examples
A genetic algorithm was used in [17] to search for component placements in VLSI (Very Large Scale Integration) circuit layouts. The layout was 2-dimensional, and the solutions used a corresponding 2-dimensional encoding. Crossover generated a new child by copying a rectangular section from the first parent, and everything else from the second parent.

In [62], a genetic algorithm was used to search for seismic velocity models that fit observed data. The seismic velocity models were two-dimensional vertical cross-sections of the Earth’s crust, divided into cells representing the velocity at which seismic waves pass through the crust at that location. The task of the GA was to reconstruct the seismic velocity model that predicted the observed seismic events with minimal error.

The GA was tested with traditional 2-point 1-dimensional crossover and 2-point 2-dimensional crossover. For 1-dimensional crossover, the seismic model was encoded as a 1-dimensional string. This encoding kept a low $L$ between cells adjacent to one another in one of the original dimensions, but created a high $L$ between cells adjacent in the other original dimension. For 2-dimensional crossover, the seismic model was encoded as a 2-dimensional grid with decision-variable locations corresponding to the cells in the original model. Then, crossover was performed randomly either horizontally or vertically across the grid. Results showed the 2-dimensional encoding and crossover to be superior.
In [7], \( n \)-point crossover was generalised to any number of dimensions. The encoding to be operated on by the crossover operator was assumed to be a multi-dimensional grid of decision-variables. \( n \) crossover cuts were made. Each cut was a hyperplane perpendicular to a randomly selected dimension at a randomly selected distance along that dimension. Each cut bisected the multidimensional structure, and all cuts together defined the segments to be crossed over. Segments were copied alternately into the two children such that segments sharing the same section of a cut were copied into opposite children.

6.2.6.2 Geographical crossover example

The work in [7] was extended in [54] by recognising that a limitation of existing multidimensional crossover is the reduced number of potential crossover sites: that is, schemata exploration is reduced. Basic multidimensional crossover operators use hyperplane cuts that are perpendicular to one dimension. To address this, [54] presented “geographic” crossover that was not limited to perpendicular hyperplanes. Each cut was formed by starting to cut from two different edges of the hypercube, and monotonically growing the cuts in a random walk through the structure, until the two cuts could be connected by a perpendicular hyperplane.

However, the work in [7, 54] did not respect the underlying structure of the solution. The encoding of the solution was designed to fit the new crossover operator, rather than being designed to fit the solution’s natural structure. \( L \) was not satisfactorily minimised for all equally epistatic schemata, thereby allowing bias to still exist in the encoding and crossover.

Geographical crossover was implemented on a number of graph problems with differing structures. Graph nodes were mapped onto either 2-dimensional or 3-dimensional
encodings. Heuristics were used to map graph nodes to positions in the encoding: Either the Euclidean distance (in the 2-d or 3-d encoding) between two nodes sharing an edge (in the graph) was minimised, or the graph was traversed depth-first (from an arbitrary starting node) to produce a 1-dimensional ordering of nodes, and that 1-dimensional ordering was mapped onto the 2- or 3-d encoding using row-major ordering.

The inherent arbitrariness of this method is concerning.

6.2.7 Schemata exploration

Besides expressing encoding bias (or not, in the case of uniform crossover), choice of crossover has another effect on the evolutionary search: schemata exploration.

One way of thinking about schemata exploration is the number of different potential crossover cuts on the encoding used with the crossover operator. This is different from the number of actual crossover cuts performed, which for example is \( n \) for \( n \)-point crossover.

**Traditional crossover** (vector-encoding-based crossover) has a number of potential crossover cuts equal to the length of the vector; i.e., the total number of decision-variables (versions that disallow crossover at the very end of the vector will have 1 less potential cut).

Simple **multi-dimensional crossover** that uses axis-parallel hyperplane cuts as in [7] has a number of potential crossover cuts equal to the sum of the lengths of the sides of the multidimensional encoding structure. The number of potential crossover cuts is

\[
\sum_{n=1}^{N} |M_n|
\]

where \( M_n \) is the set of indices into dimension \( n \): in other words, \( |M_n| \) cuts per dimension.

Note that this is less than traditional crossover, which, expressed in the same notations, is
that is, the product (vs. the sum, above) of the lengths of the sides of the multidimensional structure, or the total number of elements (decision-variables) in the structure.

The number of potential crossover cuts for geographical crossover was not proven in [54]. However, because geographical crossover performs a random walk through the structure rather than using perpendicular hyperplanes, geographical crossover can produce many different cuts from each starting crossover cut, resulting in a greater number of effective crossover sites compared to simple multi-dimensional crossover.

Uniform crossover has a number of potential cuts equal to the total number of decision-variables, like traditional crossover. However, whereas both traditional crossover and multi-dimensional crossover have a fixed number of actual crossover cuts (i.e., \( n \) in \( n \)-point crossover), uniform crossover has a variable number of actual crossover cuts. Since each decision-variable has an independent probability of being crossed over, the expected number of actual crossover cuts increases as \( P_U \) approaches 0.5.

Another way of thinking about schemata exploration is the number of different schemata that a crossover operator can copy.

Traditional 1-point crossover is limited to copying schemata that are contiguous blocks of decision-variables anchored at either end of the vector. In general, even-pointed \( n \)-point crossover is limited to copying schemata with exactly \( n/2 \) blocks of contiguous decision-variables, and odd-point \( n \)-point crossover copies \((n+1)/2\) blocks, with one anchored at either end of the vector.
Multi-dimensional $n$-point crossover has fewer potential crossover cuts than traditional $n$-point crossover, but is capable of copying schemata that traditional crossover, with an equivalent number of points, cannot. This is because traditional crossover must make multiple cuts on a vector encoding to emulate a single cut on a multidimensional encoding (imagine a many-folded piece of paper: one cut on the folded (multidimensional) paper makes multiple cuts when the paper is unfolded (single-dimensional)).

Finally, uniform crossover is capable of copying any possible schemata, albeit with a greater or lesser probability depending on the number of decision-variables specified and $P_U$. [79] describes some criteria by which a recombination operator’s effect on schemata exploration can be understood: respect for decision variable values common to all parents; strict transmission of values from parents to children; and proper assortment – the ability to copy any combination of compatible schemata from parents to children. It is the latter, proper assortment, that distinguishes uniform crossover from all of the above crossover operators.

### 6.2.8 Other sources of bias
Search bias need not result only from solution-to-encoding mapping. In [51], search bias resulting from solution-to-fitness mapping was identified. As shown in figure 6.4, fitness evaluation is entirely separate from encoding. Usually multiple solutions of differing decision-variable values can map to the same fitness evaluation. Due to this redundancy, an unbiased search – for example, a random walk – in decision-variable space can be biased towards redundant areas of fitness space. Alternative designs for the solution structure and fitness function can result in different degrees of redundancy and therefore
bias. If this bias is undesirable, or even if the usefulness of the bias cannot be determined either way, [51] advocated changing the solution structure and fitness evaluation to minimise this bias as much as possible.

As a caveat, it is worth noting that changing the solution structure and fitness evaluation is usually outside the scope of EA design and application, and therefore this work ignores this potential source of bias.

6.3 Motivation for research

A fully specified, global semantics rulebase can be encoded as a fixed number of decision-variables, each of which is a single consequent for a single rule. The value of the decision variable is an integer identifying the membership set used for that consequent. Integer values are assigned to represent membership sets such that consecutive integers represent neighbouring membership sets. This aids mutation, so that a small mutation (e.g. +/- 1) results in a small change to the consequent’s membership set (e.g. from Very High down to High).

This section examines what can be known a-priori when encoding a rulebase, and how that knowledge can guide the encoding (and crossover) design.

6.3.1 Epistatic encoding bias

For all fuzzy rulebases, without any further knowledge of the problem domain, the designer can make two assumptions:

Firstly, that there exists some epistasis between neighbouring decision-variables (consequents) in the multidimensional rulebase structure. Membership sets covering inputs are designed to overlap in order to give smooth coverage. This causes rules to overlap in input-space, so that multiple neighbouring rules will cover every possible
input. When the fuzzy system is given particular input data, the consequents from all matching rules are combined by the inference engine according to the matching strength of each rule, in order to give a smooth output from the fuzzy system. Therefore, for any given rule, the consequent that will result in the most accurate output from the inference engine will depend on the neighbouring consequents. In other words, the optimal value for every decision-variable depends on the values of neighbouring decision-variables, resulting in some degree of epistasis between neighbouring decision-variables. This epistasis will usually be highly localised, since membership sets usually overlap only with their immediate neighbours.

Secondly, that there is no reason to assume that the epistasis is any stronger for one input dimension’s membership sets than it is for any other input dimension. In the absence of a deeper understanding, the designer must assume that all dimensions are equally epistatic when designing the encoding.

6.3.1.1 Function approximation vs. control
The above two assumptions apply to any fuzzy rulebase. An additional assumption is applicable to rulebases for fuzzy controllers, as opposed to fuzzy function approximators.

A function approximator’s fitness is evaluated by testing it on a collection of known data points. For each data-point, the rules that cover the data point are processed by the inference engine, and an approximation is output.

A typical fitness evaluation is the root mean square error of the individual approximations for all data points. It is important to note that the contribution of one approximation’s error, to the overall fitness evaluation, does not depend on the error of any other
approximation. In other words, there is no “epistas is” between data-points, unless they happen to be so close that they are matched by the same rules. There is no epistasis beyond the epistasis resulting from overlapping membership sets, as described previously.

A controller’s fitness is evaluated differently. The controller is given a starting position in state-space; this is its input. Given a position in state-space as input, the controller produces an action as output. The action changes the current state, thereby changing the controller’s position in state-space (and leading to the next input). The controller’s task is to iteratively monitor its state (input) and produce an action (output) that will move it to a more favourable position in state-space. Its goal is to move through state-space, and eventually reach a position in state-space that satisfies some condition(s).

The controller is still essentially an input-output function approximator. Each input is matched by one or more overlapping rules, and the inference engine generates an output from the rules’ consequents. The important point is that the controller is not evaluated on a pre-selected set of input-output data-points. Instead, the controller’s output (action) determines the subsequent input (position in state-space). This results in epistasis beyond the epistasis resulting from overlapping membership sets. The contribution of one rule, to the overall fitness evaluation, does depend on the contribution of other rules. In a typical control problem, it is impossible to determine the appropriateness of a particular behaviour at a particular point in time in a particular state (this is known as the apportionment of credit problem). Only the overall effectiveness of the controller can be measured, and this depends on a long chain of appropriate behaviours (outputs) for
each state (input) encountered. This leads to interdependence between many rules – epistasis between many decision-variables.

The patterns of epistasis in the rulebase will be highly problem-dependent, and will even change depending on the rulebase itself. This makes generalisations, intended to exploit epistasis at an algorithmic level, difficult. However, it is possible to assume that, in general, epistasis between two decision-variables will be inversely proportional to their distance in the multidimensional rulebase structure. This is because the multidimensional rulebase structure corresponds to state-space, and a controller’s output moves it smoothly through this continuous space. The closer together two decision-variables are in the corresponding state-space, the more likely they are to contribute to the same controller behaviour, and the greater their epistasis.

6.3.2 Dimensional encoding bias

The most common way of encoding a rulebase is to use the table form of a rulebase: enumerate every possible rule and store the consequent to that rule in the corresponding position in a vector. This vector can then be easily used with almost any evolutionary algorithm. For example, figure 6.3 shows a row-major encoding of figure 6.1.

However, the choice of row-major encoding in figure 6.3 was arbitrary; it might just have well been column-major encoding. Indeed, the assignment of particular fuzzy system inputs (e.g. $x_1$) to particular dimensions (e.g. vertical) in figure 6.1 was arbitrary in the first place. This should suggest that, in the absence of a deeper understanding of the problem domain, one should not assume that any one dimension in the original multidimensional structure has greater epistatic links than any other dimension.
Given that that is the case, any single-dimensional encoding will produce schemata of equivalent epistasis, but different defining length, \( L \). Using \( n \)-point crossover, the different \( L \) will lead to an unwarranted higher probability of disruption, \( P_D \), for certain schemata. The higher \( P_D \) for some epistatic schemata will make the decision-variables in those schemata harder to optimise by evolutionary search, by acting contrary to the heuristic of exploiting epistasis.

In the particular case of row-major (or equivalent) encoding, schemata containing decision-variables from the same row will have lower \( L \) than schemata containing decision-variables from the same column (or any other dimension).

In the general case, for any single-dimensional encoding of a multi-dimensional rulebase, it is impossible to encode all equally epistatic schemata with equal \( L \): Consider that a decision-variable can have \( 2N \) immediate neighbours in an \( N \)-dimensional structure, but only 2 immediate neighbours in a 1-dimensional structure (vector).

### 6.3.3 Exploiting epistasis vs. dimensional encoding bias

The common single-dimensional row-major encoding described above has two distinct but related shortcomings:

- Epistasis cannot be effectively exploited because \( L \) (and therefore \( P_D \)) does not correspond to degree of epistasis.
- The search is arbitrarily biased because \( L \) (and therefore \( P_D \)) does correspond to the dimension along which the schemata stretch.

The above discussion identified two potential sources of bias when encoding rulebases: epistatic bias and dimensional bias. This illustrates the difference between useful bias and arbitrary bias. Search bias in itself is not a bad thing; useful bias can be exploited to
improve the algorithm. However, arbitrary bias should be avoided. The difference is that useful bias is supported by reasonable assumptions about the problem domain, but arbitrary bias results from unfounded, and possibly unintended assumptions.

An appropriate encoding (and associated crossover) will exploit known, useful bias, and avoid unknown, arbitrary bias. Such an encoding, for fuzzy rulebases, is suggested below.

### 6.3.4 Appropriate encoding

Given the preceding discussion, an appropriate encoding for fully specified fuzzy logic rulebases is immediately obvious. A multi-dimensional encoding corresponding to the rulebase’s natural structure will satisfy all the criteria previously identified:

1. Schemata $L$ is correlated with what epistasis can be assumed (between immediately neighbouring, overlapping rules (for all fuzzy rulebases), and between closely neighbouring rules (for controllers)).

2. Dimensional encoding bias is avoided.

This study focuses on the encoding of fuzzy rulebases, *not* improving the basic crossover operator. For a discussion on improving the multi-dimensional crossover operator independently of encoding, see [49].

It is coincidental that it is through the crossover operator that encoding bias is expressed, and it is therefore the crossover operator that must be modified. We have therefore selected the most basic multi-dimensional crossover because it most closely matches the equivalent basic traditional crossover. There is no difference in principle to the way in which the multi-dimensional/traditional crossover operators work.
6.3.4.1 Multi-dimensional crossover

The multi-dimensional crossover operator takes an $N$-dimensional structure of decision-variables, $S$. It selects a random dimension, $n$ (where $1 \leq n \leq N$), and a random crossover point, $p$ (where $1 \leq p < |M_n|$ and $M_n$ is the set of indices (in this case, membership sets) into dimension $n$) and performs single-point crossover along that dimension: All decision-variables on one side of the cut (i.e., all cells in the set $\{S_{i_1,\ldots,i_{n-1},i_n} : i_n < p\}$) go to one child, all other points go to the other child.

![Figure 6.9. 3D crossover along dimension $x_2$: $n = 2$, $p = 3$](image)

Note that multi-dimensional crossover where $n = 1$ is (almost) identical in effect to traditional crossover because when a multi-dimensional structure is mapped onto a vector, elements are grouped by the first dimension. Traditional crossover therefore effectively always cuts along the first dimension. Multi-dimensional crossover where $n > 1$ is effectively what would happen with traditional crossover, if a different encoding was arbitrarily chosen to begin with. Multi-dimensional crossover is essentially not doing anything more sophisticated than traditional crossover; it is behaving as though a different multi-to-single-dimensional encoding was chosen (randomly, without bias) for each crossover operation.
6.4 Experiments

6.4.1 Encoding and crossover operators

The following encodings and crossover operators were chosen for the experiments.

- **Traditional** single-dimensional, 1-point crossover on a vector encoding.
- **Fixed multi-dimensional**, 1-point crossover on a multidimensional encoding where the cut dimension is fixed at $n = 1$.
- **Random multi-dimensional**, 1-point crossover on a multidimensional encoding where the cut dimension is selected randomly.
- **Uniform** crossover where $P_U = 0.5$ and the encoding is irrelevant.

<table>
<thead>
<tr>
<th>Crossover</th>
<th>Dimensional bias</th>
<th>Epistasis exploitation</th>
<th>Schemata exploration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Traditional</td>
<td>High</td>
<td>Low</td>
<td>Medium</td>
</tr>
<tr>
<td>Fixed multi-dimensional</td>
<td>High</td>
<td>Low</td>
<td>Very low</td>
</tr>
<tr>
<td>Random multi-dimensional</td>
<td>None</td>
<td>High</td>
<td>Low</td>
</tr>
<tr>
<td>Uniform</td>
<td>None</td>
<td>None</td>
<td>High</td>
</tr>
</tbody>
</table>

The basic aim is to investigate the difference between a traditional vector-based encoding (and crossover) and a multi-dimensional encoding (and crossover). The hypothesis is that multi-dimensional encoding, which removes undesirable (dimensional) bias whilst exploiting desirable (epistatic) bias, will improve the convergence of the evolutionary algorithm. However, several factors confound a direct comparison between the two.

The first confounding factor is the difference in schemata exploration between traditional crossover and multi-dimensional crossover. Specifically, the number of potential crossover sites is less in multi-dimensional crossover. The concern is that any observed
improvement in multi-dimensional crossover may be attributable to faster evolutionary convergence due to less schemata exploration.

To address this concern, fixed multi-dimensional crossover is introduced. Fixed multi-dimensional crossover fixes the cut dimension at $n = 1$, recreating the same dimensional bias caused by row-major encoding. Comparing these three crossovers should help distinguish between the effects of dimensional bias and schemata exploration: if the difference is attributable mainly to schemata exploration, then fixed multi-dimensional should be more similar to random multi-dimensional (both of which have low schemata exploration); but if the difference is attributable mainly to dimensional bias, then fixed-multi-dimensional crossover should be more similar to traditional crossover (both of which have dimensional bias).

It is worth noting that all three of these crossovers have the same $P_D$, over all possible schemata. Therefore schemata exploration does not refer to the overall rate of disruption, but rather to the number of different schemata that can be generated (see distinction in section 6.2.7).

The second confounding factor is that, between traditional and multi-dimensional, the effects of epistatic bias and dimensional bias cannot be investigated independently. Comparing traditional and multi-dimensional will not reveal which of these is the cause of any observed difference.

To address this question, uniform crossover is introduced. Uniform crossover removes all encoding bias: in this case, both dimensional bias and epistatic bias. Comparing these three crossovers should help distinguish between the effects of dimensional bias and
epistatic bias: if the difference is attributable mainly to exploitation of epistasis, then uniform crossover should be more similar to traditional crossover (both of which exploit epistasis less than multi-dimensional); but if the difference is attributable mainly to dimensional bias, then uniform crossover should be more similar to multi-dimensional crossover (both of which have no dimensional bias).

Unfortunately even this is confounded by the differences in schemata exploration, but the results might still be able to guide conclusions.

### 6.4.2 Evolutionary algorithm

The aim of these experiments is to investigate the effect of encoding/crossover choice on multidimensional problems, isolated as much as is possible from all other factors. The aim is not to outperform any existing algorithm in the literature. It is hoped that any observations may be generalised to apply to many existing or future algorithms.

Therefore a single, basic evolutionary algorithm is used for all experiments, with only the encoding/crossover changing. The evolutionary algorithm is integer-encoded, tournament selection, full replacement, with the following parameters for the different applications:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Fedora function</th>
<th>Train data</th>
<th>Inverted pendulum</th>
</tr>
</thead>
<tbody>
<tr>
<td>population size</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>crossover rate</td>
<td>0.6</td>
<td>0.6</td>
<td>0.6</td>
</tr>
<tr>
<td>mutation magnitude</td>
<td>+/- 3</td>
<td>+/- 10</td>
<td>+/- 5</td>
</tr>
<tr>
<td>mutation rate</td>
<td>0.01</td>
<td>0.01</td>
<td>0.1 - 0.05</td>
</tr>
<tr>
<td>tournament size</td>
<td>4</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>elitism size</td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>generations</td>
<td>5000</td>
<td>7000</td>
<td>2000</td>
</tr>
</tbody>
</table>

As the aim of these experiments is to investigate specific effects and not to produce strictly optimal solutions, no parameter tuning was performed, except to a very limited
extent for the inverted pendulum problem which failed to reliably evolve successful controllers with the same parameters as the function optimisation problems.

For the inverted pendulum, mutation was flipped between 0.1 and 0.05 every 250 generations, beginning with 0.1 at 0 generations and ending with 0.05 for the final 250 generations. This was used to periodically introduce new diversity into the population to avoid premature convergence.

### 6.4.3 Applications

Three test applications were chosen to investigate the encoding of fuzzy logic rulebases: two approximation problems (the fedora function and axle-load data for trains), and a control problem (the Inverted Pendulum problem).

### 6.4.4 Fedora function approximation

Function approximation was chosen as an initial test of dimensional bias. The function to approximate, named “fedora” for its shape, was

\[
f(x_1, x_2) = \frac{\sin(10x_1^2 + 5x_2^2 + 6x_2)}{(10x_1^2 + 5x_2^2 + 6x_2)}
\]

![Figure 6.10. “Fedora” function.](image)

The fuzzy logic knowledge-base had two inputs and one output. Each input was covered by 11 membership sets, and the output was covered by 24 membership sets. The sets were defined manually, and are not expected to be optimal (but producing a strictly
optimal function approximator was not the aim of this experiment). This resulted in a 2-dimensional $11 \times 11$ structure yielding 121 rules.

The test function was sampled at intervals of 0.1 in the range $[-1,1]$ along both dimensions, and the inverse of the root mean square error over those 441 points was used as the fitness of the fuzzy logic approximator.

### 6.4.4.1 Results

The experiment consisted of 5 runs of the evolutionary algorithm for each of traditional crossover, fixed multi-dimensional crossover, random multi-dimensional crossover, and uniform crossover. Figure 6.11 shows the progression of the mean fitness of the best solutions over 1000 generations. Table 6.3 shows the mean best fitness at 200 generations and at the conclusion of the run.

![Figure 6.11. Fedora function – Fitness of best solution – mean of 5 runs per line.](image)
Table 6.3. Fedora function – Fitness of best solution – mean of 5 runs per column. Best in bold.

<table>
<thead>
<tr>
<th>Generations</th>
<th>Traditional</th>
<th>Fixed multi-dimen</th>
<th>Random multi-dimen</th>
<th>Uniform</th>
</tr>
</thead>
<tbody>
<tr>
<td>mean @ 200</td>
<td>0.230655</td>
<td>0.20792</td>
<td>0.33437</td>
<td>0.47269</td>
</tr>
<tr>
<td>stdv @ 200</td>
<td>0.02</td>
<td>0.04</td>
<td>0.04</td>
<td>0.01</td>
</tr>
<tr>
<td>mean @ 5000</td>
<td>0.548294</td>
<td>0.54828</td>
<td>0.548372</td>
<td>0.547296</td>
</tr>
</tbody>
</table>

Table 6.4. Analysis of variance P values, at 200 generations. Significant in bold.

<table>
<thead>
<tr>
<th></th>
<th>Fixed multi</th>
<th>Random multi</th>
<th>Uniform</th>
</tr>
</thead>
<tbody>
<tr>
<td>Traditional</td>
<td>0.558</td>
<td>&lt; 0.001</td>
<td>&lt; 0.001</td>
</tr>
<tr>
<td>Fixed multi</td>
<td>&lt; 0.001</td>
<td>&lt; 0.001</td>
<td>&lt; 0.001</td>
</tr>
<tr>
<td>Random multi</td>
<td></td>
<td></td>
<td>&lt; 0.001</td>
</tr>
</tbody>
</table>

An analysis of variance (one-way ANOVA with Tukey’s HSD) was performed at 200 generations. Generation 200 was chosen as representative of the evolutionary search: a stage between the random initial population and the final converged population, when all runs are still converging. ANOVA found that:

- **Traditional** crossover and **fixed multi-dimensional** crossover were not significantly different from one another (P = 0.558).

- **Random multi-dimensional** crossover was significantly better than traditional and fixed-multi-dimensional (both P < 0.001).

- **Uniform** crossover was significantly better than all other crossovers (all P < 0.001).

After the 5,000 generations allocated to each run, all runs had converged to insignificantly different final solutions.

### 6.4.5 Train data approximation

The second test was to learn a set of 961 data points predicting train wagon axle unloading, sourced from the Centre for Railway Engineering at Central Queensland University.
The test used a data set with two state variables as input, and the predicted axle unloading as output. Each input was covered by 17 membership sets, and the output was covered by 71 membership sets. As with the fedora function approximation, membership sets were not optimised. This resulted in a 2-dimensional 17×17 structure yielding 289 rules. The fitness of the solution was the inverse of the root mean square error over all 961 data points.

6.4.5.1 Results
The experiment consisted of 5 runs of the evolutionary algorithm for each of traditional crossover, fixed multi-dimensional crossover, random multi-dimensional crossover, and uniform crossover. Figure 6.12 shows the progression of the mean fitness of the best solutions over 4000 generations. Table 6.5 shows the mean best fitness at 500 generations and at the conclusion of the run.

![Figure 6.12. Train data – Fitness of best solution – mean of 5 runs per line.](image)

123
Table 6.5. Train data – Fitness of best solution – mean of 5 runs per column. Best in bold.

<table>
<thead>
<tr>
<th>Generations</th>
<th>Traditional</th>
<th>Fixed multi-dimensional</th>
<th>Random multi-dimensional</th>
<th>Uniform</th>
</tr>
</thead>
<tbody>
<tr>
<td>mean @ 500</td>
<td>0.901326</td>
<td>0.831268</td>
<td>1.15031</td>
<td>1.64732</td>
</tr>
<tr>
<td>stdv @ 500</td>
<td>0.10</td>
<td>0.10</td>
<td>0.03</td>
<td>0.02</td>
</tr>
<tr>
<td>mean @ 7000</td>
<td>2.63562</td>
<td>2.63517</td>
<td>2.63164</td>
<td>2.62119</td>
</tr>
</tbody>
</table>

Table 6.6. Analysis of variance P values, at 500 generations. Significant in bold.

<table>
<thead>
<tr>
<th></th>
<th>Fixed multi</th>
<th>Random multi</th>
<th>Uniform</th>
</tr>
</thead>
<tbody>
<tr>
<td>Traditional</td>
<td>0.486</td>
<td>&lt; 0.001</td>
<td>&lt; 0.001</td>
</tr>
<tr>
<td>Fixed multi</td>
<td>&lt; 0.001</td>
<td>&lt; 0.001</td>
<td></td>
</tr>
<tr>
<td>Random multi</td>
<td></td>
<td>&lt; 0.001</td>
<td></td>
</tr>
</tbody>
</table>

Similarly to the fedora function, an analysis of variance at 500 generations found that:

- **Traditional** crossover and **fixed multi-dimensional** crossover were not significantly different from one another (P = 0.486).

- **Random multi-dimensional** crossover was significantly better than traditional and fixed-multi-dimensional (both P < 0.001).

- **Uniform** crossover was significantly better than all other crossovers (all P < 0.001).

After the 7,000 generations allocated to each run, all runs were still converging but there was no significant difference between them.

### 6.4.6 Inverted pendulum control

The inverted pendulum problem was used for the control application, as described in chapter 5. The inverted pendulum problem is a simulation with 4 state variables: the position of a cart on a railway, \( x_1 \); the velocity of the cart, \( x_2 \); the angle of a rigid pole balanced atop the cart, \( x_3 \); and the angular velocity of the pole, \( x_4 \). The controller produces one output, \( u \), which is a force to modify the cart’s velocity (\( x_2 \)). The controller
is to bring all four state variables to 0 – that is, to balance the pole upright at the centre point of the railway.

Each state variable was covered by 5 membership sets, and the output force was covered by 7 membership sets. As with the approximation problems, the membership sets were not optimised.

In this experiment, the rulebase covered all state variables \( x_1, x_2, x_3, x_4 \) with 5 membership sets each. This resulted in a 4-dimensional \( 5 \times 5 \times 5 \times 5 \) structure yielding 625 rules.

Conceptually, a fuzzy logic controller’s rulebase can be mapped onto state-space: at any given point in state-space, the rule(s) covering that point will be used to determine the controller’s behaviour. Unlike the fuzzy logic approximators, which were tested on uniformly distributed samples, different fuzzy logic controllers will move through different areas of state-space, and will therefore make use of different rules.

The fitness of each solution was evaluated by running the controller it represented through 16 test-cases. The objective was always to bring all state variables to 0, but from a different starting position in state-space for each test-case. Two starting positions, “easy” and “hard”, were chosen for each of the 4 state variables, resulting in 16 combinations of starting positions.

Details of the inverted pendulum problem are given in chapter 5.

**6.4.6.1 Results**

The experiment consisted of 5 runs of the evolutionary algorithm for each of traditional crossover, fixed multi-dimensional crossover, random multi-dimensional crossover, and uniform crossover. The fitness was (the inverse of) a weighted aggregation of running sums of each of the state variables. The faster the state variables were brought to 0, the
higher the fitness. Figure 6.13 shows the progression of the mean fitness of the best solution over 1000 generations. Table 6.7 shows the mean best fitness at 200 generations and at the conclusion of the run.

![Inverted Pendulum – Fitness of best solution – mean of 5 runs per line.](image)

**Figure 6.13. Inverted Pendulum – Fitness of best solution – mean of 5 runs per line.**

**Table 6.7. Fitness of best individual – mean of 5 runs per column. Best in bold.**

<table>
<thead>
<tr>
<th>Generations</th>
<th>Traditional</th>
<th>Fixed multidimensional</th>
<th>Random multidimensional</th>
<th>Uniform</th>
</tr>
</thead>
<tbody>
<tr>
<td>mean @ 200</td>
<td>0.00077661</td>
<td>0.00076656</td>
<td>0.00066247</td>
<td>0.00082893</td>
</tr>
<tr>
<td>stdv @ 200</td>
<td>0.00022</td>
<td>0.00014</td>
<td>0.00012</td>
<td>0.00010</td>
</tr>
<tr>
<td>mean @ 2000</td>
<td>0.00123981</td>
<td>0.00123851</td>
<td>0.00123534</td>
<td>0.00123778</td>
</tr>
</tbody>
</table>

**Table 6.8. Analysis of variance P values, at 200 generations. Significant in bold.**

<table>
<thead>
<tr>
<th></th>
<th>Fixed multi</th>
<th>Random multi</th>
<th>Uniform</th>
</tr>
</thead>
<tbody>
<tr>
<td>Traditional</td>
<td>0.999</td>
<td>0.652</td>
<td>0.949</td>
</tr>
<tr>
<td>Fixed multi</td>
<td>0.713</td>
<td>0.917</td>
<td></td>
</tr>
<tr>
<td>Random multi</td>
<td></td>
<td></td>
<td>0.351</td>
</tr>
</tbody>
</table>

Figure 6.13 shows (amongst other things) the mutation level being flipped for the first time at generation 250. By generation 500 the population had nearly converged to the
final optimum, and subsequent changes to mutation level every 250 generations had no effect.

ANOVA was performed at 200 generations, which is one of the widest gaps between the means, but even here there were no statistically significant differences between any means. Even the widest gap between means, seen between random multi-dimensional crossover and uniform crossover, was statistically insignificant with $P = 0.351$.

The differences seen in the graph are accounted for by the high natural variability between individual runs; variability is much higher in this control problem than in the approximation problems above.

The solutions found at the end of each run were subjectively inspected by running the controller through all test cases and examining the behaviour over all four state variables. All solutions performed indistinguishably well on all 16 test cases (the solutions all satisfactorily controlled 13 tests, and all failed on the same 3 tests).

6.5 Conclusions

6.5.1 Approximation

Both approximation problems returned very similar results.

Comparing traditional crossover to random multi-dimensional crossover shows that random multi-dimensional crossover significantly improves upon traditional crossover. However, that comparison alone does not show whether the improvement is attributable to increased epistatic exploitation, decreased dimensional bias, or decreased schemata exploration.

Fixed-multidimensional crossover was not significantly different from traditional crossover, but was significantly different from random multi-dimensional crossover.
Fixed-multidimensional crossover shares a high dimensional bias and low epistatic exploitation with traditional crossover, but shares a low schemata exploration with random multi-dimensional crossover. The conclusion is that the improvement observed in random multi-dimensional crossover is attributable to high exploitation of epistasis and/or absence of dimensional bias.

Uniform crossover significantly improved upon both traditional crossover and random multi-dimensional crossover. Uniform crossover, which has no encoding bias at all, shares a low epistatic exploitation with traditional crossover, and Shares an absence of dimension bias with random multi-dimensional crossover. The conclusion is that the improvement observed in random multi-dimensional crossover is attributable to the absence of dimensional bias.

This conclusion is consistent with the discussion of epistatic bias in fuzzy rulebases used for approximation: epistasis will be confined locally to overlapping neighbouring rules, as the error in approximation of one data point (i.e., the contribution to the fitness evaluation by one decision-variable) does not affect the error in approximation (i.e., contribution) of another data point. With limited scope for epistatic exploitation, uniform crossover is not significantly hindered by its inability to exploit epistasis. However, both uniform crossover and random multi-dimensional crossover benefit from the absence of dimensional bias.

Finally, it is interesting to question why uniform crossover significantly improved upon random multi-dimensional crossover. One explanation lies in the very different schemata exploration available to uniform crossover but not to any of traditional, fixed multi-
dimensional, or random multi-dimensional crossover: Uniform crossover can generate any possible schemata.

6.5.2 Control
The control problem promised to be interesting due to the increased scope for epistatic exploitation, as the contribution of any one rule’s behaviour to the overall fitness of the controller depends heavily on the behaviour of many other rules. Unfortunately this experiment failed to show a significant difference between any of the encodings and crossovers, presumably due to the high variability between runs overshadowing any effects of epistatic exploitation or dimensional bias.

One might tentatively hypothesise that the increased scope for epistatic exploitation in this problem meant that uniform crossover was hindered by its inability to exploit epistasis, and that this contributed to uniform crossover’s inability to distinguish itself from random multi-dimensional crossover. However, this does not explain why random multi-dimensional crossover was unable to distinguish itself from fixed multi-dimensional and traditional crossover.

Another consideration is the degree of usage of the rulebase. At the end of a run, the best solution was run through all 16 test cases again, and the total amount that each rule contributed to the overall control was recorded. If a rule contributed a value greater than 1.0 it was deemed to have a significant effect, as the Standard Additive Model inference engine assigns a membership of 1.0 to a rule if it exactly matches an input. A total of 1.0 or greater therefore means that the rule either exactly matched input at least once, or it partially matched input multiple times adding up to 1.0 or greater in total.
Out of the 625 rules in the 4-dimensional rulebase, a typical solution would have 70 rules (11%) that contributed 1.0 or greater to the product inference engine. Only 138 rules (22%) would contribute any amount at all.

In the inverted pendulum problem the goal, measured as a running sum in the fitness evaluation, is to bring the state variables to 0 as quickly as possible, which can be accomplished by using as few rules as possible other than those that cover (0,0,0,0) in state-space. Therefore, the evolutionary algorithm is expected to produce a rulebase that uses rules as efficiently as possible. Unfortunately for conclusions on encoding and crossover, the relatively small percentage of decision-variables being utilised may be one reason why the different types of encoding and crossover perform indistinguishably well in this case.

### 6.5.3 Overall

The most appropriate encoding for a fully specified fuzzy rulebase is a multidimensional structure. Such an encoding enables the exploitation of known epistasis whilst avoiding dimensional bias. Both multi-dimensional and uniform crossovers are appropriate for this encoding. The results suggest that uniform crossover is more suitable for function approximation problems with limited scope for epistasis exploitation. Multi-dimensional crossover, particularly more sophisticated forms such as in [54] may conceivably be more suitable for more epistatic problems, such as control problems.

From an entirely pragmatic standpoint, both uniform and multi-dimensional crossover remove any difficulty the system designer might have in deciding how to order dimensions when mapping to a single-dimensional vector-based encoding.
It is expected that multidimensional encodings will never have a negative effect, because a multidimensional encoding using multi-dimensional crossover is equivalent to the common single-dimensional encoding, assuming a different arbitrary mapping from n dimensions to one dimension.

Ultimately, the conclusion from this work is that the encoding (and crossover) should be designed with consideration to what can be known about the problem domain; the designer should question what assumptions are reasonable and be cautious of unsubstantiated, arbitrary design decisions. Better to adapt the algorithm to the problem, than the problem to the algorithm.

6.5.4 Publications

Part of this research was published in [110].

Its conclusions were subsequently adopted in [92, 96, 113].
Chapter 7. Cooperative Coevolution

7.1 Introduction

This chapter investigates the application of cooperative coevolution to hierarchical rulebases (see sections 3.4 and 2.7 respectively). Hierarchical rulebases and cooperative coevolution have similar goals and requirements: specifically, they both mitigate the “curse of dimensionality” by breaking down the problem, and they both require consideration of interdependence, or epistasis, between variables.

One of the main advantages that cooperative coevolution claims is faster convergence due to a smaller search-space. Another advantage is the coarse-grained parallelism possible due to the use of multiple populations (see section 3.3). Given that two of the advantages of cooperative coevolution are about speedup, this chapter also examines the issues involved in implementing cooperative coevolution as parallel code.

In this chapter alone, a distinction is made between an individual and a solution, though the usage remains compatible with other chapters that do not make this distinction. Previously, “solution” referred to both the conceptual solution represented by decision-variables, and the individual in the evolutionary population represented by an encoding. Here, solution refers only to the conceptual solution represented by decision-variables. Individual is used to refer specifically to an encoded structure in the evolutionary
population. This distinction is necessary because, in cooperative coevolution, a single individual is *not* an entire solution; it is only a partial solution. Cooperative coevolution is examined in section 7.2.2.

### 7.2 Background

#### 7.2.1 Hierarchical rulebases

Chapter 2 introduced the use and design of hierarchical rulebases. The topology should be designed such that the interdependence imposed upon state-variables corresponds to the natural interdependence between state-variables in the problem description, and therefore the natural interdependence – also known as epistasis – of the decision-variables. Unfortunately there is no standard methodology for designing an appropriate topology. Section 7.4.2 describes a problem-specific topology design for the inverted pendulum problem.

There is also no standard methodology for designing the fuzzy sets covering the intermediate variables. However, some general observations can be made that may guide the design (which, for the inverted pendulum problem, is described in section 7.4.3).

For the purposes of this discussion, assume the rightmost topology in figure 7.1.

![Figure 7.1. Some different possible topologies for a 4-input fuzzy system.](image)

The intermediate output is only used as input into rulebase B, and only affects the matching strength of rules in rulebase B. Because the rulebase is fully specified, there is a separate rule for every intermediate fuzzy set combined with every other combination of
input fuzzy sets. Therefore, the actual value of the intermediate output can be interpreted somewhat arbitrarily simply by choice of consequents in rulebase B.

That said, an efficient rulebase interpretation should preserve the continuous nature of the intermediate output, because a continuous interpretation is more compatible with overlapping input fuzzy sets (which will cause rule matching strength to be correlated between rules with neighbouring fuzzy sets).

However, the orientation of the interpretation – from low-to-high or from high-to-low – should be entirely arbitrary, if the membership functions are symmetrical on either side of the mean value.

Although the absolute range and orientation of the intermediate output is meaningless, the relative distribution and shape of membership functions inside that range is meaningful, as the relative distribution affects the matching strength of rules – particularly rules with neighbouring intermediate fuzzy sets – in rulebase B.

For example, it is possible to symmetrically widen any intermediate fuzzy set without changing its centroid (and therefore without changing the defuzzified intermediate output), but this would cause the matching strength of rules in rulebase B to change (therefore changing the final output).

In summary, the absolute values of the intermediate fuzzy sets are arbitrary, but the distribution and shape of the membership functions must be considered.
7.2.2 Cooperative Coevolution
Chapter 3 introduced cooperative coevolution. This section gives details of the algorithm used in this work.

Each population is the same size and evolves for the same number of generations.

7.2.2.1 Representative selection
Chapter 3 identified the effect representative selection has on the apportionment of credit problem. As in [34, 76], this work uses exploitative and explorative representative selection, implemented in two algorithms:

1. **Greedy**: The representative manager selects the best individual from each population to be the sole exploitative representative of that population. When an individual is evaluated, it is combined with the representatives to form a single greedy solution. The solution is evaluated and its fitness score assigned to the individual.

2. **Explorative**: The representative manager selects the best individual and a random individual from each population to be the two exploitative and explorative
representatives of that population. When an individual is evaluated, it is combined with the exploitative representatives to form a greedy solution, and with the explorative representatives to form an explorative solution. Both solutions are evaluated, and the better fitness score is assigned to the individual.

In both algorithms, the representative manager updates the representatives in between every generation. All EAs advance generations synchronously; each EA waits for all other EAs to complete the current generation before advancing to the next. This addresses two potential issues with cooperative coevolution:

1. The same representatives are used to evaluate all individuals in a population; the representatives do not change partway through a generation, which would bias the apportionment of credit. This could also be addressed by each EA taking a local copy of the representatives at the beginning of each generation; however that does not address the second issue:

2. If EAs are allowed to evolve asynchronously then results depend on the implementation of program and hardware, because the speed of the implementation will affect which generation an EA is at, and therefore which representative is selected, at any given point in time relative to the other EAs. There is no theoretical reason why this should be a problem, but it makes it impossible to reproduce results for the purpose of verification. This is discussed further in section 7.5.1.
7.2.2.2 Hypotheses

There are two proposed advantages to cooperative coevolution:

1. Better solutions due to a more modular search.

2. Faster progress due to a reduced total search space.

The potential disadvantage to a more modular search is the effect of epistasis between modules. If there is a high epistasis between modules, the use of a small number of representatives may limit the exploratory power of the algorithm, causing it to more easily become stuck in local optima and ultimately limiting the final solution quality.

In [76], greedy and explorative representative selection was applied to function optimisation, and it was found that over 5 test functions, greedy representative selection was very good on functions with low epistasis, but very poor (worse than a traditional EA) on a function with high epistasis. The use of explorative representative selection allowed the coevolutionary EA to perform no worse than a traditional EA on the highly epistatic problem. On the functions with low epistasis, the coevolutionary EAs made faster progress over the duration of the runs, and found better quality solutions at the end of the runs. Unfortunately, it is unclear whether the solutions were better quality due to the greater modularity (1) or due to the faster progress (2), because the fitness curve of the traditional EA showed no sign of plateauing (convergence) by the end of the runs – it is conceivable that the traditional EA may have caught up to the coevolutionary EAs given enough generations.
In [34], the effect of different decompositions with varying degrees of epistasis between modules was investigated. It tested all combinations of high- and low-epistasis decompositions with greedy and explorative representative selection (total of 4 algorithms) on both a high- and a low-epistasis version of the same inventory control problem (total of 8 experiments). As with [76], on the low-epistasis problem, all coevolutionary EAs outperformed the traditional EA, and on the high-epistasis problem the greedy, high-epistasis EA performed poorly but the explorative and low-epistasis decomposition EAs outperformed the traditional EA and the greedy, high-epistasis EA. When the coevolutionary algorithms outperformed the traditional EA, it was by better final solutions and faster convergence, but again unfortunately it is unclear whether the better final solutions were due to greater modularity (1) or due to faster convergence (2), because again the traditional EA had not converged by the end of the runs.

In [77], dynamic decomposition using Cascade Neural Networks was applied to a function approximation problem. As above, the coevolutionary EA made faster progress and found better quality final solutions, but again as above the traditional EA had not converged by the end of the runs.

In [78], coevolution was applied to learning robot control rules. It was found that the coevolutionary EA made faster progress to a predefined fitness level, but that the coevolutionary EA and the traditional EA converged to final solutions of the same fitness at the end of the runs. Interestingly, [78] reported that the coevolutionary EA produced qualitatively modular final solutions (by examining the rulebases), although this did not result in a better overall fitness.
In [104] another dynamic decomposition was applied to function optimisation, with similar results: when a high-epistasis decomposition is applied to a high-epistasis problem, the coevolutionary EA performs worse than a traditional EA; otherwise coevolutionary EAs perform equal to or better than a traditional EA. However, again, in all cases where the coevolutionary EAs outperformed the traditional EA by final solution quality, the traditional EA had not converged. In all cases where the traditional EA had converged, it achieved an equal final solution quality to the coevolutionary EAs (though the coevolutionary EAs may have made faster progress over the duration of the run).

There are some common observations from all these results that suggest the following hypotheses:

1. Greedy representative selection, combined with high-epistasis decompositions, applied to high-epistasis problems, causes coevolutionary EAs to easily become stuck in local optima, ultimately performing worse than a traditional EA.

2. Both explorative representative selection and low-epistasis decompositions mitigate the coevolutionary EA’s disadvantage on high-epistasis problems, performing equal to or better than a traditional EA.

3. Coevolutionary EAs do not necessarily converge to better quality final solutions than a traditional EA; if they do it is due to faster progress over a limited duration rather than due to more modular final solutions.

The experiments in section 7.5 attempt to provide further evidence to investigate these hypotheses.
7.2.3 Parallelism
Chapter 3 introduced approaches to parallelising EAs, and described ways to structure the population to support parallelism; specifically the cellular model and the island model. Both models change the essential evolutionary algorithm in addition to supporting parallelism – by structuring the population they encourage niche formation and can help avoid premature convergence.

Another EA that changes the essential algorithm with the aim of improving the evolutionary search is the cooperative coevolutionary algorithm, the focus of this chapter. It may be parallelised similarly to the island model, but its purpose is quite different. Instead of the island model’s competing subpopulations of whole solutions, the cooperative coevolutionary algorithm uses cooperative subpopulations of partial solutions. The algorithm is designed to exploit modularity, not maintain niches, but it happens to be similarly parallelisable.

Cooperative coevolution, similarly to the island model, supports coarse-grained parallelism at the population level. Each population can be evolved by a separate process, with minimal communication between the population and the representative manager responsible for storing the current representative from each population (and therefore the final complete solution).

Similarly to the master-slave model and the island model, the parallelism can be synchronous or asynchronous. The cooperative coevolutionary algorithm is synchronous; every population completes one generation before they all move on to the next. Therefore synchronous parallelism does not affect the outcome of the algorithm. Unfortunately, blocking becomes a potential problem – arguably more so in cooperative coevolution due
to the disparate populations evolving different partial solutions. Asynchronous parallelism solves the blocking problem, but changes the essential algorithm.

Figure 7.3 shows the scope for parallelism in a cooperative coevolutionary algorithm. Coarse-grained, asynchronous parallelism can be applied to different populations, similar to the island model. Fine-grained, synchronous parallelism can be applied to individuals within each population, using the master-slave model (but not the cellular model unless explicitly desired, as the cellular model will change the algorithm). Finally, in some rare problems the fitness function itself may be parallelisable.

Here we examine the coarse-grained asynchronous parallelism of populations. The synchronous master-slave model for individuals has previously been described adequately; the synchronous parallelism of populations as described above is trivial; and parallelising the fitness function is problem specific.

There are two ways of implementing the asynchronous communication between populations and the representative manager. In both cases, a population updates the
representative manager if the population discovers a new representative at the end of a generation. After that, either:

a) the population processes can request the current representatives at the beginning of each generation; or

b) the representative manager can push out the current representatives whenever they change.

Neither method changes the algorithm; in both cases each population will use the current representatives at the beginning of each generation, for that entire generation.

Method (b) sends provably less data. Assume $N$ populations each completing $G$ generations at any arbitrary speeds. Assume that under both methods, a population only updates the representative manager at the end of a generation in which that population’s representative changes. Representative transfers from population to manager are therefore the same in both methods, and will be ignored.

Under method (a), representatives will always be transferred $N(G(N-1))$ times: every population, at every generation, requests $N-1$ representatives from the manager.

Now assume the worst case: that every population’s representative changes every generation. Under method (b), representatives will also be transferred $N(G(N-1))$ times: every population, at every generation, discovers a new representative that is pushed to the other $N-1$ populations. However, this is only an upper bound; in reality not every population will discover a new representative at every generation, and the total number of transfers will be less.

In practice, though, method (a) is easier to implement, as the population processes do not need to listen for updates from the representative manager, which may come at any time.
Also, because method (a) requests $N$-1 representatives all together at the beginning of each generation, all $N$-1 representatives can be sent as one message resulting in just $N(G(1))$ messages. In contrast, method (b) cannot pack multiple representatives into a single message, due to the asynchronous pushes, and (b) therefore does not send provably fewer messages than (a). Method (b) still sends provably less data, but in message-passing architectures the overhead to set up a single message can be far greater than the cost-per-byte to send the data inside it. This effect is seen in section 7.5.1.

Finally, in explorative versions of the cooperative coevolutionary algorithms, the representatives may be chosen randomly rather than deterministically, making it more likely that the representatives will change every generation, and mitigating the data advantage that method (b) has over method (a).

### 7.3 Motivation for research

There are certain similarities between hierarchical rulebases and cooperative coevolution that suggest these two techniques are very naturally suited to one another.

- Both techniques are used to mitigate the curse of dimensionality, and in similar ways: hierarchical rulebases partition a high-dimensional input-space, and cooperative coevolution decomposes a high-dimensional solution-space, into loosely coupled modules.

- Both techniques should ideally minimise the interdependence between modules: hierarchical rulebases to minimise the impact of the topology on the expressiveness of the system, and cooperative coevolution to minimise the effect of epistasis on the apportionment of credit problem.
There is no established method for determining the best topology for a hierarchical rulebase, or for determining the best decomposition for cooperative coevolution. However, the above observations suggest that the design goals and requirements are closely matched, and that the problem-dependent design of one can guide the design of the other. Specifically, the individual rulebases in a hierarchical topology can correspond directly to modules in a cooperative coevolutionary decomposition.

The use of cooperative coevolution also neatly solves a minor difficulty in combining hierarchical rulebases and multidimensional crossover (as in chapter 6): how should multiple rulebases be encoded in a single structure without bias? If each rulebase is a separate individual, the encoding of each individual is still a single multidimensional rulebase.

### 7.4 Application

#### 7.4.1 Inverted pendulum problem

This chapter uses a slightly simplified version of the inverted pendulum problem to that described in chapter 5. The only differences are:

1. Only one initial test-case is used: $x_1 = 0.5; x_2 = 0.0; x_3 = 0.01; x_4 = 0.0$

2. The control output is restricted to [-10,10], and is covered by just 7 fuzzy sets, the centroids of which are {-10, -4.5, -2.5, 0, 2.5, 4.5, 10}.

![Figure 7.4. Centroids of membership functions for output $y$.](image-url)
These restrictions were made to facilitate the hierarchical design, particularly the intermediate output, described in section 7.4.3. This chapter is principally a study of cooperative coevolution, and these changes simplify the investigation.

The equations governing the state variables are unchanged, but are restated here for reference:

$$\dot{x}_1 = x_2$$  \hspace{1cm} (7.1)

$$\dot{x}_2 = u + ml(\sin(x_3)x_4^2 - \dot{x}_4 \cos(x_3))/(M + m)$$ \hspace{1cm} (7.2)

$$\dot{x}_3 = x_4$$ \hspace{1cm} (7.3)

$$\dot{x}_4 = \frac{g \sin(x_3) + \cos(x_3)(u - mlx_4^2 \sin(x_3))/(M + m)}{l\left(\frac{x_3^2 - m \cos(x_3)^2}{(M + m)}\right)}$$ \hspace{1cm} (7.4)

### 7.4.2 Topology

A two-rulebase hierarchical topology for the inverted pendulum problem was previously used in [95]. In that topology, shown in figure 7.5, the intermediate output from rulebase A was added to the input $x_3$ before being fed into rulebase B. Whilst it results in small rulebases (5 fuzzy sets per input; $5 \times 5 + 5 \times 5 = 50$ rules total), the limitation of this topology is the inability to distinguish between $x_1$, $x_2$, and $x_3$ in determining an appropriate final output.
The topology used in this study is shown in figure 7.6. Each input is covered by 5 fuzzy sets (as described in chapter 5) and the intermediate output, $y_A$, is covered by 7 fuzzy sets (see section 7.4.3), giving $5 \times 5 + 7 \times 5 \times 5 = 200$ rules total – more than figure 7.5 but still much less than a single rulebase of 625 rules (as in chapter 5).

This topology was chosen according to understanding of the problem description, particularly the system of equations (7.1)-(7.4) governing the state variables (each of which is an input).

First, the inputs are grouped by interdependence: $x_1$ depends only upon $x_2$ (7.1); and $x_3$ depends only upon $x_4$ (7.3). $x_4$ in return depends only upon $x_3$ (7.4). $x_2$ has the most varied dependencies, depending upon $x_3$ and $x_4$ but not $x_1$ (7.2). All things considered, and despite $x_2$’s varied dependencies, it is most natural to group $x_1$ with $x_2$; and $x_3$ with $x_4$. The former group describes the cart; the latter group describes the pole.

Given the above grouping, it was determined that $x_1$ and $x_2$ should feed into rulebase A to produce an intermediate output. $x_3$ and $x_4$ should feed directly into rulebase B, along with the intermediate output, to produce the final output. This is consistent with the weights given in the problem description in chapter 5, in which the greatest weight is given to keeping the pole upright ($x_3$) and the (equal) second greatest weight is given to minimising the pole angular velocity ($x_4$). This topology allows $x_3$ and $x_4$ to directly and independently influence the final output, giving finer control over the pole than the cart.

Other topologies for the inverted pendulum problem were subsequently investigated in [97, 113].
7.4.3 Intermediate output

The intermediate output, $y_A$, is covered by 7 fuzzy sets, but as the Standard Additive Model uses additive interpolation, only the centroid of each membership function is required. The centroids, shown in figure 7.7, are \{-10, -4.5, -2.5, 0, 2.5, 4.5, 10\}.

![Figure 7.7. Centroids of membership functions for output $y_A$.](image)

The input accepting the intermediate output is also covered by 7 fuzzy sets designed to fully cover the range of the intermediate output. For simplicity, as this is not an exercise in tuning fuzzy sets, the membership functions were designed with their peaks corresponding to the centroids of the intermediate output fuzzy sets.

![Figure 7.8. Membership functions covering input $y_A$.](image)

(Note that centroids of the $y_A$ output fuzzy sets are not technically the centroids of the $y_A$ input fuzzy sets. They are more accurately the means-of-maximum (see chapter 2).)

The range of the intermediate fuzzy sets is arbitrary and unimportant. The range $[-10,10]$ was chosen to be comparable to the range of the final output, but the values themselves are meaningless.

In this case, the distribution of intermediate output centroids was designed to be the same as that of the final output centroids, and the shapes of the intermediate input membership
functions were designed to cover each output centroid with full overlap. Although in general the meaning of the intermediate output can be arbitrary, in this topology the intermediate output is anticipated to be an approximation of the control force required to move the cart to the centre and minimise the velocity (although due to the symmetry around 0, it may be reversed). Therefore the intermediate output centroids and membership functions were chosen to provide finer control around 0, just as with the final output centroids.

7.5 Experiments

The following algorithms were chosen for the experiments.

- **High-population traditional** EA with population of 200 individuals.
- **Low-population traditional** EA with population of 100 individuals.
- **Greedy coevolutionary** EA with two populations each of 100 individuals.
- **Explorative coevolutionary** EA with two populations each of 100 individuals.

In the coevolutionary algorithms, each population evolves one of the two rulebases in the hierarchical topology described in section 7.4.2.

The basic aim is to investigate the difference between using a traditional EA and using a cooperative coevolutionary EA to reduce the size of the search space. The hypothesis is that by reducing the search space, a coevolutionary search will make faster progress towards finding a near-optimal solution. Whether the final solution will be of better or worse quality is an open question to be answered: the final solution may be better quality if the faster progress is maintained for the duration of the run; or it may be worse quality if the reduced search space ultimately limits the exploratory power of the EA.
Two parameter variations of the traditional EA are used: one with a large population equivalent to the total number of individuals in both cooperative populations; and one with a small population equivalent to the number of individuals in each cooperative population separately. The high-population EA is intended to have the same total number of trials per generation as the greedy coevolutionary EA. The low-population EA is intended to have the same potential for population diversity, and therefore convergence, as the greedy coevolutionary EA with its evolutionarily distinct small populations. The explorative coevolutionary algorithm is intended to investigate the above question of exploratory power affecting final solution quality.

7.5.1 Parallelism
The following results were obtained using a clustered parallel computer architecture:

- 80 nodes, each a 1GHz Pentium3.
- 100baseT Ethernet communications network.
- Linux operating system.
- openMosix clustering software.
- MPICH message-passing software.

The cluster’s inter-node message-passing and processing times were tested to determine the feasibility of parallelising the cooperative coevolutionary algorithm for the inverted pendulum problem.
Table 7.1. Inter-node send/receive times. Mean of 1,000 iterations.

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Time to send (seconds)</th>
<th>Time to receive (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.002582</td>
<td>0.003278</td>
</tr>
<tr>
<td>200</td>
<td>0.002758</td>
<td>0.003527</td>
</tr>
<tr>
<td>2,000</td>
<td>0.003525</td>
<td>0.004283</td>
</tr>
<tr>
<td>5,000</td>
<td>0.003839</td>
<td>0.004621</td>
</tr>
<tr>
<td>10,000</td>
<td>0.004448</td>
<td>0.005217</td>
</tr>
<tr>
<td>100,000</td>
<td>0.019477</td>
<td>0.020281</td>
</tr>
<tr>
<td>200,000</td>
<td>0.037070</td>
<td>0.037740</td>
</tr>
<tr>
<td>400,000</td>
<td>0.079540</td>
<td>0.080173</td>
</tr>
</tbody>
</table>

Of particular note are the test cases for 200 bytes and 2,000 bytes, as these represent the time taken to send and receive 1 solution and a population of 100 solutions, respectively (each solution to the hierarchical inverted pendulum problem is 200 decision variables each of 1 byte).

The time cost to send (or receive) a message can be described as

\[ t = b \cdot r + o \]  \hspace{1cm} (7.5)

where \( t \) is the total time in seconds, \( b \) is the number of bytes, \( r \) is the transfer rate in bytes/second (Bs\(^{-1}\)), and \( o \) is the overhead. Solving for \( t = 0.002582, b = 1 \) and \( t = 0.079540, b = 400,000 \), approximate values for transfer rate and overhead are:

\[
\begin{align*}
  r & \approx 1.923945e^{-7} \text{ Bs}^{-1} \\
  o & \approx 0.0025818 \text{s}
\end{align*}
\]

which is roughly consistent with the other test cases.

In the overhead time it takes to set up one message, approximately 13,400 bytes can be transferred. As all messages in this application will be 200 bytes (a single solution) or less (eg. a fitness score or partial solution), minimising the number of messages, and therefore overhead, is worthwhile.
In many cases, it will even be worthwhile packing extra anticipated, possibly redundant data in a single message, to avoid multiple request/reply control messages. The cost of packing extra data into a message is negligible. On the above platform, the cost of packing 200 bytes is ~0.000001s, or approximately 3 orders of magnitude less than the overhead of a single message.

Section 7.2.3 discussed how the choice of implementation affects the number of messages and amount of data that the coevolutionary algorithm sends. This discussion suggests that the implementation that minimises the number of messages, even at the cost of additional data, should be preferred.

All of this must be considered in the context of the amount of computation that can be parallelised. In this case, it is the fitness evaluations and evolutionary code. In almost all evolutionary algorithms, fitness evaluations can be distributed to different nodes. In traditional EAs the evolutionary code cannot be distributed, but in coevolutionary EAs it can.

The fitness evaluation, as given in section 5.5, is by far the most expensive computation in this application. Each evaluation is a 1000 time-step simulation of 4 state-variables, with each time-step updated by the fourth-order Runge-Kutta method, plus code to measure the controller’s performance.

On the above platform, a single evaluation costs ~0.1653s (mean of 100 evaluations). The evolutionary code (not including evaluations) costs ~0.0014s (mean of 100 generations). In other words, evaluating a single solution is approximately 2 orders of magnitude more expensive than the cost of sending the solution (or even an entire population) to another
processing node, so the master-slave model of parallelisation is worthwhile. Consider also that this chapter uses only a single test-case; the fitness evaluation cost is directly proportional to the number of test-cases.

This is an example of how “trivially parallel” any evolutionary algorithm with a relatively expensive fitness function is. Even traditional EAs can easily benefit from master-slave parallelism for fitness evaluation, and communications overhead, including blocking in the synchronous master-slave model, will be masked by the fitness evaluation cost. The question of how a coevolutionary EA can improve upon already “trivial” parallelism will be addressed in the conclusions.

Finally, the number of parallel nodes to distribute the computation across must be determined. The optimal number of nodes depends upon communication cost, fitness evaluation cost, and the number of solutions (and/or populations, in coevolutionary models). Adding additional nodes can actually degrade performance when communication overhead (which rises with additional nodes) grows larger than the amount of useful computation performed by each node (which decreases with additional nodes) [10].

The above analysis has shown the feasibility of parallelising the EA for the inverted pendulum problem. Studies can ordinarily omit implementation details, but if the model of parallelism changes the algorithm in such a way that the hardware or software implementation affects the results, then the implementation must be taken into account.
For this reason, this study uses the synchronous coevolutionary model, so that the essential algorithm is unchanged and the results presented here do not depend upon the implementation.

### 7.5.2 Evolutionary algorithm

The aim of these experiments is to investigate some specific questions concerning cooperative coevolution – namely convergence and final solution quality – isolated as much as is possible from all other factors. The aim is not to outperform any existing algorithm in the literature. It is hoped that any observations may be generalised to apply to many existing or future algorithms.

Therefore a single, basic evolutionary algorithm is used for all experiments. The evolutionary algorithm operating on each population in isolation is the same for all the above algorithms; the coevolutionary algorithm simply uses multiple instances of the basic evolutionary algorithm (one per population), and the coevolutionary logic is encapsulated in the fitness evaluation (representatives from all coevolutionary populations must be combined for fitness evaluation).

The EA used tournament selection and full replacement, with the parameters in table 7.2 for the different applications:

<table>
<thead>
<tr>
<th>Table 7.2. Algorithm parameters</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>population size</td>
<td>100</td>
</tr>
<tr>
<td>crossover rate</td>
<td>0.6</td>
</tr>
<tr>
<td>mutation magnitude</td>
<td>+/- 5</td>
</tr>
<tr>
<td>mutation rate</td>
<td>0.7 - 0.01</td>
</tr>
<tr>
<td>tournament size</td>
<td>2</td>
</tr>
<tr>
<td>elitism size</td>
<td>1</td>
</tr>
<tr>
<td>generations</td>
<td>1000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 7.3. Mutation schedule</th>
<th>Generation</th>
<th>Mutation rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generation</td>
<td>0</td>
<td>0.7</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>0.6</td>
</tr>
<tr>
<td></td>
<td>200</td>
<td>0.5</td>
</tr>
<tr>
<td></td>
<td>300</td>
<td>0.2</td>
</tr>
<tr>
<td></td>
<td>400</td>
<td>0.1</td>
</tr>
<tr>
<td></td>
<td>500</td>
<td>0.01</td>
</tr>
</tbody>
</table>

The mutation rate decreased according to the schedule given in table 7.3.
As the aim of these experiments is to investigate specific questions and not to produce strictly optimal solutions, no parameter tuning was performed, except to the very limited extent of finding a parameter set that reliably resulted in successful controllers.

The solution encoding was the multidimensional encoding for fully-specified fuzzy rulebases, as used in chapter 6. Rulebase A was encoded as a 5×5 structure, and rulebase B was encoded as a 7×5×5 structure.

7.5.3 Results

7.5.3.1 By generations
Figure 7.9 shows the progression of the mean fitness of best solutions from 8 runs of each of the algorithms. The results are generally as expected, with the low-population traditional EA making slower progress per generation than the high-population traditional EA. This may be attributed to simply having fewer trials per generation. Both coevolutionary EAs make much faster progress, which may be attributed to the smaller search-space compared to a traditional EA. Of the two coevolutionary EAs, the explorative EA makes faster progress than the greedy EA, which may be attributed to the greater number of trials (one each for greedy and explorative representatives) allocated to the explorative EA.

An analysis of variance (one-way ANOVA with Tukey’s HSD) performed on the final solutions found by each run found that the differences between algorithms were not significantly different (all P > 0.05).
Table 7.4. Mean and standard deviation of final solutions from 8 runs of each algorithm. Best in bold.

<table>
<thead>
<tr>
<th>Generation</th>
<th>High-pop</th>
<th>Low-pop</th>
<th>Greedy</th>
<th>Explorative</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mean</td>
<td>0.47126</td>
<td>0.50670</td>
<td>0.48028</td>
<td>0.37895</td>
</tr>
<tr>
<td>stdv</td>
<td>0.15431</td>
<td>0.24114</td>
<td>0.21111</td>
<td>0.16326</td>
</tr>
</tbody>
</table>

Table 7.5. Analysis of variance P values, for final solutions. Significant in bold.

<table>
<thead>
<tr>
<th>Low-pop</th>
<th>Greedy</th>
<th>Explorative</th>
</tr>
</thead>
<tbody>
<tr>
<td>High-pop</td>
<td>0.984</td>
<td>0.999</td>
</tr>
<tr>
<td>Low-pop</td>
<td></td>
<td>0.993</td>
</tr>
<tr>
<td>Greedy</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

However, given the differences between the algorithms it may not be appropriate to compare by generation. The low-population traditional EA performs 100 trials per generation. The high-population traditional EA and greedy coevolutionary EA perform 200 trials per generation. The exploratory coevolutionary EA performs 400 trials per generation.
By evaluations

Figure 7.10 shows the algorithms’ performances plotted by cumulative number of fitness evaluations rather than the usual generations. This comparison reverses the apparent standing of greedy vs. explorative coevolution and low- vs. high-population traditional algorithms.

The greedy coevolutionary EA now makes faster progress than the explorative EA, which may be attributed to it committing to trial solutions more readily. The low-population traditional EA now makes faster progress than the high-population EA, which may be attributed to it having performed double the number of evolutionary generations for a given number of fitness evaluations (the same applies to greedy vs. explorative).
Table 7.6. Mean and standard deviation of best solutions from 8 runs of each algorithm. Best in bold.

<table>
<thead>
<tr>
<th>Evaluations</th>
<th>High-pop</th>
<th>Low-pop</th>
<th>Greedy</th>
<th>Explorative</th>
</tr>
</thead>
<tbody>
<tr>
<td>25,000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mean</td>
<td>2.35579</td>
<td>1.96382</td>
<td><strong>0.90836</strong></td>
<td>0.95245</td>
</tr>
<tr>
<td>stdv</td>
<td>0.20503</td>
<td>0.65610</td>
<td>0.40707</td>
<td>0.22560</td>
</tr>
<tr>
<td>50,000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mean</td>
<td>1.86142</td>
<td><strong>0.64221</strong></td>
<td>0.83971</td>
<td>0.87585</td>
</tr>
<tr>
<td>stdv</td>
<td>0.61076</td>
<td>0.18905</td>
<td>0.39546</td>
<td>0.22933</td>
</tr>
<tr>
<td>100,000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mean</td>
<td>0.76699</td>
<td>0.50670</td>
<td><strong>0.50099</strong></td>
<td>0.81898</td>
</tr>
<tr>
<td>stdv</td>
<td><strong>0.12461</strong></td>
<td>0.24114</td>
<td>0.20849</td>
<td>0.24758</td>
</tr>
</tbody>
</table>

Table 7.7. Analysis of variance P values, at 25,000 evaluations. Significant in bold.

<table>
<thead>
<tr>
<th>25,000</th>
<th>Low-pop</th>
<th>Greedy</th>
<th>Explorative</th>
</tr>
</thead>
<tbody>
<tr>
<td>High-pop</td>
<td>0.247</td>
<td>&lt; <strong>0.001</strong></td>
<td>&lt; <strong>0.001</strong></td>
</tr>
<tr>
<td>Low-pop</td>
<td>&lt; <strong>0.001</strong></td>
<td>&lt; <strong>0.001</strong></td>
<td></td>
</tr>
<tr>
<td>Greedy</td>
<td></td>
<td></td>
<td>0.996</td>
</tr>
</tbody>
</table>

Table 7.8. Analysis of variance P values, at 50,000 evaluations. Significant in bold.

<table>
<thead>
<tr>
<th>50,000</th>
<th>Low-pop</th>
<th>Greedy</th>
<th>Explorative</th>
</tr>
</thead>
<tbody>
<tr>
<td>High-pop</td>
<td>&lt; <strong>0.001</strong></td>
<td>&lt; <strong>0.001</strong></td>
<td>&lt; <strong>0.001</strong></td>
</tr>
<tr>
<td>Low-pop</td>
<td>0.748</td>
<td>0.639</td>
<td></td>
</tr>
<tr>
<td>Greedy</td>
<td></td>
<td></td>
<td>0.998</td>
</tr>
</tbody>
</table>

Table 7.9. Analysis of variance P values, at 100,000 evaluations. Significant in bold.

<table>
<thead>
<tr>
<th>100,000</th>
<th>Low-pop</th>
<th>Greedy</th>
<th>Explorative</th>
</tr>
</thead>
<tbody>
<tr>
<td>High-pop</td>
<td>0.082</td>
<td>0.073</td>
<td>0.960</td>
</tr>
<tr>
<td>Low-pop</td>
<td>0.999</td>
<td><strong>0.026</strong></td>
<td></td>
</tr>
<tr>
<td>Greedy</td>
<td></td>
<td></td>
<td><strong>0.023</strong></td>
</tr>
</tbody>
</table>

An analysis of variance was performed at 25,000, 50,000, and 100,000 evaluations. These locations were chosen as the three interesting stages of evolution of the algorithms relative to one another: at 25,000 evaluations the coevolutionary EAs are approaching convergence far sooner than the traditional algorithms; at 50,000 evaluations the low-population traditional EA has surpassed the coevolutionary EAs; and at 100,000 evaluations the final solutions are compared.
At 25,000 evaluations:

- Both traditional EAs were not significantly different from one another (P = 0.247).
- Both coevolutionary EAs were not significantly different from one another (P = 0.996).
- Both coevolutionary EAs were significantly better than both traditional EAs (all P < 0.001).

At 50,000 evaluations:

- Both coevolutionary EAs were not significantly different from one another (P = 0.998).
- High-population traditional was significantly worse than all other algorithms (all P < 0.001).

At 100,000 evaluations:

- Both greedy coevolution (P = 0.023) and low-population traditional (P = 0.026) were significantly better than explorative coevolution. (and both were almost significantly better than high-population traditional).
7.5.3.3 Examination of solutions
The convergence of decision variables in the final populations was examined. A decision-variable was held to have converged if the 50 best solutions in the population all have the same value for that decision-variable.

For all runs, the final populations showed the same pattern of convergence for rulebase A. That is, it was always the same set of decision variables that converged, and the same set that did not. As each decision variable encodes a single rule, this pattern of convergence reflects the selection pressure placed on specific rules.

The rules that had the highest selection pressure (those whose decision variables had converged) were those where the $x_1$ antecedent was one of the last 4 fuzzy sets, and where the $x_2$ antecedent was one of the central 3 fuzzy sets (see bold consequents in table 7.10). We infer that the controller did not spend much (if any) time with $x_1$ very low, or with $x_2$ very low or very high, because there was little selection pressure on the rules covering these states.

However, this universal pattern of convergence held only for decision-variables encoding rulebase A. Different runs produced populations with different patterns of convergence for rulebase B. This observation is explained as follows:

Although all runs showed the same pattern of convergence (for rulebase A), the values that were converged to – i.e., the rule consequents – differed greatly from run to run. Because the intermediate output (from rulebase A) is an input into rulebase B, the rule usage in rulebase B will depend on rulebase A. Therefore, given that the consequents in rulebase A differed from run to run, the selection pressure on rules in rulebase B also differed from run to run, and the pattern of convergence was different from run to run.
The fact that different runs converged to different consequents, for both rulebases, could be due to:

1. There is no inherent meaning for the intermediate variable; and/or
2. There are many possible successful control strategies.

The effect of the arbitrary intermediate variable must be isolated from the effect of different control strategies. To investigate the effect of the intermediate output, the final solutions from all 32 runs were simulated and the control strategies were assessed visually.

Figure 7.11. State and control for inverted pendulum.

Figure 7.12. State and control for inverted pendulum.
Figure 7.11 and figure 7.12 illustrate just two different control strategies (chosen because the relationship between the control and the state variables is visually apparent). Figure 7.11 shows a control strategy that pushes the cart towards the origin and lets it roll without any further control input, until the cart reaches the origin and the controller “catches” it and stabilises the cart and pole around the origin. Figure 7.12, in contrast, shows a control strategy that keeps the cart and pole under a controlled damped oscillation towards the origin. There are many other control strategies that are not described so succinctly.

Even after the control strategy is accounted for, different runs still produce different consequents to both rulebases. Table 7.10 shows the consequents for two solutions, i and ii, that both exhibit the same control strategy (as illustrated in figure 7.11). Only the 25-rule rulebase A is shown, for brevity and illustrative purposes only.

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_1$ is 1</th>
<th>$x_1$ is 2</th>
<th>$x_1$ is 3</th>
<th>$x_1$ is 4</th>
<th>$x_1$ is 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_2$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>i</td>
<td>1 2 3 4 5</td>
<td>1 2 3 4 5</td>
<td>1 2 3 4 5</td>
<td>1 2 3 4 5</td>
<td>1 2 3 4 5</td>
</tr>
<tr>
<td>ii</td>
<td>3 5 1 1 1</td>
<td>5 7 5 3 5</td>
<td>2 7 1 2 2</td>
<td>4 5 1 2 1</td>
<td>2 1 1 7 7</td>
</tr>
</tbody>
</table>

When the control strategy is known to be the same, the relationship between rulebase A consequents and rulebase B consequents can be teased out.

The examples below show a given state, and the matching rules (one rule from rulebase A covering $x_1$ and $x_2$, and one rule from rulebase B covering $x_3$ and $x_4$) for each solution. Note that the rules are linked by the intermediate variable, $y_A$, such that the rule from rulebase B depends on the consequent of the rule from rulebase A; and note that therefore solutions i and ii use different rules from rulebase B to cover the same state.
Example 1. (eg. Figure 7.11, time ≈ 5s):

\[ x_1 = \text{high (4)}; x_2 = \text{medium (3)}; x_3 = \text{medium (3)}; x_4 = \text{medium (3)} \]

Solution i:
\[ R_{18}: \text{if } x_1 \text{ is high and } x_2 \text{ is medium, then } y_A \text{ is very high (7)} \]
\[ R_{188}: \text{if } y_A \text{ is very high and } x_3 \text{ is medium and } x_4 \text{ is medium, then } y \text{ is medium (4)} \]

Solution ii:
\[ R_{18}: \text{if } x_1 \text{ is high and } x_2 \text{ is medium, then } y_A \text{ is very low (1)} \]
\[ R_{38}: \text{if } y_A \text{ is very low and } x_3 \text{ is medium and } x_4 \text{ is medium, then } y \text{ is medium (4)} \]

Example 2. (eg. Figure 7.11, time ≈ 8s):

\[ x_1 = \text{medium (3)}; x_2 = \text{low (2)}; x_3 = \text{medium (3)}; x_4 = \text{medium (3)} \]

Solution i:
\[ R_{12}: \text{if } x_1 \text{ is medium and } x_2 \text{ is low, then } y_A \text{ is very low (1)} \]
\[ R_{38}: \text{if } y_A \text{ is very low and } x_3 \text{ is medium and } x_4 \text{ is medium, then } y \text{ is high (6)} \]

Solution ii:
\[ R_{12}: \text{if } x_1 \text{ is medium and } x_2 \text{ is low, then } y_A \text{ is very high (7)} \]
\[ R_{188}: \text{if } y_A \text{ is very high and } x_3 \text{ is medium and } x_4 \text{ is medium, then } y \text{ is high (6)} \]

Example 3. (eg. Figure 7.11, time ≈ 0.5s):

\[ x_1 = \text{very high (5)}; x_2 = \text{high (4)}; x_3 = \text{low (2)}; x_4 = \text{low (2)} \]

Solution i:
\[ R_{24}: \text{if } x_1 \text{ is very high and } x_2 \text{ is high, then } y_A \text{ is very high (7)} \]
\[ R_{32}: \text{if } y_A \text{ is very high and } x_3 \text{ is low and } x_4 \text{ is low, then } y \text{ is medium-low (3)} \]

Solution ii:
\[ R_{24}: \text{if } x_1 \text{ is very high and } x_2 \text{ is high, then } y_A \text{ is very high (7)} \]
\[ R_{32}: \text{if } y_A \text{ is very high and } x_3 \text{ is low and } x_4 \text{ is low, then } y \text{ is medium-low (3)} \]

(Rule numbers refer to a row-major encoding of rulebase A and B combined, and are only listed for quick reference. Rules 12, 18, and 23 can be seen in Table 7.10. Also note that y does not correspond exactly to u in Figure 7.11, as u is the product of multiple overlapping rules and the examples above are single rules selected for illustrative purposes.)
It is observed that, given two solutions (e.g. i and ii) exhibiting the same control strategy:

1. The consequents for both rulebases – even the consequents in rulebase B that produce the final output – may be different, yet the final output (i.e., the control strategy) remains the same.
2. In Example 1 and Example 2 the interpretation of $y_A$ is reversed between solution i and ii; however
3. In Example 3 the interpretation of $y_A$ is the same in solution i and ii.
4. The magnitude of $y_A$, does not correspond to the magnitudes of $x_1$ and $x_2$.

Example 1 and Example 2 show that both extreme values of $y_A$ are used to represent just the medium and high values of $x_1$, and the medium and low values of $x_2$.

Section 7.4.3 predicted that $y_A$ would have an arbitrary meaning, and this is upheld by observations 1 and 2. Section 7.4.3 also predicted that $y_A$ would naturally represent an approximation of the control force required to centre the cart ($x_1$ and $x_2$), and that the interpretation would be arbitrarily reversed around the central set, since that interpretation would provide the finest degree of control when the cart is around the origin. This is consistent with observations 1 and 2.

However, observation 3 shows that $y_A$ is not simply reversed between solutions i and ii, and observation 4 shows that $y_A$ does not represent the control force required to centre the cart.

The actual effect of the intermediate variable is much more subtle: it does not have a fixed interpretation, but is context-sensitive. In this case, the context is provided by $x_3$ and
When \( x_3 \) and \( x_4 \) are medium (as in Example 1 and Example 2), solution i’s interpretation is the reverse of solution ii’s. When \( x_3 \) and \( x_4 \) are low (as in Example 3), solution i’s interpretation is the same as solution ii’s.

Part of the reason that the interpretation given to \( y_A \) was so unexpectedly flexible is undoubtedly due to the fact that the controller was evolved on only one test-case, so the controller encounters only a small region of the entire possible state-space, and the interpretation of \( y_A \) is evolved in the context of just this small region of the \( x_1, x_2 \) subspace. Indeed, when the initial state of the test-case is changed just slightly, the same solution (solution i, for example) fails dramatically.

The initial state used during evolution was:

\[
\begin{align*}
    x_1 &= 0.5; & x_2 &= 0.0; & x_3 &= 0.01; & x_4 &= 0.0 \\

\end{align*}
\]

If a small change is made, even such that the initial state is easier (in the sense that it is closer to the origin), for example:

\[
\begin{align*}
    x_1 &= 0.5; & x_2 &= 0.0; & x_3 &= 0.005; & x_4 &= 0.0 \quad \text{or} \\
    x_1 &= 0.4; & x_2 &= 0.0; & x_3 &= 0.01; & x_4 &= 0.0
\end{align*}
\]

then the precisely tailored interpretation of \( y_A \) fails and the solution fails to maintain control over the state-variables.

### 7.6 Conclusions

#### 7.6.1 Cooperative coevolution

The results clearly support the hypothesis that cooperative coevolution makes faster progress due to the reduced search-space. This is true regardless of whether the progress is compared over generations or over evaluations, and regardless of whether coevolution
is compared against a traditional EA with an equal number of trials per generation; or with equal evolutionarily distinct population size.

The results reveal little about the hypothesis that greedy representative selection, high-epistasis decomposition causes the coevolutionary EA to more easily become stuck in local optima, as the experiments were not designed to investigate this – the degree of epistasis between modules and the epistasis of the problem remained fixed for all runs. Nevertheless, the results from these experiments are not inconsistent with the strong evidence for this hypothesis in the literature: even though greedy representative selection made faster progress than explorative representative selection over the short term (when progress was measured by evaluations), both greedy and explorative representative selection converged to insignificantly different final solution quality over the long term. Furthermore, it is suggested that this decomposition, corresponding to carefully designed hierarchical rulebases, was successful in minimising epistasis between modules, such that explorative representative selection is not expected to outperform greedy representative selection; this also is consistent with the literature discussed in section 7.2.2.2.

The results support the hypothesis that coevolution does not necessarily produce better final solutions due to greater modularity. The final solutions in table 7.2 (where final solutions are selected by generation) show that all algorithms produced insignificantly different solutions when allowed to run for long enough.

Likewise, the final solutions in table 7.9 (where final solutions are selected by number of evaluations) show that the coevolutionary algorithms were not consistently significantly
better than the traditional algorithms, even though at 100,000 evaluations the algorithms were still making progress – they had not all reached the final solutions they reached after 1,000 generations (as shown in table 7.2).

The number of evolutionary generations appears to be an important factor here. In 100,000 evaluations, greedy coevolution completes more evolutionary generations than its counterpart, explorative coevolution; and low-population traditional completes more generations than its counterpart, high-population traditional. The more-generations-algorithms were not significantly different from one another (with very high $P = 0.999$); and the fewer-generations-algorithms were not significantly different from one another (with very high $P = 0.960$). Furthermore, the more-generations-algorithms were either significantly better ($P < 0.05$) or almost significantly better ($P = 0.082$, $P = 0.073$) than the fewer-generations-algorithms. This suggests that the effect of coevolution is not as important as the effect of evolutionary generations in determining final solution quality, and this is in agreement with the previous conclusion from table 7.2.

It is interesting that [78], which also applied cooperative coevolution to a rule-based robot controller, reported qualitatively more modular solutions from the coevolutionary EA than from the traditional EA (although the final fitness scores were the same). A similar observation could not be made from examination of rulebases in this work. One explanation for this difference may be the different encodings. [78] used an unstructured encoding for rulebases that gave the EA large scope for improving the modularity of the solution. In contrast, this work used a structured encoding (a hierarchical rulebase) that was the same for all algorithms, and so the only scope for improving the modularity was
the interpretation of the intermediate output, which proved to be very flexible across all algorithms.

7.6.2 Hierarchical design
The fuzzy system design did not impose any particular interpretation of the intermediate output, y_A, upon the EA. However, the intermediate output was anticipated to evolve to become an approximation of the control force required to bring the cart (x_1 and x_2) to the origin. Therefore the intermediate output centroids and membership functions were chosen to provide finer control around 0, just as with the final output centroids.

It is interesting to note that the evolutionary algorithm was very flexible in assigning meaning to y_A, whilst still being able to produce the same control strategies. However, this flexibility depended on a very precise interpretation of y_A in the context of a limited region of the subspace that it approximated, and this precise interpretation was possible only because it only needed to work for one test-case.

This suggests that the true flexibility of the intermediate variable’s interpretation should be treated with caution, and in the context of the environment it is evolved in.

7.6.3 Parallelism
Observe that the only reliable advantage to cooperative coevolution is the faster evolutionary progress due to a reduced search-space; that cooperative coevolution may produce better quality final solutions if the duration of the run is restricted. It may be concluded that cooperative coevolution is, at present, most useful in applications where computational cost, and therefore run time, is a significant factor.
If computational cost and run time is relevant then so is the inherent parallelism of the algorithms under consideration. Most scientific computing facilities, and even many desktop computers, make use of multiple processors, or nodes, executing in parallel. Making use of parallel processing is therefore a worthwhile consideration when comparing algorithms.

Almost all evolutionary algorithms are trivially parallel; an individual can be evaluated independently of all other individuals in the population, and the task of fitness evaluation can therefore be trivially distributed across multiple nodes. Section 7.5.1 showed that, in the case of the inverted pendulum problem executing on the specified cluster, the cost of fitness evaluation is approximately 2 orders of magnitude more expensive than the cost of communication, making parallelising the code using the master-slave model worthwhile. The only caveat is that in the master-slave model, all solutions must be evaluated before the next generation can be computed and distributed; if one solution takes much longer to evaluate, for whatever reason, then all other processes will block until it finishes and the next generation can be distributed.

Most scientific computing facilities are either multiprocessor computers or homogenous clusters with fast networks. This easily supports the distribution of solutions, as the fast network supports sending out all of the solutions at the beginning of every generation and collecting all of the fitness scores at the end, and the homogenous nodes support a predictable distribution of solutions so that the fitness scores all return at approximately the same time with minimal blocking.

However, there is another fast-growing model of supercomputing. Distributed grid-based computing makes use of wide networks of heterogeneous processors connected by
potentially slow networks. In such a model it is much more difficult to distribute solutions and collect results every generation in a way that maximises processor usage. Cooperative coevolution is well suited to take advantage of grid-based computing, because it can be parallelised with a coarser granularity and asynchronously. There is less communication between processors – only the representative individuals are sent, not the whole population, and only upon request, not every generation – which can be supported by the slower networks. The populations can be asynchronous – one population does not need to wait for another population to complete each generation – which can maximise the usage of heterogeneous processors.

This is a pragmatic advantage of cooperative coevolution that was not investigated in this study, but should be considered on a case-by-case basis depending on the computing facilities available.

### 7.6.4 Publications

Part of this research was published in [109].

Its conclusions were subsequently adopted in [96], and hierarchical topologies for the inverted pendulum problem were further investigated in [97, 113].
Chapter 8. Constrained Multiobjective Optimisation of Inverted Pendulum

8.1 Introduction
This chapter presents a multiobjective formulation of the inverted pendulum problem, solved by a constrained multiobjective evolutionary algorithm (CMEA). There are two distinct aims that are furthered by the work in this chapter:

1. In the first half, multiobjective optimisation is used to gain a deeper understanding of the inverted pendulum problem. Section 8.2 describes the inverted pendulum as a multiobjective problem. Section 8.3 discusses qualitative results.

2. In the second half, the inverted pendulum problem is used to gain a deeper understanding of multiobjective optimisation. Section 8.4 examines the theoretical basis of the CMEA. Section 8.5 describes the implementation. Section 8.6 gives quantitative results used to support conclusions.

This chapter also functions as a bridge between the previous chapters, which were focussed on the evolution of fuzzy systems and controllers, and the following chapter, which is focussed on constrained multiobjective optimisation.

8.1.1 Motivation
Pareto-based multiobjective optimisation is commonly held to be preferable to the weighted-sum aggregation traditionally used to solve multiobjective problems. As previous implementations of the inverted pendulum problem, both in this thesis and in the
literature, have predominantly used the weighted-sum approach, this chapter investigates a Pareto-based implementation.

This chapter serves as an introduction into CMEA research, which is taken up in the following chapter and ultimately extended into a novel CMEA.

8.1.2 Relationship to other chapters
This chapter uses the inverted pendulum problem from chapter 5.

Multidimensional encoding and crossover are used from chapter 6.

 Hierarchical rulebases and cooperative coevolution, used in chapter 7, are inappropriate for this study on multiobjective optimisation. This is discussed in sections 8.2.1.2 and 8.5.1.4.

The study of constrained multiobjective optimisation is continued in chapter 9.

8.2 Multiobjective inverted pendulum
This section describes the design of a multiobjective formulation of the inverted pendulum problem. The emphasis of this section is on the inverted pendulum problem from a multiobjective point-of-view, not on the particular CMEA techniques. As such, the overview of multiobjective and constrained multiobjective techniques given in chapter 3 is sufficient to appreciate this discussion. The CMEA used here is based on NSGA-II [28]. A more detailed examination of CMEA techniques is given in later sections of this chapter, and in the following chapter.

8.2.1 Model, fuzzy system, and simulation
The model, fuzzy system, and simulation used to control the inverted pendulum are identical to those described in chapter 5.
8.2.1.1 Encoding
The solutions are fully specified fuzzy rulebases as described in chapter 5. There are 4 input variables (the state variables) each covered by input 5 fuzzy sets, resulting in a $5 \times 5 \times 5 \times 5$ rulebase of 625 rules. Each decision variable in the solution is a single consequent corresponding to one of the 625 rules. Each decision variable can take one of 13 integer values corresponding to the 13 output fuzzy sets, such that neighbouring membership sets are consecutive integers.

This algorithm uses the multidimensional encoding and crossover described in chapter 6. A multidimensional encoding and crossover removes dimensional encoding bias, whilst retaining epistatic bias that may be exploited.

8.2.1.2 Flat rulebase vs. hierarchical rulebase
One of the first design decisions was the choice between a single-rulebase fuzzy design (as used in chapter 6) and a hierarchical, multiple-rulebase fuzzy design (as used in chapter 7 in conjunction with cooperative coevolution).

The advantage of a hierarchical design is that fewer rules need be learnt. In a hierarchical design, inputs are divided into groups and each group of inputs is fed into its own rulebase. These rulebases produce outputs which are fed, as inputs, into another rulebase that aggregates them in some way to produce a final output (there may be more than 2 layers of rulebases as described here). This modular design reduces the total number of rules in a fully-specified rulebase because the number of unique combinations of input membership sets is reduced. By limiting the number of inputs to a single rulebase, the exponential growth of rulebase size as number of inputs increases is limited; the sum of rules in multiple low-dimensional rulebases is less than the number of rules in a single high-dimensional rulebase.
The disadvantage to a hierarchical design is a possible loss of ability to accurately map all input combinations to appropriate outputs. Each low-level rulebase in the hierarchy must produce a single output that appropriately represents the information contained in 2 (or more) inputs. This assumes that the 2 (or more) inputs carry some information in common that can be distilled as a single output intended for further processing. The fuzzy system designer must decide what decomposition of inputs into groups will produce the most effective hierarchical system.

A study of hierarchical decomposition of inputs for the inverted pendulum problem was given in [97] (for 2 layers) and [113] (for 3 layers). These studies identified the importance of grouping input variables appropriately. In particular, the design of the hierarchical fuzzy system can restrict the control strategy exhibited by the controller.

As this chapter is a study of the diversity of Pareto-optimal control strategies possible for controlling the inverted pendulum, the design of the fuzzy system should avoid restricting, or even biasing, the controller to a particular strategy. To this end, a single-rulebase design is used to ensure that every possible combination of input variables can result in an independent output; the design does not restrict or bias any input-output mappings, allowing any possible control strategy to be represented.

### 8.2.2 Extracting objectives and constraints

Appropriate objectives and constraints must be extracted from the inverted pendulum problem description, given in chapter 5. The objective functions were immediately obvious, being the weighted components of the single-objective fitness function used previously.
It was not obvious at first whether the constraints should represent the state-space boundaries, the goal boundaries, or both. However, since the simulation terminates immediately upon leaving the state-space boundaries, they are not useful as constraints: the solution cannot violate state-space boundary constraints by a variable amount (it either breaks the constraint or not), so two solutions that both break the same constraint cannot be differentiated from one another, and so the constraint cannot guide the EA’s search towards feasibility. Constraint scores should guide the search; state-space boundary constraints cannot do this.

In contrast to state-space boundary constraints, goal boundary constraints can be violated by a variable amount, and so are useful in guiding the search towards feasible solutions.

The following is the first constrained, multiobjective formulation of the inverted pendulum problem:

**Formulation 1:**

**Objectives**

\[ f_i = \frac{1}{T} \sum_{t=0}^{T} |x_i^t| \quad \text{for} \quad i = 1, \ldots, 4 \]

**Constraints**

\[ g_i = |x_i^t| - b_i \quad \text{for} \quad i = 1, \ldots, 4 \]

\[ f_5 = \frac{(T_{\text{max}} - T)}{T_{\text{max}}} \]

where \( T \) is the time-step that the simulation terminated at (may be less than \( T_{\text{max}} \) if the controller failed early), \( x_i^t \) is the state variable \( x_i \) at time-step \( t \), and \( b_i \) is the goal boundary for the \( i^{th} \) state-variable.

The objectives are similar to the fitness measures described in chapter 5: the mean of the absolute value of each state-variable, over the duration of the run. However, the penalty term is no longer needed, as the constraints are handled explicitly in this multiobjective formulation.
Each constraint is the difference, positive or negative, between the state-variable and the goal boundary at termination of the simulation. This value will be maximal if the controller also causes the state-variable to leave the bounds of state-space, as the simulation will terminate at that point, with the state-variable’s value as far from the goal boundary as it is possible to be in state-space. Successful solutions, that complete the simulation with the state inside the goal region, will have constraint scores ≤0. Thus, the constraints were chosen such that feasible and infeasible solutions correspond to controllers that “succeed” or “fail” at the problem, as defined in the problem description in chapter 5.

Formulation 1 was implemented and tested, but found to be unsatisfactory: the multiobjective EA failed to reliably find successful controllers for the same test-cases that the single-objective EA used in chapter 6 and 7 successfully solved. Upon further consideration, it was determined that the constraints were redundant. The objectives (the mean of state-variable values), on their own, should guide the search into the feasible region. Arguably, they would do so more effectively, being a measure of the controller’s performance over the entire duration of the simulation instead of a measure of the controller’s performance at its termination.

Moreover, the constraints were preventing $f_5$, the time remaining fraction, from ever being useful. All feasible (“successful”) controllers maintain control for the entire duration of the simulation, so $f_5$ is necessarily 0 for all feasible solutions, and is therefore useful only for differentiating infeasible solutions. However, objective scores are only
used to compare solutions that are feasible, so $f_5$ was only used on solutions it couldn’t differentiate!

In addition to being useless as measure of performance, $f_5$ also prevented the hypervolume diversity measure from differentiating solutions. The uniquely dominated hypervolume of a solution, since it is a measure of volume, is essentially the product of distances between points in objective-space (as opposed to the sum of distances between points otherwise commonly used). As the distance between all solutions in one objective-space dimension, $f_5$, was 0, the uniquely dominated hypervolume was also 0 regardless of the distance between solutions in other dimensions. (See section 8.4.3.2 for a description of how hypervolume is calculated.)

Ordinarily, when solutions are expected to vary rarely have any identical objective scores, this drawback of hypervolume should not be noticeable: in a stochastic search, what should it matter if one solution is evaluated unfairly? However, when the entire population is evaluated unfairly it becomes a problem! This characteristic of the hypervolume diversity measure is discussed further in the conclusions to this chapter.

In this case the immediate solution is given in formulation 2.

Formulation 2:

Objectives

$$f_i = \frac{1}{T} \sum_{t=0}^{T} |x'_t| \quad \text{for} \ i = 1, \ldots, 4$$

Constraints

$$g_1 = (T_{\text{max}} - T) / T_{\text{max}}$$

In this final formulation, the problem is represented by 4 objectives (the state-variable means) and 1 constraint (the time remaining fraction). This allows $g_1$, previously $f_5$, to usefully differentiate infeasible solutions, whilst simultaneously removing the problematic objective-space dimension that was causing all hypervolumes to be 0.
Upon careful consideration, it is clear that time remaining is more properly a constraint than an objective. Time remaining measures how long the controller kept the state inside the state-space bounds; a time remaining of 0 is required of all successful controllers, and is therefore required of all feasible solutions. Time remaining determines feasibility of a solution; as such it is more appropriately formulated as a constraint.

Furthermore, formulation 2 also reflects the weights (given in chapter 5) that were found to be effective in single-objective implementations of the inverted pendulum problem. In single-objective implementations, the fitness measures are weighted and summed, with the greatest weight going to the time remaining. This weighting is also intuitive: when learning to balance a pole, it would be natural to first concentrate on simply keeping the pole upright, and after that skill is mastered, only then try to keep it upright and in place and with the least movement necessary.

This learning strategy is mimicked by formulating time remaining as a constraint, since constraints must be satisfied before objectives are optimised.

Formulation 2 was found to be highly effective, as demonstrated in the following sections.
8.2.3 Test cases

The following test cases were used to evaluate the controllers:

<table>
<thead>
<tr>
<th>Test case</th>
<th>x₁</th>
<th>x₂</th>
<th>x₃</th>
<th>x₄</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.25</td>
<td>0.25</td>
<td>0.1</td>
<td>0.1</td>
</tr>
<tr>
<td>2</td>
<td>0.25</td>
<td>0.25</td>
<td>0.1</td>
<td>-0.1</td>
</tr>
<tr>
<td>3</td>
<td>0.25</td>
<td>0.25</td>
<td>-0.1</td>
<td>0.1</td>
</tr>
<tr>
<td>4</td>
<td>0.25</td>
<td>-0.25</td>
<td>0.1</td>
<td>-0.1</td>
</tr>
<tr>
<td>5</td>
<td>0.25</td>
<td>-0.25</td>
<td>0.1</td>
<td>0.1</td>
</tr>
<tr>
<td>6</td>
<td>0.25</td>
<td>-0.25</td>
<td>0.1</td>
<td>-0.1</td>
</tr>
<tr>
<td>7</td>
<td>0.25</td>
<td>-0.25</td>
<td>-0.1</td>
<td>0.1</td>
</tr>
<tr>
<td>8</td>
<td>0.25</td>
<td>-0.25</td>
<td>-0.1</td>
<td>-0.1</td>
</tr>
</tbody>
</table>

In the single-objective implementations of the inverted pendulum problem, in previous chapters, the fitness function was evaluated for each test case, and the mean of the evaluations for all test cases was taken to be the solution’s overall fitness.

It was not immediately obvious how test-cases should be aggregated into the multiobjective formulation of the problem:

- Should aggregation be at a high level, above ranking – should the entire population be ranked for each test-case, and the mean of the ranks for all test cases be taken to be the solution fitness?
- Should aggregation be at a low level, inside the objective functions – should the objective functions be the means of the measures over all test cases?
- Should aggregation be at a medium level, outside the objective functions but below ranking – should every measure for every test-case be an independent objective?

Previous studies [91, 93, 95] have also considered aggregation at an even higher level, above the EA, by running the EA once for each test-case and using fuzzy amalgamation.
to aggregate the best controllers for each test-case. Besides the disadvantage of increased computational overhead, this is inappropriate for a multiobjective, Pareto-based EA that returns a set of solutions (not a single “best” solution) with deliberately conflicting behaviours.

Aggregation at the high level, by averaging ranks, was tested but found to be counterproductive because it disrupted the formation of orderly ranks inside which diversity can be used to differentiate solutions. (A similar problem is encountered with blended ranking, investigated in the following chapter, which also “averages” ranks.)

Aggregation at the medium level, by making every measure for every test-case an objective sounds attractive from an idealistic Pareto point of view, because it avoids assumptions about the commensurability and weighting of state-variable measures from different test-cases. It takes a “hands-off” approach, letting Pareto optimisation take care of the aggregation. Unfortunately it results in an exponential increase in the size of objective-space, as the number of dimensions is multiplied by the number of test-cases.

Aggregation at the lowest level, by averaging state-variable measures inside the objective functions, appears disconcertingly similar to the weighted-sum aggregation traditionally used to aggregate multiple objectives into a single objective function, which is one of the things Pareto-based optimisation avoids! It assumes that all test cases should be given equal weight, and that the results from one test case are commensurable with the results from another test case.
However, it may be entirely reasonable to assume equal weight and commensurability for all test-cases.

As explained in chapter 3, Pareto optimisation assumes that all objective functions measure non-commensurable qualities of the solution. The key here is to identify what qualities the objective functions should measure. Arguably, the objectives should not measure performance on a specific test case, but rather measure the solution’s behaviour – in this case, its control strategy – in general. After all, the whole point of using multiple test-cases in the first place is to obtain a more general picture of a solution’s behaviour. Test cases are meant to be representative of conditions the solution is expected to encounter, in order to predict its behaviour. Therefore, by definition, individual test-cases are insignificant and must be commensurable. Furthermore, the weighting on all test-cases must be equal, otherwise it implies that certain test cases are more important than others, and that should be reflected in the problem description.

It may also be assumed that, for test-cases that are initial positions in state-space, different test-cases do not present conflicting objectives. Performance gain in one test-case cannot come at the expense of a performance loss in another test-case, for any given objective.

At best, a performance gain in one test case will also be a performance gain for all test-cases in which the controller encounters overlapping regions of state-space. This is because the goal for all test-cases is the same, so the optimal control strategy for any given region of state-space is the same for all test-cases. At worst, a performance gain for one test case will have no effect for all other test-cases. This is because the rules
governing control strategy are local to regions of state-space, so the control strategy in one region (i.e., one test-case) will have no effect (beyond the scope overlapping membership sets) on the control strategy in another region.

It is therefore appropriate that each objective function measures the solution’s behaviour in general, over multiple test-cases.

8.3 Discussion of inverted pendulum

8.3.1 Visualisation of controller-space

Every controller is characterised by 4 objective scores, reflecting the controller’s ability to effectively minimise the 4 state-variables. There is no single perfect controller: past a certain point, improving the control over one state-variable comes at the cost of poorer control over other state-variables. This is the Pareto front in objective space.

The visualisations below show the non-dominated solutions found from 20 runs of the CMEA (the algorithm is described in detail in the following sections; this section is only concerned with the multiobjective nature of the inverted pendulum problem). All solutions from all runs were pooled, and those that were universally non-dominated are shown below. The intent is to illustrate the space of Pareto-optimal controllers.

Figure 8.1 - figure 8.2 show the solutions in objective space. Three axes in 3D Euclidean space are used to represent the controller’s ability to control cart position ($f_1$), cart velocity ($f_2$), and pole angle ($f_3$); and grey-scale intensity is used to represent the controller’s ability to control pole angular velocity ($f_4$).
Figure 8.1. 4D visualisation of non-dominated front. Origin at lower left front corner.

Figure 8.2. 4D visualisation of non-dominated front. View from the right of Figure 8.1.
Careful examination of the graphs will reveal some points that appear to be dominated by other points, in 3D Euclidean space. However, further examination will show that a point
that appears to be dominated will have a lower grey-scale intensity (darker) than those that “dominate” it, making it non-dominated in 4D objective space. This is difficult to show on paper, hence this note.

8.3.2 Trade-offs

These graphs are enlightening for considering the set of all Pareto-optimal controllers, in which no solution can be dominated by any other. The graphs clearly show the trade-offs that a controller must make to successfully control the inverted pendulum. The inverted pendulum is truly a multiobjective problem.

The graphs show a “leading edge” that stretches from high-velocity-low-position to low-velocity-medium-position, with relatively stable pole angle and pole angular velocity across the length of the edge. The majority of solutions cluster along this “leading edge”, suggesting that the trade-off between controlling cart position and cart velocity is one of the easier trade-offs for the EA to make. In hindsight this is not surprising, since the controller’s output modifies the cart’s velocity and so velocity and position are the two state-variables most directly affected by the control strategy.
There are three main ways in which solutions depart from this leading edge trade-off. The first departure is a “spire” from the high-velocity-low-position end, reaching up into high pole angles (vertical axis) and high pole angular velocities (grey-scale intensity). The spire leans away from the leading edge ever so slightly, continuing the leading edge’s trade-off of velocity for position. The spire shows that optimal control of cart position comes at the cost of poor control over the pole.

The second departure is a “leg” kicking backward from the low-velocity-medium-position end of the leading edge. The leg is “dominated” in the visualisation’s 3D Euclidean space, but has lower grey-scale intensities. The leg shows that control of velocity and position can be traded off for better control of pole angular velocity.

The final departure from the leading edge is less specific than the spire and the leg. It is a gradual fall-off backwards from the leading edge across its length. The fall-off corresponds to a decrease in vertical height (pole angle) and a decrease in grey-scale intensity (pole angular velocity). This fall-off shows the more general trade-off between controlling the cart and controlling the pole.

8.4 Background

This section gives the background to the remainder of this chapter, which examines the quantitative performance of the CMEA used to achieve the qualitative results above.

8.4.1 Pareto-based multiobjective optimisation

This chapter uses the constrained multiobjective problem formulation from section 3.6:

\[
\begin{align*}
\text{minimise} & \quad f_i(x), \quad i = 1, \ldots, M \\
\text{subject to} & \quad g_j(x) \leq 0, \quad j = 1, \ldots, N
\end{align*}
\]
8.4.1.1 Pareto dominance

Pareto dominance is a means of comparing the quality of two solutions in objective space. One solution, \( a \), dominates another solution, \( b \), written as \( a > b \):

\[
\text{a > b} \quad \text{iff} \quad \left( \forall i \in \{1, \ldots, M\}, f_i(a) \leq f_i(b) \right) \land \left( \exists i \in \{1, \ldots, M\} \mid f_i(a) < f_i(b) \right)
\]

(8.1)

that is, \( a \) is at least as good as \( b \) on all objectives, and is better on at least one. If neither \( a > b \) nor \( b > a \) then \( a \sim b \), that is \( a \) and \( b \) are non-dominated.

Figure 8.6. Visualisation of domination. Points \( a \) and \( b \) are non-dominated with respect to one another. Point \( c \) is dominated by \( b \) but not \( a \).

There may be many solutions to (3.5) such that it is impossible to improve the solution’s performance on one objective, \( f_i \), without degrading its performance on another objective, \( f_j \). These solutions are non-dominated with respect to each other. The set of all such solutions is called the Pareto front. Defined alternatively, the Pareto front is the set of all universally non-dominated solutions.

8.4.1.2 Approximating the Pareto front

It is the goal of all Pareto-based multiobjective EAs to find a set of solutions that approximates the Pareto front. A good approximation set will contain solutions that are close to the Pareto front and that are distributed evenly across its surface.

Proximity to the Pareto front is usually considered to be the most important characteristic of the solution set, as proximity is related to dominance – the only way to compare individual solution quality.
However, **diversity** of the solution set is also extremely important, because the fundamental purpose of finding a *set* of solutions, rather than a single solution, is to offer the final decision-maker a choice of solutions of diverse characteristics. A solution set that lies exactly on the Pareto front (i.e., has the best possible proximity) is useless if it does not contain a solution close to the region of the Pareto front that the decision-maker is ultimately interested in.

Unfortunately, because the final solution set’s diversity is used to support an ultimately subjective decision, it is difficult to define what is expected of an ideal diversity. Nevertheless, diversity can be broadly characterised in two ways: the extent of the solution set, and the distribution of the solution set. The extent should be wide in all objective dimensions, to identify the entirety of the Pareto front. The distribution should be uniform so that no region of the Pareto front is over- or under-represented, and the set has the greatest variety of solutions to offer to the decision-maker.

### 8.4.2 Ranking

#### 8.4.2.1 Pareto dominance ranking

The Pareto dominance relation is the only way of directly comparing solution quality in a Pareto-based multiobjective optimisation. If the evolutionary algorithm only uses binary comparisons for selection – tournament selection – then the Pareto dominance relation is sufficient on its own. However, using any stochastic selection (including tournament selection) for selection for survival (as opposed to selection for reproduction) is counterproductive in most multiobjective EAs: see section 8.4.4 below, and section 9.3.2 in the following chapter.
Dominance ranking [38] is one way of assigning fitness to a solution in a multiobjective problem. It is an iterative process: those solutions in the population that are non-dominated are assigned to the first rank. They are then removed from consideration, and those solutions that previously were dominated only by non-dominated solutions become the new non-dominated solutions and are assigned to the next rank. They are then removed, and the process continues until all solutions in the population are ranked. A solution’s fitness then corresponds to its rank.

8.4.2.2 Extended dominance ranking
Ranking can also handle constraints by replacing the Pareto dominance relation with the extended dominance relation, as given in (3.6) in section 3.6.2.1.

Extended dominance ranking is used in this chapter. Chapter 9 examines extended dominance ranking, and alternative constraint-handling heuristics, in greater detail.

8.4.3 Diversity
Dominance ranking provides the selection pressure to move through objective space towards the Pareto front, but provides no pressure to spread out along the Pareto front. Ranking is therefore combined with some measure of the diversity that a solution contributes to the solution set, to provide selection pressure to evolve a diverse solution set. The diversity measure distributes solutions to clearly and uniformly approximate the Pareto front, giving the final decision maker the most diverse set of solutions possible.

Chapter 9 examines the role that diversity plays during evolution, particularly in avoiding local optima, in greater detail.

Note that the measure of a solution’s contribution to diversity is different from the measure of a solution set’s diversity: the former guides evolution to achieve the latter.
There are many possible measures of solution diversity (see section 3.5.2.4). The two measures used in this chapter are described below.

### 8.4.3.1 Distance-to-neighbours

The distance-to-neighbours measure (used in NSGA-II [28]) is a measure of the distance to a solution’s two closest neighbours (one on either side) over all objective dimensions. It can be visualised as the average length of the sides of a hypercuboid enclosing the solution and bounded by its neighbours in each dimension.

For each objective dimension:

1. The objective scores for that dimension are normalised over the current rank.
2. Solutions are sorted by objective score.
3. For each solution, the distance between objective scores of that solution’s two neighbours (one on either side) is added to that solution’s diversity score.
4. Solutions at the extremes of each dimension are assigned an arbitrary diversity, equal to the largest representable real number.

![Figure 8.7. Distance-to-neighbours metric. Dotted box shows hypercuboid enclosing the white point.](image)

This measure is sensitive to the scale of different dimensions – different scales will change the relative ordering of solutions produced by this measure. That is why it is important to normalise all objective scores to [0,1] for each dimension.

This method is only accurate for ranks of mutually non-dominated solutions, since it assumes that solutions lie on the same surface. This means that the diversity measure can
only meaningfully differentiate solutions within the same rank, but this is not a problem when diversity is secondary to dominance.

8.4.3.2 Uniquely dominated hypervolume

The uniquely dominated hypervolume measure (used in LAHC [56] to admit solutions into an archive) uses the volume of objective space dominated by that solution but not dominated by any other solution in its rank.

![Figure 8.8. Hypervolume metric. Dotted lines show dominated space extending back from each point. Shaded box shows the uniquely dominated hypervolume of the white point.](image)

The uniquely dominated hypervolume can be calculated by Fleischer’s algorithm [56], a recursive process by which the solution being measured is divided into successively smaller non-dominated hypercuboids. The sum of volumes of all the non-dominated hypercuboids is the uniquely dominated hypervolume of the solution. To begin, the solution becomes the first parent node:
1. Parent nodes generate child nodes that are not dominated by any solution in the population other than the parent node, or any other child node.

2. Since a child node is non-dominated except by its parent node, that child node’s uniquely dominated hypervolume can be added to its parent node’s volume. To calculate a child node’s volume, it becomes a parent node and its own child nodes’ volumes are calculated recursively (step 1).

3. Recursion halts when no more non-dominated child nodes can be generated – at the tail end of recursion, each child node is a simple hypercuboid whose volume can be calculated trivially.

Solutions that have extreme positions in objective space are assigned an arbitrary volume equal to the largest representable real number to preserve them (any measure of uniquely dominated hypervolume on the extreme points in space would be arbitrary anyway).

Unlike distance to neighbours, uniquely dominated hypervolume is insensitive to scale: the scale of different dimensions cannot change the relative ordering of solutions produced by this measure, so no normalisation is required.

This measure was proposed in [56] as the basis for admitting solutions to an external archive. Here it is used as a diversity measure. Like distance-to-neighbours, uniquely dominated hypervolume is only applicable to ranks of mutually non-dominated solutions, since all dominated solutions will have a 0 uniquely dominated hypervolume, and dominated solutions (which are useless in a final solution set) can limit non-dominated solutions’ uniquely dominated hypervolumes, giving a distorted measure of their usefulness.
Solution-set hypervolume (not uniquely dominated hypervolume) is used as a metric for comparing solution sets. See section 8.4.5.4.

8.4.3.3 Distance vs. hypervolume
Uniquely dominated hypervolume is arguably a more accurate measure, since Fleischer’s algorithm calculates the sum of all hypercuboids of objective space that a solution uniquely dominates, whereas distance-to-neighbours uses the single largest hypercuboid that encloses the solution and fits inside all neighbours in all dimensions. Figure 8.7 and figure 8.8 show that these measures are similar in 2 dimensions, but in higher dimensions the distance-to-neighbours hypercuboid is only an approximation of the empty space surrounding a solution.

However, distance-to-neighbours is much simpler to calculate, and as the following example shows, the two measures are measuring fundamentally different things.

8.4.3.4 Approximating the Pareto front
The final solution set from a multiobjective EA is an approximation of the true Pareto front. It is a finite set of points that lie on (or near) the Pareto front, from which its shape may be inferred. In most real-world problems, the true Pareto front is unknown.

Given the task of reconstructing the Pareto front from the approximation set, an intuitive approach is to treat the approximation set as a piecewise linear approximation, as shown in figure 8.9-centre. A piecewise linear approximation can give a good visual approximation of the Pareto-front’s curves, but it will actually lie beyond the Pareto front (in unreachable regions of objective space) when approximating convex curves of the Pareto front.
Another approach is to use Pareto dominance. The Pareto front, by definition, cannot be dominated by any solutions that are known to exist (i.e., the approximation set). The approximation set can therefore determine a boundary that is “greater-than-or-equal-to” the true Pareto front in objective space. The exact shape of the Pareto front cannot be determined by any approximation method, but the dominance approximation at least imposes a hard limit on where the Pareto front can lie. Dominance approximation is shown in figure 8.9-right.

![Figure 8.9. Left: Pareto front and an approximation set of 10 points. Centre: Piecewise linear approximation. Right: Dominance approximation.](image)

With this knowledge, the effect of the two solution diversity measures on the evolution of the solution set can be considered. Solution diversity is commonly used as a secondary fitness measure, after dominance rank, to select solutions for reproduction and survival. In particular: if a rank is too large to fit into the population (or external archive), the solutions with the lowest diversity score will be removed.
Consider the solution set shown in figure 8.9, and imagine that only 8 of the 10 solutions can survive to the next generation. Figure 8.10-left shows the solution set with both diversity scores recorded for each solution: the distance score is below each point, and the hypervolume score is above each hypercuboid. Note that the size of the distance score does not correspond to the size of the hypervolume score.

Figure 8.10-centre shows the solution set after removing the 2 solutions with the lowest distance to neighbours. Figure 8.10-right shows the solution set after removing the 2 solutions with the lowest uniquely dominated hypervolume.
Finally, Figure 8.11 shows the dominance approximation from each set overlayed on the true Pareto front. The distance-to-neighbours diversity measure causes selection to favour a uniform distribution of solutions across the Pareto front, whereas the uniquely dominated hypervolume measure causes selection to favour a solution set whose dominance approximation most closely matches the shape of the Pareto front.

In particular, uniquely dominated hypervolume will favour a distribution of points that allocates more points to concave regions of the Pareto front than to convex regions, since the dominance approximation matches convex regions more accurately than concave regions. (Note that it will not move all points to concave regions, since the more points there are in a region, the less uniquely dominated hypervolume each will have; after some threshold is crossed, a point on a convex region will have a greater uniquely dominated hypervolume)

The diversity measure, used in conjunction with dominance ranking, is intended to provide the selection pressure to evolve a desirable distribution of solutions across the Pareto front. A desirable solution set is usually described as: a) clearly approximating the shape of the Pareto front, and b) offering a uniform distribution of choices to offer the decision-maker. These are often assumed to be the same thing, under the assumption that a uniform distribution should leave no region of the Pareto front over- or under-represented. However, this examination has shown that the best approximation of the Pareto front is not necessarily achieved by a uniform distribution.
The question facing the practitioner is, therefore, does the final-decision-maker desire an optimal approximation of the Pareto front, or an optimal selection of available choices? That is a question whose answer is outside the scope of this research.

8.4.4 Elitism

Excessive elitism is typically viewed with extreme caution in single-objective EAs. Elitism is regarded as producing too great a selection pressure on the population, leading to loss of diversity and premature convergence. The main purpose of elitism, which is to preserve the best solution(s) found so far, can be accomplished with a low elitism, which minimises the effect on diversity loss and premature convergence.

A fundamental difference between single-objective and multiobjective EAs is that single-objective EAs are expected to produce just one final solution, and the rest of the population is irrelevant; whereas multiobjective EAs are expected to produce a set of Pareto-optimal solutions, and the entire population forms the solution set.

At first it might seem that for this reason elitism, associated with loss of diversity and premature convergence, would be even worse for multiobjective EAs than for single-objective EAs. However, elitism in some form is actually essential for a multiobjective EA.

Firstly, diversity techniques in single-objective EAs typically work independently of fitness (as in restricted mating, niching, and crowding), and so elitism based on fitness works against diversity. In contrast, diversity techniques in multiobjective EAs typically form part of the fitness evaluation (in conjunction with rank), and so elitism based on fitness actually promotes diversity. Furthermore, because rank (coming from Pareto-dominance) is a very coarse-grained measure of fitness, it is not uncommon for the entire
population to have the same rank (particularly towards the end of the run). In this case the entire population is differentiated only by the diversity component of fitness, and elitism is therefore a very strong tool for increasing diversity.

Secondly, the goal of any multiobjective EA is to produce an entire population of non-dominated solutions representing the Pareto set. Dominated solutions are useless in identifying the Pareto set, and waste space in the population. Elitism ensures that as much of the population as possible is useful in identifying the Pareto front.

Whereas in a single-objective EA, elitism accelerates convergence to a single solution, in a multiobjective EA it accelerates collapse to a single rank, and then diversification of that rank – exactly what is desired of the final solution set.

The necessity of elitism is also discussed in section 9.3.2 in the following chapter, and empirical evidence was presented in [114]. A method for controlling the extent of elitism in multiobjective EAs was given in [29].

### 8.4.5 Comparing solution sets

When comparing traditional, single-objective EAs, it is relatively simple to compare the objective scores of the best solutions found, in order to compare the overall suitability of the algorithms. It is not so simple to compare multiobjective EAs that produce a large set of non-dominated solutions. Metrics for comparing non-dominated solution sets are an ongoing area of research (see [118]).

#### 8.4.5.1 Pareto comparison

The concept of Pareto dominance (of solutions) can be extended to apply to sets of non-dominated solutions. One solution set, $A$, dominates another solution set, $B$, if for every
solution in $B$, there is an equal or better solution in $A$, and there is at least one solution in $A$ that dominates a solution in $B$. In terms of Pareto dominance, this can be phrased as:

$$A \succeq B \text{ iff } (\forall b \in B, \exists a \in A \mid (a > b) \lor (a = b)) \land (\exists a \in A, \exists b \in B \mid a > b) \quad (8.2)$$

In other words, there is never a positive reason to select a solution from $B$, because there will always be an equal or better solution in $A$.

Pareto dominance applied to solutions is analogous to Pareto dominance applied to solution sets; the difference is that in the former, objective scores are compared, whereas in the latter, solutions are compared. However, there is a difference between comparing objectives vs. comparing solutions: the number of objectives is constant across solutions, but the number of solutions is not constant across solution sets. This creates an additional way that a solution set, $A$, can be better than a solution set, $B$: if $A \triangleright B$. In other words, there is never a positive reason to select a solution from $B$, because there will always be an equal solution in $A$ as well as additional choices.

These relationships, $A \succ B$ and $A \triangleright B$ are the only relationships that can be used to say that one set is “better than” another, in Pareto optimisation. $A$ better than $B$ is written as $A \triangleright B$, which is shorthand for $(A \succ B) \lor (A \triangleright B)$.

If there is at least one solution in $A$ that is not dominated by any solution in $B$, and at least one solution in $B$ that is not dominated by any solution in $A$, then $A$ and $B$ are incomparable (written $A \parallel B$), no matter what other characteristics the sets may have.

Just as $A \succ B$ and $A \triangleright B$ are the only ways $A$ can be better than $B$, so are they the only ways that $B$ can be worse than $A$. Any other relationship, including equals and incomparable, results in $B$ being “not-worse-than” $A$. It can be useful to know that one
set is not worse than another, without necessarily knowing if it is actually better than the other, or simply equal or incomparable to.

**Table 8.2. Relevant Pareto relationships. All are reversible (eg. \((A \triangleright B) \iff (B \triangleright A)\)**

<table>
<thead>
<tr>
<th>Relationship</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>(A = B)</td>
<td>Equals</td>
</tr>
<tr>
<td>(A \supset B)</td>
<td>Superset of</td>
</tr>
<tr>
<td>(A \triangleright B)</td>
<td><strong>Dominates:</strong> Every solution in (B) is dominated by or equal to a solution in (A), and at least one is dominated.</td>
</tr>
<tr>
<td>(A \parallel B)</td>
<td><strong>Incomparable:</strong> (A) contains at least one solution non-dominated by (B) and vice versa.</td>
</tr>
<tr>
<td>(A \triangleright B)</td>
<td><strong>Better than:</strong> ((A \triangleright B) \lor (A \supset B))</td>
</tr>
<tr>
<td>(A \not\triangleright B)</td>
<td><strong>Not worse than:</strong> ((A \triangleright B) \lor (A \parallel B) \lor (A = B))</td>
</tr>
</tbody>
</table>

This notation is based on the notation used in [118].

### 8.4.5.2 Limitations of Pareto comparison and other metrics

Pareto comparison provides unambiguous, simple criteria for one set being “better” than another. Unfortunately, from a practical point of view, it is extremely common for two solution sets to be incomparable by Pareto comparison, making it all but useless for comparing solution sets from competing algorithms. Furthermore, whilst Pareto comparison can tell whether one set is better than another, it cannot give a metric of *how much* better the set is.

This work therefore uses some comparison metrics that are not strictly Pareto-compatible, but may nevertheless be useful for comparing algorithm behaviours.
The results from these measures must be interpreted with caution, because they do not necessarily imply that one set is better than another from a Pareto standpoint. In [116] it was shown that comparing scores for two sets obtained from any unary metric can never prove that one set is better than another. The best that can be proved is that one set is not worse than (i.e., better than or incomparable to, but not worse than) another set. It is also possible to prove that one set being better than another implies a better score, but it is impossible to say that a better score implies a better set.

Despite these limitations, the metrics described below are used, with caution, in this chapter and the following chapter.

8.4.5.3 Visualisation

With a 2-dimensional objective space, sets can easily be visualised as 2D graphs. With a 3-dimensional objective space, sets can adequately be visualised as 2D projections of 3D graphs. With a 4-dimensional objective space, sets can, with some effort, be visualised as 2D projections of 3D graphs with a 4th colour dimension. Beyond 4 dimensions it becomes difficult to visualise solution sets. A set could be shown in multiple 2D graphs of subspaces of the larger objective space, but it is impractical to cross-reference solution points between the graphs to identify features in higher dimensions.

The application of visualisation is therefore highly restricted to particular multiobjective problems but, where it is applicable, visualisation is a highly effective method of subjectively comparing solution sets. Subjective comparison need not be disregarded, considering the context in which it is used: it is most useful when objective comparison – i.e., Pareto comparison – fails to differentiate incomparable sets.

Further visualisation techniques can be found in [27, 66].
8.4.5.4 Hypervolume
The dominated hypervolume [117] of a set is an unary metric that measures the total volume of objective space (bounded by a given point) dominated by all solutions in a solution set. Fleischer’s algorithm [56] is an efficient implementation for calculating the hypervolume of a set. Calculating the hypervolume requires a bounding point, somewhere behind the solution set in objective space, so that the amount of space dominated is finite. In maximisation problems, the bounding point might naturally be the origin of objective space. In minimisation problems, it might naturally be the maximum possible value for each objective function. Otherwise, if there is no bounding point suggested by the problem description, a point must be chosen arbitrarily (and the same point must be used when measuring all sets).

Hypervolume can be understood intuitively as measuring both proximity to the Pareto front and distribution across the Pareto front. A set closer to the Pareto front will dominate more objective space than a set further away, all else being equal. Less obviously, and more generally, a set yielding a more accurate dominance approximation (see section 8.4.3.4) will dominate more objective space than a set yielding a less accurate dominance approximation. The accuracy of the dominance approximation depends on distribution of the solutions (see for example Figure 8.11).

Note that the most accurate distribution of points in a finite solution set is not necessarily the most uniform distribution: it is the distribution that most closely matches the Pareto front to the faces of the hypercuboids dominated by the solution set, and this is what hypervolume measures.
The unary hypervolume metric, $I_h$, is the only currently known unary metric that satisfies both of the following [118]:

$$I_h(A) > I_h(B) \Rightarrow A \triangleright B$$

That is, $A$ having a greater hypervolume than $B$ means that $A$ cannot be worse than $B$ (it might be better, but it might also be incomparable). However, if $A$ is better than $B$, then it must have a greater hypervolume.

### 8.4.5.5 Coverage

Coverage [117] is a simple binary metric to compare two sets using Pareto dominance. $I_c(A, B)$ returns the fraction of solutions in $B$ that are dominated by or equal to (i.e., “covered” by) solutions in $A$. As a binary (rather than unary) metric, coverage can imply the following Pareto relationships:

$$I_c(A, B) = 1 \land I_c(B, A) = 0 \Rightarrow A \triangleright B$$
$$I_c(A, B) = 1 \land I_c(B, A) < 1 \Rightarrow A \triangleright B$$
$$I_c(A, B) < 1 \Rightarrow B \ntriangleright A$$
$$I_c(A, B) < 1 \land I_c(B, A) < 1 \Rightarrow A \parallel B$$

It is expected that solution sets from competing algorithms will commonly be incomparable in the Pareto sense. In these cases, coverage can (cautiously) be used to describe the degree to which one set is preferable to the other, but any conclusions must take into account the non-dominated solutions from each set.
8.4.5.6 Binary hypervolume
Binary hypervolume [57] is a binary metric that measures the volume of objective space dominated by one objective set but not by another. $I_{h2}(A, B)$ returns the hypervolume dominated by $A$ but not $B$. Binary hypervolume can imply the following Pareto relationships:

\[
(I_{h2}(A, B) > 0) \land (I_{h2}(A, B) = 0) \Rightarrow A \succ B
\]

\[
(I_{h2}(A, B) > 0) \Rightarrow A \not\succ B
\]

\[
(I_{h2}(A, B) > 0) \land (I_{h2}(A, B) > 0) \Rightarrow A \parallel B
\]

Binary hypervolume cannot imply dominance ($A > B$) because hypervolume can be increased by improving proximity to the Pareto front (i.e., dominance) or by improving distribution. The most it can imply is better-than ($A \succ B$).

8.5 Application
This section describes the EA applied to the inverted pendulum to generate the qualitative results in section 8.3 and the quantitative results in the following section.

The aim is to investigate the differences between two diversity measures, hypervolume and distance-to-neighbours, isolated as much as is possible from all other factors. The aim is not to outperform any existing algorithm in the literature. It is hoped that any observations may be generalised to apply to many existing or future algorithms. Therefore a single evolutionary algorithm is used for all experiments, with only the diversity measure changing.
8.5.1.1 Overview

The EA used is based on NSGA-II [28].

Each solution has the following properties:

- Extended-dominance rank, $R_E$
- Objective-space diversity, $D_O$

The EA maintains a population of fixed, even, size $P$ from generation to generation. At each generation:

1. Selection for reproduction: $P/2$ pairs of parents are selected, with replacement, using binary tournament selection. Solutions are compared first by $R_E$ and, if tied, then by $D_O$.

2. Each pair of parents generates a pair of children using crossover and mutation suitable for the decision variables (given in the table of parameters). This process generates $P$ children, which are evaluated and added to the population. Duplicate solutions are removed from the population. After this stage, parents and children in the expanded population compete equally for survival.

3. All of the solution properties are calculated relative to the entire expanded population.

4. Selection for survival to the next generation: the population is sorted (first by $R_E$ and second by $D_O$). The best $P$ are carried over to the next generation.

8.5.1.2 Rank

Solutions are assigned a rank, $R_E$, using the non-dominated ranking procedure, and replacing basic Pareto dominance with extended-dominance (see section 8.4.2.2). The extended-dominance relation incorporates constraint satisfaction into the ranking
procedure. This is similar to NSGA-II in [31], but NSGA-II uses a sum of normalised
constraint violations to compare infeasible solutions, whereas this algorithm uses Pareto
dominance in constraint space to compare infeasible solutions.
Ranking is performed on the combined population of the previous generation + children.
A solution’s rank may change from the previous generation due to the presence of new
children that may dominate it. All solutions must be re-ranked every generation.

8.5.1.3 Diversity
Solutions are assigned a diversity using either the uniquely dominated hypervolume
measure or the distance-to-neighbours measure (see section 8.4.3). The hypervolume
measure is the same as Fleischer’s algorithm in [56], and the distance measure is the
same as NSGA-II [28]. The two measures are compared in the experiments.
As with ranking, diversity must be calculated every generation for the combined
population of the previous generation + children.

8.5.1.4 Cooperative coevolution
This algorithm is unable to make use of the cooperative coevolution described in chapter
7. Cooperative coevolution uses multiple populations, each population evolving part of
the solution. A partial solution cannot be evaluated on its own, so when individuals in one
population are evaluated they must be combined with a single representative individual
from each other population. It is not clear which solution should be chosen to be
“representative” of solutions in a multiobjective problem, for which a diverse range of
solutions is expected.
For this work, a single-population EA is used.
8.6 Experiments

The CMEA described above was run 10 times using the uniquely dominated hypervolume diversity measure, and 10 times using the distance-to-neighbours diversity measure. The following parameters were used for all runs:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>population size</td>
<td>100</td>
</tr>
<tr>
<td>crossover rate</td>
<td>0.6</td>
</tr>
<tr>
<td>mutation magnitude</td>
<td>+/- 5</td>
</tr>
<tr>
<td>mutation rate</td>
<td>0.05</td>
</tr>
<tr>
<td>tournament size</td>
<td>2</td>
</tr>
<tr>
<td>generations</td>
<td>1000</td>
</tr>
</tbody>
</table>

As the aim of these experiments is to investigate the differences between specific diversity measures and not to produce strictly optimal solutions, no parameter tuning was performed.

8.6.1 Progress of evolution

All runs followed a similar progression of evolution. The following is a description of one randomly selected run.

The first 70 generations are spent satisfying the single constraint without regard to objective space – the objective scores do not provide selection pressure when the search is for feasibility. As such, the early generations are similar to single-objective optimisation, since a solution’s non-dominated rank corresponds to its single constraint score. The single elite, non-dominated solution wanders randomly through objective space from generation to generation as the constraint score is minimised.
Figure 8.12. Progress of constraint satisfaction of best solution during early generations.

Figure 8.13. Random wander (starting top-right) of best solution through objective space during early generations.

Around generation 80 sees the appearance of the first feasible solutions. Up to around generation 150, diversity is relatively unimportant as the population becomes feasible and collapses into a single non-dominated rank. After the entire population is in one non-dominated rank, diversity influences selection much more strongly to distribute the population across the Pareto front.

Figure 8.14. Non-dominated solutions found after 80 (left), 110 (centre), and 1000 (right) generations. 3d space represents $f_1$, $f_2$, and $f_3$. Grey-scale intensity represents $f_4$. 

207
8.6.2 Comparing diversity measures

8.6.2.1 Single run

Figure 8.15 shows the final non-dominated solution sets from one randomly selected run each of uniquely dominated hypervolume and distance-to-neighbours.

For all runs, the entire final population – the final solution set – was non-dominated. However, a solution from one solution set may be dominated by a solution from the other solution set. Unfortunately it is impossible to identify those solutions visually from the above graphs, since it cannot be seen which point in \( f_1 \) vs. \( f_2 \) maps to which point in \( f_3 \) vs. \( f_4 \), and therefore a point that *appears* to be dominated in one graph may not be in the other graph. This will be more clearly illustrated later.
Figure 8.15. One typical run each of hypervolume and distance diversity metrics. Two views of the same data: $f_1$ vs. $f_2$ and $f_3$ vs. $f_4$.

As single runs are subject to variation, little can be concluded from this comparison. It is intended simply to illustrate the extent of coverage of a typical run.
8.6.2.2 Cumulative solution sets

Figure 8.16 shows the cumulative solution set from all 10 runs each of uniquely dominated hypervolume and distance-to-neighbours.

Although all individual runs produced entirely non-dominated solution sets, the cumulative solution sets include some dominated solutions: those that were non-dominated in their local population but dominated by a solution found in some other population. And similarly to above, it is impossible to tell from these graphs which solutions are truly dominated and which only appear to be dominated in one graph. Therefore it is impossible to draw conclusions about best-case performance of the uniquely dominated hypervolume and distance-to-neighbours diversity measures.

However, it is possible to draw some conclusions about the general behaviours of the two measures. By comparing the cumulative solution sets (Figure 8.16) to the single solution sets (Figure 8.15) shown above, it is observed that the single hypervolume solution set is broadly representative of the cumulative solution set. Such was the case for all single hypervolume sets. In contrast, the single distance solution set is not representative of the cumulative solution set: it misses the regions corresponding to low $f_4$ values.

Nevertheless, the cumulative solution sets show that distance-to-neighbours is capable of finding solutions optimal in $f_3$ vs. $f_4$ that uniquely dominated hypervolume cannot.
Figure 8.16. Cumulative solution set from 10 runs each of hypervolume and distance diversity metrics. Two views of the same data: $f_1$ vs. $f_2$ and $f_3$ vs. $f_4$
8.6.2.3 Universally non-dominated set

Figure 8.17 shows the universally non-dominated solution set. This is the set of solutions that are non-dominated by all solutions from both cumulative sets.
Different glyphs are used to divide the set into solutions contributed by hypervolume cumulative set and solutions contributed by distance cumulative set, for the purposes of comparison, but unlike the previous figures this is one single set of mutually non-dominated solutions, regardless of glyph.

By comparing the universally non-dominated solution set (Figure 8.17) to the cumulative solution sets (Figure 8.16) shown above, it is observed that the regions where the cumulative solution sets overlap tend to be strongly dominated by the hypervolume-contributed solutions. Most of the non-dominated distance-contributed solutions are in regions where there were no hypervolume solutions at all in the cumulative set.

8.6.2.4 Comparison metrics
Table 8.4 shows a number of metrics comparing the solutions obtained from runs using uniquely dominated hypervolume as the distance measure, and runs using distance-to-neighbours as the distance measure.

Set $A$, for hypervolume, is the set of non-dominated, unique solutions from the cumulative hypervolume solution set. The basic cumulative solution set (Figure 8.16) was simply the union of solution sets from 10 independent runs, and as such contained duplicate and dominated solutions. A set containing duplicate and dominated solutions is inappropriate for the following comparison metrics, and so the basic cumulative solution set is refined into the non-dominated cumulative solution set, $A$.

The same is done for the distance measure, producing set $B$.

The non-dominated cumulative solution sets are used, instead of the smaller solution sets from independent runs, in order to smooth out variance between runs whilst avoiding an exhaustive number of comparisons between every combination of 2 independent runs.
The bounding point for calculation of hypervolume is the maximum possible value for each objective score. As each objective score is the mean value of the corresponding state-variable, the maximum value for each objective is the corresponding bound on state-space \((1, 1, \pi/6, 3)\).

Table 8.4. Comparison metrics

<table>
<thead>
<tr>
<th>Metric</th>
<th>Hypervolume ((A))</th>
<th>Distance ((B))</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unary hypervolume, (I_h)</td>
<td>(I_h(A) = 1.0333)</td>
<td>(I_h(B) = 0.900797)</td>
</tr>
<tr>
<td>Binary hypervolume, (I_{h2})</td>
<td>(I_{h2}(A,B) = 0.000155503)</td>
<td>(I_{h2}(B,A) = 0.000422228)</td>
</tr>
<tr>
<td>Num. of non-dominated in cumulative set ((/\text{max}))</td>
<td>693 / 1000</td>
<td>537 / 1000</td>
</tr>
<tr>
<td>Num. of universally non-dominated ((/\text{cumulative}))</td>
<td>680 / 693</td>
<td>125 / 537</td>
</tr>
<tr>
<td>Coverage, (I_c)</td>
<td>(I_c(A,B) = 0.767)</td>
<td>(I_c(B,A) = 0.019)</td>
</tr>
</tbody>
</table>

The first thing to note, which was clear from visual observation, is that the hypervolume and distance sets are incomparable; neither is better than the other from a Pareto comparison. This is true for all solution sets from individual runs, as well as the non-dominated cumulative solution sets.

It is unsurprising that the hypervolume set outperforms the distance set on the unary hypervolume metric. Notice, however, that although the hypervolume set has a greater unary hypervolume score than the distance set, the distance set has a greater binary hypervolume score than the hypervolume set – the amount of space that the distance set
dominates but the hypervolume set does not, is greater than the amount of space that the hypervolume set dominates but the distance set does not.

The relative scores on unary hypervolume and binary hypervolume support the visual observations: the hypervolume set dominates the distance set in almost every region that they overlap (giving the hypervolume set a greater unary hypervolume score), but the distance set finds some small regions that the hypervolume set does not (giving the distance set a greater binary hypervolume score). This is because in the overlapping regions, the hypervolume set dominates the distance set, but not by much – the distance set is close behind, limiting the amount of space that the hypervolume set uniquely dominates. In contrast, in the non-overlapping regions the distance set finds a few solutions that the hypervolume set is not even close to, giving those few non-overlapping solutions a large amount of uniquely dominated space.

The coverage comparisons also confirm what was observed visually: that the majority of the distance set is dominated by the hypervolume set. In particular, the fraction of the distance set not dominated by the hypervolume set (1-\(I_c(A,B) = 0.233\)) dominates proportionately very little of the hypervolume set in return (\(I_c(B,A) = 0.019\)). This indicates that the 125 universally non-dominated solutions from the distance set are non-dominated not because they dominate solutions from the hypervolume set, but because they find regions of the Pareto front that hypervolume does not, and are therefore not in competition.
8.7 Conclusions

8.7.1 Design

8.7.1.1 Hierarchical rulebases
A hierarchical rulebase structure was unsuitable for this study due to the way in which a hierarchical rulebase biases the behaviour of a fuzzy system, and this study examined the range of control strategies possible for the inverted pendulum. However, hierarchical rulebases are not incompatible with multiobjective optimisation in general. The design of a hierarchical rulebase requires an understanding of how the design will affect the fuzzy system (as found in [97, 113]; this understanding is necessary simply to achieve any acceptable fuzzy system at all. Given that the practitioner using a hierarchical structure will have this understanding, they will understand that it will bias the system towards certain behaviours, and presumably this bias will be acceptable or even desirable. Multiobjective optimisation can still be used within the scope allowed by the hierarchical design.

8.7.1.2 Objectives and constraints
Objectives and constraints must be extracted from the problem description.

- The problem description must unambiguously specify what constitutes feasibility, and how feasibility is different from optimality: section 8.2.2 demonstrated that, when all optimal solutions are inherently feasible (i.e., when constraints do not restrict the Pareto front), constraints are redundant.
- Objectives should be able to differentiate feasible solutions to guide the search to optimality, and constraints should be able to differentiate infeasible solutions to guide the search to feasibility. See section 8.2.2.
Objectives should measure non-commensurable qualities of the solution. In principle, Pareto optimisation will correctly handle commensurable measurements implemented as separate objectives (for example, different test-cases represented by different objectives) since the dominance relation will still favour solutions that optimise all objectives. However, it will not be as efficient since adding objectives adds more ways for solutions to be non-dominated during the optimisation process. See section 8.2.3.

8.7.1.3 Cooperative coevolution
Cooperative coevolution, used in chapter 7, was not used in this study because it was not obvious how to select a representative individual from each sub-population that would be “representative” of the diverse range of optimal solutions expected from multiobjective optimisation.

In chapter 7 and the works cited therein, cooperative coevolution was used for single-objective optimisation, in which it is expected that the population should converge to a single optimal solution. In such a case it is reasonable to assume that differences in fitness between individuals (partial solutions), combined with the same representative, should be indicative of the difference in quality between the individuals since they are assumed to be evolving towards the same global optimum.

However, for a multiobjective optimisation problem, no matter what partial solution is chosen as representative of other sub-populations, it cannot be assumed that the difference in fitness between individuals combined with that representative is indicative of their quality, since they are expected to evolve towards different regions of the Pareto front, and the representative itself will be biased to a particular region.
In summary, the concept of a single representative individual appears to be incompatible with the concept of a set of non-dominated solutions identifying the Pareto front, which contains many solutions with deliberately diverse characteristics. Finding a way to overcome this difficulty is a possible, different, future avenue of research.

(Another reason for not using cooperative coevolution, in this study, was that cooperative coevolution requires a modular decomposition of whole solutions into partial solutions. Hierarchical rulebases were deemed inappropriate – for a different reason – and so a single-layer rulebase was used and there was no modular decomposition. However, this reason is incidental, as it does not apply to cooperative coevolution and multiobjective optimisation in general.)

8.7.2 Hypervolume vs. distance-to-neighbours

8.7.2.1 Theoretical results
Pareto comparison revealed that all sets produced by the uniquely dominated hypervolume and distance-to-neighbours diversity measures were incomparable to one another: all contained at least one non-dominated solution. However, some conclusions may be drawn regarding the general behaviour of the two diversity measures.
By comparing the universally non-dominated solution set to the cumulative solution sets (Figure 8.17, Figure 8.16 and Table 8.4), it is observed that the regions where the cumulative solution sets overlap are almost entirely dominated by the hypervolume measure. Most of the non-dominated distance-to-neighbours solutions are in regions where there were no hypervolume solutions at all in the cumulative set: distance-to-neighbours was able to find some regions of the Pareto front that hypervolume could not.
It is concluded that the hypervolume measure produces a solution set with closer proximity to the Pareto front, whereas the distance-to-neighbours measure produces a solution set with greater diversity. This was predicted in the discussion of diversity measure in section 8.4.3.

At first it may be counter-intuitive that a diversity measure should be able to influence proximity to the Pareto front (as opposed to distribution along the Pareto front). However, this conclusion makes sense when it is considered that uniquely dominated hypervolume reflects both distance from neighbouring solutions and proximity to the Pareto front, as both will increase the uniquely dominated hypervolume. Therefore, even when the entire population has collapsed into one rank and the EA can only differentiate solutions by their diversity measure, the hypervolume measure still provides some selection pressure to move closer to the Pareto front.

It should be noted that rank still provides the strongest selection pressure to move towards the Pareto front, and the algorithm using the distance-to-neighbours diversity measure was not much worse. Since rank alone is effective at moving the population towards the Pareto front, one might argue that a diversity measure that finds more regions of the Pareto front is more useful than a diversity measure that further improves the proximity. However, this must ultimately be the decision of the practitioner; these results cannot conclude that one measure is strictly superior to the other.
8.7.2.2 Pragmatic results

Whilst hypervolume produced solution sets with advantages discussed above, it has two important disadvantages when practical implementation is considered:

- Fleischer’s algorithm is more complicated to implement, and computational complexity is significantly higher, at $O(m^2 n^3)$ where $m$ is the dimensionality and $n$ is the number of points [56]. This cost is acceptable when hypervolume is used to compare solution sets, as the computation is only performed once per experiment, and is more relevant to research than to practical application. However, for high dimensionalities this cost may be unacceptable when hypervolume is used to compare individual diversity, as this must be calculated for every solution at every generation. (This is found to be the case in the crop rotation problem, in chapter 9).

- The difficulty of dealing with solutions sharing exactly the same position in one or more dimensions. As the hypervolume is effectively the product of distances (as opposed to the sum of distances in distance-to-neighbours), the uniquely dominated hypervolume of any solution sharing the same objective score as any other solution in any dimension, regardless of its distance to solutions in other dimensions, will be 0.

The problem of dealing with solutions sharing the same objective score can be alleviated, as described in section 8.2.2, by ensuring that there are no objectives for which solutions are expected to have the same objective score. Careful consideration may reveal that such objectives are more properly formulated as constraints, and not objectives. After all, Pareto-based multiobjective optimisation is used when a trade-off is expected to generate
solutions of varying objective scores; the concept of Pareto optimality is *inappropriate* when applied to objectives where a certain score is expected of *all* solutions.
Chapter 9. A Novel Constrained Multiobjective Evolutionary Algorithm

9.1 Introduction

The previous chapter described a simple application of constrained multiobjective optimisation to the inverted pendulum problem. In that chapter, the techniques used were taken from the literature for the purposes of demonstrating the use of constrained multiobjective optimisation.

This chapter re-examines some of the basic techniques of constrained multiobjective optimisation, questions their appropriateness, and ultimately proposes a novel constrained multiobjective EA (CMEA) derived from the lessons learnt.

This chapter is broadly organised in two halves:

1. The first half of this chapter implements a novel CMEA (called Blended Space Evolutionary Algorithm (BSEA)) derived from the literature. Section 9.2 - 9.3 gives a background to constrained multiobjective optimisation. Section 9.4 describes BSEA. Section 9.5 - 9.6 examines the behaviour of BSEA.
2. The second half of this chapter reflects upon the desired behaviour of a CMEA, and refines BSEA into Blended Rank Evolutionary Algorithm (BREA), designed to address the shortcomings observed in BSEA. Section 9.7 describes insights into the difficulties. Section 9.8 describes BREA. Section 9.9 gives experimental results from BSEA, BREA, and NSGA-II. Section 9.10 concludes the chapter.

This chapter is distinguished from the others in that the test problems are not fuzzy systems. This chapter is focussed entirely upon constrained multiobjective optimisation. This chapter is also distinguished from the others in that the aim is to develop an entire novel algorithm that outperforms an existing, widely used algorithm, rather than investigating very specific questions as in previous chapters.

9.2 Background

Many real world problems can be formulated as constrained multiobjective optimisation problems. This chapter uses the constrained multiobjective problem formulation given in section 3.6:

\[
\begin{align*}
\text{minimise} & \quad f_i(x), \quad i = 1, \ldots, M, \\
\text{subject to} & \quad g_j(x) \leq 0, \quad j = 1, \ldots, N.
\end{align*}
\]  

(3.5)

Unconstrained multiobjective EAs have been extensively studied; see for example the survey [14]. Constrained multiobjective EAs have received growing attention recently; see for example the survey [13], and specific multiobjective EAs [28, 81, 82].
This chapter uses the basic Pareto-based multiobjective optimisation techniques introduced in the previous chapter:

- Pareto dominance (section 8.4.1.1).
- Approximating the Pareto front (section 8.4.1.2).
- Dominance ranking (section 8.4.2).
- Diversity (section 8.4.3).
- Comparing solution sets (section 8.4.5).

### 9.2.1 Spaces

A constrained multiobjective EA can operate in multiple homomorphic spaces: decision-variable space, objective space, and constraint space. The objective functions $f_1, \ldots, f_M$ map from decision-variable space to objective space; and the constraint functions $g_1, \ldots, g_N$ map from decision-variable space to constraint space. No space can be searched independently of the others.

The term **space** refers to a (usually infinite and continuous) space defined by one of the above sets of dimensions. The term **region** refers to a discrete region within a space, for example bounded by constraint functions.

### 9.2.2 Constrained Pareto front

Multiobjective EAs are designed to find a set of non-dominated solutions that approximate the true Pareto front. The algorithm should produce an approximation set that has close proximity to the Pareto front and good distribution of points across the front.

In constrained multiobjective problems, sections of the unconstrained Pareto front will usually be infeasible. The aim is therefore to find the constrained Pareto front, which is
the set of all non-dominated feasible solutions. If there are multiple discrete feasible regions of objective space, then there may be false constrained Pareto fronts. False constrained Pareto fronts are fronts that are non-dominated in their local discrete region, but that are dominated by other discrete feasible regions. The true constrained Pareto front is non-dominated by all regions.

9.2.3 Dominance ranking
Section 8.4.2 introduced Pareto dominance ranking, a method for sorting the population into ranks of mutually non-dominated solutions. For this chapter it is important to note that the number of ranks expected in a population is inversely proportional to the dimensionality of the space in which Pareto dominance is calculated. With more dimensions there are more ways in which any two solutions can be non-dominated with respect to each other, leading to more mutually non-dominated solutions and therefore fewer ranks. This becomes an issue in high-dimensional spaces, when dominance ranking may become too coarse-grained to differentiate solutions and provide selection pressure towards the Pareto front.

9.2.4 Constraints
Constrained EAs can handle constraints in many ways: the EA’s variation operators (crossover and mutation) can be designed to avoid creating infeasible solutions; the EA can introduce an additional operator to repair infeasible solutions; infeasible solutions can be discarded; or infeasible solutions can be retained in the population and used in the search process.

In many problems it may be necessary to retain infeasible solutions for use in the search process, as finding any feasible solution may be difficult. Of these EAs, some algorithms
do not treat constraint scores as a multidimensional space and simply sum or min-max constraint violations (collapsing the space into one dimension) or apply penalties back onto objective scores (distorting the objective space). Other algorithms do treat constraint space as a space in its own right, in which Pareto dominance ranking can be performed. Early use of Pareto-based techniques in constrained problems was in single-objective problems, where Pareto-based techniques were used solely to navigate constraint space (see for example [12] and the citations therein).

The notions of Pareto dominance and dominance ranking are applied to constraint space for constraint satisfaction in the same way as to objective space for objective optimisation.

9.2.5 Constraint satisfaction vs. objective optimisation
Constrained, multiobjective EAs (CMEAs) need to manage the conflicting goals of objective optimisation and constraint satisfaction. Often neither goal will be trivial to achieve.

This work presents the reconciliation of objective optimisation and constraint satisfaction as the main challenge facing any CMEA, and characterises CMEAs by the heuristic used to relate decision-variable space, objective space, and constraint space.

9.2.5.1 Extended dominance
Many algorithms use some form of an extended dominance relation that bridges together objective space and constraint space to form an extended space in which to perform one ranking. The extended dominance relation replaces the basic Pareto dominance relation. It is still a binary relation to compare two solutions, like the Pareto dominance relation, so the dominance ranking procedure remains unchanged. The only
difference is that, whereas the Pareto dominance relation compares solutions in one space – either objective space or constraint space – the extended dominance relation compares solutions in both spaces.

The extended dominance relation is:

1. If both solutions are feasible, dominance is measured in objective space.
2. If one solution is feasible and the other is infeasible, the feasible solution dominates.
3. If both solutions are infeasible, dominance is measured in constraint space.

The effect of this dominance relation is to ensure that a non-dominated rank is either all feasible or all infeasible, and that all feasible ranks are better than all infeasible ranks.

Precursors to the extended dominance relation appeared in [26] and [53]: they were for single-objective problems, so there was no use of Pareto dominance in rule 1; and they either summed [26] or min-maxed [53] the constraints in rule 3.

The algorithms were extended in [31] and [52] to solve multiobjective problems: they now used Pareto dominance in rule 1; but they still either summed [31] or min-maxed [52] the constraints in rule 3.

Whereas [31] and [52] used Pareto dominance in rule 1, [15] introduced a limited form of Pareto dominance in rule 3 (but used a single-objective problem, so rule 1 did not use Pareto dominance). In rule 3, infeasible solutions were only compared using Pareto dominance inasmuch as a globally non-dominated solution was selected over a globally dominated solution. If both solutions were globally non-dominated, or both were globally dominated, they were compared by min-maxing constraints.
In this work, all of the above techniques have been amalgamated to give the general-case Pareto-based extended dominance relation, above. What all versions of the extended dominance relation have in common, regardless of how rules 1 and 3 are implemented, is that constraint satisfaction is always prioritised over objective optimisation: this is guaranteed by rule 2.

**9.2.5.2 Combined space**
An alternative to extended dominance is combined space [82], in which the constraint dimensions 1,...,$N$ are added to the objective dimensions 1,...,$M$ to create a single $M+N$ dimensional space in which Pareto dominance can be evaluated using objective scores and constraint scores together. Combined space is similar to extended space in that only one ranking is performed. It is dissimilar in that an infeasible solution with high objective scores can be non-dominated with respect to a feasible solution with low objective scores (however, an infeasible solution can never dominate a feasible solution as a feasible solution by definition must have at least one constraint score better than an infeasible solution). The disadvantage is that only a very small volume of this combined space represents feasible solutions, where the values for all $N$ constraint dimensions are 0. To address this, some preference is given to carrying over feasible solutions from generation to generation.

**9.2.5.3 Stochastic ranking**
Another alternative is stochastic ranking [85]. Solutions are ranked using a stochastic bubble sort. Bubble sort iteratively ranks a population by comparing a solution with its immediately better-ranked neighbour and swapping its rank with its neighbour if a comparison indicates that the solution is in fact superior to its neighbour. Ordinarily this
comparison would be deterministic – for example using the extended dominance relation – but in stochastic ranking each comparison has a set probability of being performed only in objective space. The result is that sometimes an infeasible solution can attain a better rank than a feasible solution, if it dominates the feasible solution in objective space and the comparison was stochastically performed in objective space.

In [85], it was shown that stochastic ranking performed better than deterministic (i.e., extended dominance) ranking in single-objective problems using a penalty approach to measure constraint satisfaction. The work was extended in [84] to use Pareto dominance to measure constraint satisfaction, and stochastic ranking was still found to perform better than deterministic (extended dominance) ranking.

In contrast to extended dominance, which rigidly prioritises constraint satisfaction over objective optimisation, stochastic ranking allows objective optimisation to influence the search for feasibility.

### 9.2.5.4 Blended space

**Blended space** is similar to stochastic ranking in that it allows objective optimisation to influence the search for feasibility, unlike extended dominance. However, although blended space and stochastic ranking are similar in aim, they are quite different in implementation.
In an algorithm using blended space, every solution is assigned two Pareto dominance ranks: one calculated in objective space, $R_O$, and the other in constraint space, $R_C$. The two ranks are weighted and summed to give one blended rank which can be used to compare solutions for reproductive selection and survival. The blended rank, $R_B$, is defined as:

$$ R_B = \alpha R_O + (1- \alpha)R_C \quad (9.2) $$

where $\alpha = \frac{\text{num feasible solutions}}{\text{population size}}$.

The parameter $\alpha$ acts as a weight in an adaptive feedback loop that controls the behaviour of the search, by moving the emphasis between objective optimisation ($R_O$) and constraint satisfaction ($R_C$).

$\alpha$ varies adaptively so that if the population is entirely infeasible ($\alpha = 0$) all emphasis is placed on minimising the constraint rank – i.e., searching for a feasible solution – thereby increasing the number of feasible solutions and increasing $\alpha$. At the other extreme, if the population becomes entirely feasible ($\alpha = 1$) all emphasis is placed on minimising the objective rank – i.e., searching for the optimal solution – likely increasing the number of infeasible solutions and decreasing $\alpha$. $\alpha$ oscillates somewhere in between during the run of the algorithm. The effect is that the population will straddle the constraint boundaries that separate feasible region from the unconstrained optimum.

Blended space was proposed in [4] for single-objective optimisation, and $R_O$ was therefore a one-dimensional ranking that did not require Pareto dominance. $R_C$, however, was calculated using Pareto dominance ranking.

Blended space is similar to an adaptive penalty function [40] in that it makes use of a weight calculated from the fraction of feasible solutions in the population. However,
adaptive penalty functions use a sum or max of constraint violations, and are applied to a single objective function; Pareto-based optimisation is used for neither objectives nor constraints.

The term “blended space” is proposed by this work to refer to the numerical aggregation of independent measures from multiple spaces, for example, rank in both objective space and constraint space.

9.3 Motivation for research

9.3.1 Limitations of extended dominance

The extended dominance relation has proved to be very effective in many algorithms. The extended dominance relation causes the search to move through four stages, determined by the three rules in (9.1), restated here:

1. If both solutions are feasible, dominance is measured in objective space.

2. If one solution is feasible and the other is infeasible, the feasible solution dominates.

3. If both solutions are infeasible, dominance is measured in constraint space.

The search will (usually) begin with no feasible solutions, and the population will evolve subject only to rule 3. In this stage, the algorithm is performing a search in constraint space for any feasible solutions.

As soon as the first feasible solution is found, rule 2 will come into effect and place extreme selection pressure on any feasible solutions. In this stage, rule 2 drives the transition of the population from infeasibility to feasibility. This transition will be facilitated mostly by selection of the new feasible solutions, rather than the mutation of
old infeasible solutions, since selection pressure is strong and mutation is aimless. The population “jumps” into the feasible region around the first feasible solutions.

Rule 1 governs the feasible solutions in the population. When rule 2 causes the entire population to become feasible, the population evolves subject only to rule 1. In this stage, the algorithm is performing a search in objective space for the constrained Pareto front. Rule 2 will cause any infeasible solutions generated during this stage of the search to be immediately discarded.

Since diversity is secondary to dominance, it will only provide strong evolutionary pressure when the entire population is in one rank, since when solutions cannot be differentiated by rank, they are differentiated by diversity. This will usually only occur when the population is entirely feasible and pushing up against the constrained Pareto front (rule 2 prevents the population from crossing the constrained Pareto front). At other times the population will be strung out moving through constraint space and/or objective space.

Therefore, the typical search progression of an algorithm using extended dominance is:

1. move through the infeasible region to a feasible region,
2. jump into that feasible region,
3. move through that region to the constrained Pareto front, and finally
4. spread out along that constrained Pareto front.

The result is a single, well-distributed rank of non-dominated solutions.

Whilst this usually works well, it is potentially a problem when the constrained Pareto front lies across multiple discrete feasible regions: the algorithm may converge to the first feasible region it finds. Any new solutions that are subsequently generated anywhere in
infeasible regions are discarded due to rule 2. The only remaining mechanism to escape a discrete feasible region is to generate a new solution by just one crossover and mutation operation that “jumps” into another feasible region and is non-dominated by the existing solutions. Depending on the crossover and mutation rates and, more importantly, the mutation magnitude, it may be highly unlikely to generate a new solution in a non-dominated feasible region in a single step.

![Figure 9.1](image-url)

**Figure 9.1.** A two-objective, two-constraint problem based on CTP (see section 9.5). Infeasible regions are shaded and are darker where constraints overlap. Feasible regions are white.

Figure 9.1 shows NSGA-II, an extended dominance based algorithm, trapped along a false constrained Pareto front. Except for finding the false constrained front rather than the true constrained front, the extended dominance based algorithm’s solution set has desirable characteristics: it is very close to the (false) Pareto front and has an excellent distribution.
This behaviour was observed in both NSGA-II and a version of the algorithm presented in this chapter that used the extended dominance relation.

### 9.3.2 Limitations of blended space

In contrast to the rigid constraint-handling of extended dominance, blended space has flexible constraint-handling that allows infeasible solutions to play a role in evolution throughout the entire search. Blended space algorithms promise the ability to cross large infeasible regions in small, evolutionary steps. An infeasible solution can trade off a bad rank in constraint space for a good rank in objective space, and the feedback loop controlled by $\alpha$ ensures that there is a balance of feasible and infeasible solutions in the population.

However, the single-objective blended space algorithm proposed in [4] has several limitations when applied to a multiobjective problem. Figure 9.1 shows the typical behaviour of a straightforward multiobjective implementation of [4]. It successfully finds the feasible regions in which the true constrained Pareto front lies, but coverage is poor. The poor coverage is attributable to multiple interacting causes.

The proximity to the Pareto front is poor because the blended space algorithm actively avoids collapsing to a single non-dominated rank. The feedback loop will keep a mix of feasible and infeasible solutions in the population, a continuous trade-off between constraint satisfaction and objective optimisation that gives rise to a large number of ranks.

The distribution of the solutions is poor because, as with extended dominance based algorithms, diversity is secondary to dominance. Diversity can only differentiate
solutions of the same rank, and so the more ranks, the less the selection pressure on diversity.

9.3.2.1 Selection for survival
Furthermore, in multiobjective algorithms, whilst selection for reproduction is usually stochastic, selection for survival is usually deterministic (elitist). Elitist survival encourages the formation of a single non-dominated rank, and since this rank is then differentiated by diversity, elitism in multiobjective algorithms avoids the association with premature convergence in single-objective algorithms. However, only the last surviving rank will be differentiated by diversity in elitist survival. This is not a problem for extended dominance algorithms that readily collapse to a single rank. It is a problem for blended space algorithms that avoid collapsing to a single rank: in elitist survival only the worst rank will be differentiated by diversity, severely limiting the effectiveness of diversity.

Frustratingly, stochastic selection for survival does not solve this problem of deterministic selection for survival. Stochastic selection does allow diversity to differentiate solutions in all ranks, not just the worst rank. However, stochastic selection even further disrupts consolidation of ranks, creating more ranks and still managing to ultimately reduce the scope of diversity to differentiate solutions.

9.3.3 Constraint-handling strategies
As previously stated, this work describes CMEAs in the context of how they reconcile the often conflicting goals of constraint satisfaction and objective optimisation. This heuristic strategy defines how the CMEA will measure and use information from all three spaces
identified by the constrained multiobjective problem formulation: decision-variable space; objective space; and constraint space.

The extended dominance strategy is to rigidly prioritise constraint satisfaction, then objective optimisation, then objective diversity. This is exactly the prioritisation that is desired in a final solution set, but this rigid prioritisation leaves extended dominance susceptible to local optima during evolution.

The blended space strategy is to adaptively trade off constraint satisfaction for objective optimisation, placing a greater emphasis on the strategy’s effect on evolution rather than on the final solution set.

The following sections describe the implementation of two blended space CMEAs, and compares their behaviour and performance to an existing extended dominance algorithm, NSGA-II.

9.4 Blended space evolutionary algorithm

This section describes Blended Space Evolutionary Algorithm (BSEA), a multiobjective implementation of [4] designed to overcome the limitations identified in the previous section.

9.4.1 Overview

Each solution has the following properties:

- Blended rank, $R_B$ (from objective rank, $R_O$, and constraint rank, $R_C$)
- Extended-dominance rank, $R_E$
- Objective-space diversity, $D_O$
The EA maintains a population of fixed size $P$ from generation to generation. At each generation:

1. Selection for reproduction: $P/2$ pairs of parents are selected, with replacement, using binary tournament selection. Solutions are compared first by $R_B$ and, if tied, then by $D_O$.

2. Each pair of parents generates a pair of children using crossover and mutation suitable for the decision variables. This process generates $P$ children, which are evaluated and added to the population. Duplicate solutions are removed from the population. After this stage, parents and children in the expanded population compete equally for survival.

3. All of the solution properties are calculated relative to the entire expanded population.

4. Selection for survival to the next generation:
   
   i. A fraction of $P$ solutions are selected to form the final solution set, which will be carried over to the next generation.

   ii. The remainder of the population is sorted (first by $R_B$ and then by $D_O$). The best are carried over to the next generation to make up the $P$ solutions.

### 9.4.2 Ranking

$R_O$ and $R_C$ are both calculated using Pareto dominance in their respective spaces, and the ranking procedure in [38].

The ranking process in constraint space does not distinguish between feasible and infeasible solutions. The constraint functions $g(x)$ are allowed to be negative, with any
constraint \( g(x) \leq 0 \) indicating the constraint is satisfied. \( R_C \) differentiates even feasible solutions, with solutions that are most conservative being placed in the best rank.

Therefore, \( R_O \) and \( R_C \) are often in conflict. BSEA takes the approach of treating objective space and constraint space on equal terms, relying solely on \( \alpha \) to mediate the two ranks. The blended rank is calculated similarly to [4] (see section 9.2.5.4):

\[
R_B = \alpha R_O + (1- \alpha)R_C, \quad \text{where} \quad \alpha = \frac{\text{num feasible solutions}}{\text{population size}}.
\] (9.2)

However, in [4], \( R_O \) and \( R_C \) are integers directly corresponding to the rank number. This creates bias in the blended rank when the number of objective dimensions is different to the number of constraint dimensions, as the number of expected ranks is inversely proportional to the dimensionality of the space. To address this, in BSEA the rank values \( R_O \) and \( R_C \) are normalised to the interval \([0,1)\), causing the blended rank \( R_B \) to also fall in \([0,1)\).

### 9.4.3 Diversity

The diversity measure is calculated for each solution, grouped by objective space ranks. Diversity is always secondary to rank. Diversity is measured in objective space, internally to objective space ranks, because that is the measure that most accurately reflects what is desired from the final solution set.

The diversity of a solution is the uniquely dominated hypervolume (described in section 8.4.3.2) – the volume of objective space dominated by that solution but not dominated by any other solution in its rank. In practice hypervolume and NSGA-II’s distance-to-neighbours worked equally well on the problems tested in this chapter.
9.4.4 Final solution set

A limitation of the blended space algorithm is that the blended rank actively avoids collapsing the population into a single non-dominated rank. This behaviour is useful during evolution, but is unfortunate for the final decision-maker, since a significant part (in practice, most) of the population is useless as a final solution set: most solutions are either infeasible or dominated.

To address this BSEA recognises that, ultimately, extended-dominance-based algorithms produce a solution set with desirable properties: an entire population of feasible, non-dominated solutions, distributed evenly by diversity. Therefore, BSEA reserves up to a certain fraction, $r$, of the population to be the final solution set:

$$r = \frac{\text{current generation}}{\text{total generations}}. \quad (9.3)$$

This solution set is selected from the population using extended-dominance rank, $R_E$, instead of $R_B$. Only non-dominated solutions are selected. If there are too few non-dominated solutions to make up $r$, then no more are selected. If there are too many non-dominated solutions to fit into $r$, then those with the highest diversity are preserved, up to $r$. The remainder of the population is selected as normal, by $R_B$.

The fraction of the population reserved for the final solution set increases linearly over the run of the algorithm, until when the algorithm terminates the entire population becomes the final solution set.

This reservation is unaffected by $\alpha$, which merely influences the blended rank; however the reserved feasible solutions are still counted in the population and still affect the feedback loop by forcing $\alpha$ to rise steadily with $r$. This relationship was found to be complementary: as the number of feasible solutions forcibly increases, $\alpha$ will shift $R_B$.
more towards \( R_o \), allowing progressively more unconstrained exploration with the fewer remaining infeasible solutions.

Alternatively, if the number of generations is not known, \( r \) should be dependent on whatever other stopping criteria is used.

### 9.5 Constrained test problems

This section is primarily concerned with qualitatively assessing the ability of BSEA to cross infeasible regions too large to be reliably crossed by extended space algorithms, such as NSGA-II. However, this must be accomplished without sacrificing performance on problems with different characteristics. The first experiment tests BSEA and NSGA-II on the harder Constrained Test Problems, CTP-6, CTP-7, and CTP-8 found in [31]. The second experiment modifies CTP-8 to ensure that the population must cross infeasible regions to reach the true Pareto front.

In all experiments, for both algorithms, the fundamental evolutionary operators and parameters were kept identical. Operators and parameter values were taken from NSGA-II settings for CTP-6,7,8 [31]. For each experiment, each algorithm was run 10 times.

<table>
<thead>
<tr>
<th>Table 9.1. Algorithm Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Parameter</strong></td>
</tr>
<tr>
<td>population size</td>
</tr>
<tr>
<td>generations</td>
</tr>
<tr>
<td>crossover rate</td>
</tr>
<tr>
<td>mutation rate</td>
</tr>
<tr>
<td>distribution index for crossover</td>
</tr>
<tr>
<td>distribution index for mutation</td>
</tr>
</tbody>
</table>
CTP-6, 7, and 8 are all of the form:

Minimise:

\[
\begin{align*}
    f_1(x) &= x_1, \\
    f_2(x) &= (1.0 + x_2) \left( 1.0 - \frac{x_1}{\sqrt{1.0 + x_2}} \right),
\end{align*}
\]

with \( 0 < x_1 \leq 1 \); \( 0 < x_2 \leq 10 \),

and subject to constraint(s):

\[
g(x) \equiv \cos(\theta) \left( f_2(x) - e \right) - \sin(\theta) f_1 \\
\geq a \left| \sin \left( b \pi \left( \sin(\theta) \left( f_2(x) - e \right) + \cos(\theta) f_1(x) \right) \right) \right|^d
\]

where the parameters \( \theta, a, b, c, d, e \) are defined for CTP-6, 7, and 8 to give different shaped constraints (CTP-8 instantiates \( g \) twice with different parameters to give two constraints).

<table>
<thead>
<tr>
<th>Table 9.2. CTP Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \theta )</td>
</tr>
<tr>
<td>---------------------------</td>
</tr>
<tr>
<td>CTP-6</td>
</tr>
<tr>
<td>CTP-7</td>
</tr>
<tr>
<td>CTP-8 ( g_1 )</td>
</tr>
<tr>
<td>CTP-8 ( g_2 )</td>
</tr>
</tbody>
</table>

Figure 9.2 - figure 9.7 show one solution set from NSGA-II and one solution set from BSEA for each of the problems CTP-6, 7, 8, and restricted CTP-8. Only one solution set is shown for each algorithm because all solution sets for all runs for CTP-6, 7, and 8 were visually indistinguishable and the goal of these experiments is to qualitatively describe the behaviour of the two algorithms.
The shaded regions of objective space represent infeasible regions. It is possible to easily show the infeasible regions for CTP, because CTP’s constraints are defined on objective space (note the use of $f_1$ and $f_2$ in (9.4)).

Figure 9.2. NSGA-II vs. BSEA on CTP-6
Figure 9.3. NSGA-II vs. BSEA on CTP-7

Figure 9.4. NSGA-II vs. BSEA on CTP-8.
Figure 9.5. The first 5 generations of NSGA-II on the restricted CTP-8.

Figure 9.6. The first 5 generations of BSEA on the restricted CTP-8.
Both BSEA and NSGA-II perform excellently on CTP-6,7,8; both evenly cover the constrained Pareto front, and there is no appreciable difference between them. The results from both algorithms on CTP-6,7,8 were visually indistinguishable for 10 out of 10 runs for each algorithm, and over many algorithm parameter values in addition to the stated parameter values.

However, CTP-6,7,8 do not test the algorithms’ ability to cross infeasible regions: the feasible regions are large enough that the random initial population finds them all. The algorithms merely need to abandon infeasible solutions in preference for the non-dominated feasible solutions.
To test the algorithms’ ability to cross infeasible regions, CTP-8 is modified to restrict the random initial value of \( x_2 \) to \( 6 < x_2 \leq 10 \). (As soon as evolution begins, the usual \( 0 < x_2 \leq 10 \) range applies.) This prevents any initial solutions from being generated in the lower feasible regions. To reach the true constrained Pareto front, the population must evolve across the middle infeasible band.

Figure 9.5 and figure 9.6 show the behaviour of NSGA-II and BSEA in detail over the first 5 generations, to directly illustrate the difference in behaviour.

In both cases the random initial population is distributed over the same upper region of objective space, with most solutions in the infeasible region and some in the upper feasible regions. The entire NSGA-II population quickly pulls back to the few feasible solutions over the first three generations, abandoning some infeasible solutions that were close to the lower feasible regions. As soon as the entire population is feasible, it becomes impossible for any infeasible solution to survive in the population leaving NSGA-II stuck behind the infeasible region after 1000 generations.

In contrast, part of the BSEA population builds around the promising infeasible solutions while part withdraws to keep feasible solutions in the population. This balance is maintained by \( \alpha \) adaptively adjusting \( R_B \) in favour of objective optimisation \( R_O \) or constraint satisfaction \( R_C \). However as the run progresses \( r \) steadily increases, allowing the population to collapse into a single non-dominated rank (selected by \( R_E \)) by the end of the run. After 1000 generations, the population has correctly found the constrained Pareto front.

The described behaviours, with the same population distributions after 1000 generations (seen in Figure 9.7), were observed in 10 out of 10 runs for each algorithm.
Further experiments confirmed that BSEA is still able to cross the infeasible region when the mutation standard deviation set to as little as $1/1000^{th}$ of the decision variable range. In this case the algorithm was allowed to run for 10,000 generations to give it time to evolve gradually across the infeasible region.

### 9.6 Nonlinear crop rotation problem

The nonlinear crop rotation problem is described in [86] and derived from [88]. It has 39 decision variable (chromosome) dimensions, 16 constraint dimensions, and 2 objective dimensions. The model used here is given in section 9.11. Both objective functions are to be minimised.

The nonlinear crop rotation problem is useful as a benchmark problem because the results in objective space are easily visualised, yet the problem is difficult. [86] showed clear room for improvement for the popular and effective NSGA-II. In contrast, other easily visualised constrained multiobjective benchmark problems such as SRN, TNK, OSY, and CTP (all described in [31]) are trivial for NSGA-II to solve.

The results shown below are for illustrative purposes only. Detailed results are given in section 9.9.
Figure 9.8 shows 10 runs each of BSEA and NSGA-II on the problem. It is clear that the solution sets found by BSEA tend to dominate the solution sets found by NSGA-II. However, the variability within each set of runs is great, showing much room for improvement for both algorithms.
Figure 9.8 showed the variability, even within the same algorithm, in solution sets generated after 2,000 generations. Figure 9.9 shows similarly high variability after 200,000 generations.

It is interesting to note the very good distributions of points inside each separate solution set, particularly those of NSGA-II, that very clearly identify the shape of the front. This strongly suggests the algorithms are stuck in local optima, for two reasons:

1. In a smoothly continuous objective space, it is too much of a coincidence that every solution set should clearly identify straight-line segments, often coinciding with one another. The solution sets have identified regions of decision-variable space that are constrained by intersecting constraints (apparent by the segmented
nature of the solution sets), preventing progress further towards the true constrained Pareto front.

2. Both BSEA and NSGA-II only use diversity measures to differentiate solutions in the same rank. Such excellent internal distributions imply that the entire population has been non-dominated for many generations, giving the diversity measure opportunity and time to distribute the population.

As it is self-apparent that at least 9 out of these 10 runs have not found the true Pareto front (and, as will be shown in section 9.9, none of these runs have found the true Pareto front), the only conclusion is that there are many local constrained optima in decision-variable space, and both BSEA and NSGA-II are susceptible to becoming stuck in them.

Figure 9.10. Detail of non-dominated fronts from BSEA
Figure 9.10 shows a detail of three non-dominated fronts found by BSEA after 200,000 generations. Along the length marked 1a, all three fronts agree closely. However, along the length marked 2a and 2b, one front diverges from the other two. Along the length marked 3a and 3b, again one front diverges from the other two, but this time it is a different one. The agreements between the fronts along 1a, 2a, and 3a are too close to be coincidental (indeed, this pattern has been observed multiple times). This is further evidence of local constrained optima in the crop rotation problem.

9.7 Further insights and improvements

The preliminary trials on the nonlinear crop rotation problem showed that BSEA, with its adaptive blended-space approach to constraint satisfaction, produces generally better quality non-dominated fronts than NSGA-II, with its rigid extended-dominance approach to constraint satisfaction. However, the wide variability in both algorithms suggested that it should be possible to improve the performance of BSEA further, even if only to the extent of narrowing the variability towards the better non-dominated fronts, so that in practice one can have confidence in the quality of the result from a single run.

It is possible to improve the reliability by tuning parameters, such as population size and mutation rate, however these values cannot be known a priori and tuning is not a practical, nor even necessarily effective, approach – figure 9.9 shows that increasing the number of generations, for example, is unreliable.

This section describes insights into the desired behaviour of a constrained multiobjective algorithm, which leads to the development of Blended Rank Evolutionary Algorithm.
BREA (see section 9.8) makes use of different information calculated from different spaces, as appropriate for the desired behaviour in each space.

9.7.1 Final solution set vs. evolving solution set
It is useful to make an explicit distinction between the evolving solution set and the final solution set, particularly the different selection pressures that are desirable for each set.

The evolving solution set should retain solutions that are useful for evolution towards a desirable final solution set, which is a single non-dominated front in objective space. However, solutions that aid the evolution of the non-dominated front might not actually belong to that front.

In particular, solutions useful to evolution might not even be feasible. This is particularly true in problems where the algorithm becomes stuck in local minima, when to improve the current non-dominated front, it is first necessary to move away from it.

9.7.2 Objective space
The final solution set is best described in objective space. At the end of the run, the population should form a single non-dominated front as close to the true Pareto front as possible, and distributed as evenly and as distantly as possible.

The fitness of solutions in the final solution set should be measured first by dominance (using the extended dominance relation, to prioritise feasible solutions) and second by diversity, measured in objective space.

9.7.3 Constraint space
All feasible solutions lie at the origin of constraint space (negative scores are truncated to zero, since once a constraint is satisfied it is no longer relevant to the solution’s desirability). At the beginning of a run the search will be mostly (if not entirely) in
constraint space, directed towards the origin. Dominance is useful in constraint space, since it drives progress towards feasibility.

Unlike objective space, though, there is no Pareto-front in constraint space and therefore diversity is not useful in constraint space. However there are additional measures of quality, such as sum of constraint scores and count of constraint violations, which can be useful. (In objective space, when trying to identify a Pareto-front, sums and counts are not useful and may be counter-productive.)

9.7.4 Decision-variable space
There is no direction to the search in decision-variable space, so dominance is not useful. Diversity in decision-variable space is irrelevant in the final solution set (which should be diverse in objective space), but may be indirectly useful during evolution:

One region of objective space can map to many discrete regions of decision space. Diversity measured in objective space might then reflect diversity in just one of many regions in decision space. If any of these homomorphic regions of decision space lead to a local optimum, then maintaining diversity in objective space will not necessarily do anything to avoid local optima. Diversity measured in decision-variable space can help avoid local optima that are invisible to diversity measures in objective space.

9.7.5 Constraint satisfaction vs. objective optimisation
There will almost always be a trade-off between constraint satisfaction and objective optimisation during the search process, since ultimately constraints limit optimisation.

Algorithms using the extended dominance relation, such as NSGA-II, approach this trade-off by always prioritising constraint satisfaction, using the rationale that any feasible solution is preferable to all infeasible solutions.
Whilst this is true for the final solution set, it might not be true for the evolving solution set. Blended space algorithms, such as BSEA and BREA, take the approach of adaptively trading off constraint satisfaction for objective optimisation, and vice versa, as the situation requires (i.e., based on $\alpha$). This allows infeasible solutions to play a greater role in the search process.

9.7.5.1 Objective creep

However, preliminary experiments showed that the trade-off must be carefully considered. It was discovered that the trade-off must be based on an adaptive feedback loop (i.e., $\alpha$), or else a phenomenon best described as “invisible creep” occurred. Invisible creep requires a trade-off between two fitness measures, one relatively coarse-grained and one relatively fine-grained:

When one measure of fitness is so coarse-grained that a mutation’s effect is immeasurable, that mutation will not produce any change in selection pressure on that fitness measure. However, if the mutation has a measurable effect in another, finer-grained fitness measure, then the mutation will produce a change in selection pressure based solely on the fine-grained measure, despite any trade-off that may actually exist between the two measures. The trade-off between the two measures is invisible to the EA; it will always select for “free” measurable gains in the fine-grained fitness measure, at the invisible expense of immeasurable losses in the coarse-grained fitness measure.

This results in a slow creep towards optimising the fine-grained fitness measure. The creep will be slow enough that detriment to the coarse-grained fitness measure is invisible over any single generation; any faster and the trade-off will be measurable and the EA will select against it. Nevertheless, despite being slow it is implacable. Evolution is blind
to change that cannot be measured inside one generation, even though changes may be
significant over multiple generations.
Pareto dominance is inherently coarse-grained: non-dominated solutions are all considered equal. Furthermore, granularity becomes coarser with more dimensions: with higher dimensionality, fewer ranks are expected.
Now consider two spaces, objective space and constraint space. If objective space is low dimensional and constraint space is high dimensional, then rank in objective space will have a finer granularity, and therefore higher visibility, than rank in constraint space.
Finally, consider the trade-off between constraint satisfaction and objective optimisation: bettering one usually worsens the other. This provides the necessary elements for invisible creep: a trade-off between two measures of differing granularity.
In this case, evolution will blindly favour objective optimisation. When this occurs over many generations, the end result is that the population creeps towards the unconstrained Pareto front, and the algorithm never finds any feasible solutions.
In summary, invisible creep occurs when there is a difference in granularity between two measures in a trade-off.
Figure 9.11. Invisible creep caused by allowing constraint satisfaction to be traded for objective optimisation. “BSEA fixed weight” shows creep towards the unconstrained (infeasible) Pareto front. Feasible BSEA sample run shown for reference.

9.7.5.2 Solutions to objective creep

Invisible creep does not occur in extended dominance algorithms because there is no trade-off between objective optimisation and constraint satisfaction: constraint satisfaction is always prioritised over objective optimisation.

In blended space algorithms, the solution to invisible creep in constraint satisfaction vs. objective optimisation is to use an adaptive feedback loop to control the trade-off. The variable $\alpha$ is the fraction of feasible solutions in the population, and is the weight placed on objective optimisation. Specifically, when there are no feasible solutions there is no
weight on objective optimisation, so there is no trade-off, and so there is no invisible creep.

9.7.6 Constraint satisfaction vs. diversity
All multiobjective evolutionary algorithms prioritise rank over diversity in the final solution set. The rationale is that as long as one rank completely dominates another, the relative diversity of each rank is irrelevant; there will always be a superior choice from the former.

Once again, whilst rank should be prioritised over diversity for the final solution set, this might not be appropriate for the evolving solution set. Maintaining decision-space diversity (see section 9.7.4), at the cost of dominance ranking, may aid evolution by avoiding local optima. BREA takes the approach of adaptively trading off rank for diversity, and vice versa, as the situation requires (again, based on $\alpha$). This allows diversity to play a greater role in the search process.

9.7.6.1 Diversity creep
However, during the search for feasible solutions, and given a high dimensional constraint space, it is not uncommon for the entire population to be non-dominated, and in one rank. In such a case, the entire population is differentiated only by diversity. Diversity is usually a fine-grained continuous measure, compared to the coarse-grained discrete ranking measure. Therefore, invisible creep can occur when a measurable increase in diversity is traded for an immeasurable decrease in constraint satisfaction. The end result is that the population creeps out in all directions under the selection pressure of diversity, and the algorithm never finds any feasible solutions unless the aimless diversity pressure pushes a solution into a feasible region by chance. The population will not
disperse so widely or so quickly that any solution’s constraint rank is worsened, though, because those solutions will be immediately discarded due to prioritisation of rank over diversity.

Figure 9.12. Invisible creep caused by allowing diversity alone to differentiate the non-dominated rank. “BSEA no count” shows creep in an infeasible dispersed cloud close to the feasible region – but not so widely dispersed that there is more than one constraint rank, which would cause the population to be differentiated by constraint rank instead of diversity. Feasible BSEA sample run shown for reference.

9.7.6.2 Solutions to diversity creep
This problem does not arise in NSGA-II, because the search in constraint space for a feasible solution is measured by a sum of normalised constraint violations – i.e., a continuous measure – rather than measured by dominance rank. Therefore, since the
population is easily differentiated by this continuous measure, diversity is never required to differentiate solutions, and invisible creep cannot occur.

This problem also does not arise when using BSEA’s objective-space diversity measure. BSEA measures diversity internally to each rank in objective-space, and must arbitrarily assign the largest possible diversity to the extremes of each rank. This has the effect of preserving the best solutions in each dimension, and preventing creep.

However, this research into decision-space diversity measures (see section 9.7.4) required a global diversity measure that proved to be susceptible to creep.

The solution to invisible creep in constraint satisfaction vs. diversity, for Pareto-dominance-based algorithms operating in constraint space, is to further differentiate the population before diversity is considered. This can be achieved by an additional complementary measure of constraint satisfaction: the count of violated constraints.

In objective-space Pareto optimisation, it does not make sense to count the number of objectives in which one solution is superior to another. If the solutions are non-dominated by each other, then they are incomparable and it is irrelevant in how many dimensions one is superior to the other. After all, the ultimate aim is to discover a trade-off surface, the Pareto front. However, in constraint satisfaction, the ultimate aim is not to discover a trade-off surface, but to bring all constraint scores to 0. It is therefore reasonable to assume that, of two non-dominated solutions, the one with the greater number of constraints satisfied is more useful to evolution.

Introducing the count of constraint violations will not produce creep itself, since constraint rank and violation count are complementary measures that essentially measure
the same thing: the progress towards feasibility. There is no trade-off between these two measures.

Comparing the number of constraint violations is thus a valid measure for further differentiating non-dominated solutions. It is also elegantly effective, because this measure’s granularity becomes finer as the number of constraint dimensions increases: exactly the opposite of the ranking measure it complements.

9.8 Blended rank evolutionary algorithm

9.8.1 Overview

Each solution has the following properties:

- $D_D$, Decision-space diversity
- $D_O$, Objective-space diversity
- $R_B$, Blended rank (from $R_O$, objective rank, and $R_C$, constraint rank)
- $R_E$, Extended-dominance rank
- Constraint violation count

The EA maintains a population of fixed size $P$ from generation to generation. At each generation:

1. Selection for reproduction: $P/2$ pairs of parents are selected, with replacement, using binary tournament selection.

2. Each pair of parents generates a pair of children using crossover and mutation suitable for the decision variables (in this study, 2-point crossover and Gaussian mutation). This process generates $P$ children, which are evaluated and added to the population. Duplicate solutions are removed from the population. After this
stage, parents and children in the expanded population compete equally for survival.

3. All of the solution properties are calculated relative to the entire expanded population.

4. Selection for survival to the next generation:
   i. A fraction of $P$ solutions are selected to form the final solution set, which will be carried over to the next generation.
   ii. The remainder of the population is sorted (see Comparing solutions, below). The best are carried over to the next generation to make up the $P$ solutions.

9.8.2 Comparing solutions

During evolution, for both sexual and survival selection, solutions are compared first by $R_b$; if tied then by constraint violation count; if tied then by $D_D$.

Constraint violation count will only have any effect when the solutions are in the same rank (i.e., not differentiated by $R_b$) and are infeasible (all feasible solutions have the same constraint violation count). Its effect is twofold: to increase selection pressure during the search for feasibility; and to differentiate infeasible solutions in the same rank in a complementary way, so that the opposing selection pressure for diversity cannot cause creep towards diversity (see section 9.7.6.2).

9.8.3 Decision-space diversity

A solution’s $D_D$ is a logarithmically-weighted sum of the distances to all other solutions in the population (of size $P$).
The norm-1 (Manhattan) distance to every other solution is calculated (normalised over each dimension), and sorted in ascending order, to give a set \( d = \{d_1, \ldots, d_P\} \), so that \( d_i \) is the distance to the solution’s \( i^{th} \) nearest neighbour.

A fixed set of corresponding weights is calculated as follows (this can be calculated and stored just once for the entire run, so long as \( P \) does not change):

\[
w_i = 1 - \frac{\log_e(i)}{\log_e(P)} \quad \text{for} \quad i = 1, \ldots, P\,.
\]

Finally, the weighted distances are summed:

\[
D_D = \sum_{i=1}^{P} w_i \cdot d_i\,.
\]

In other words, much more weight is given to the distances to a solution’s closest neighbours than is given to the distances to its more distant neighbours. \( D_D \) is normalised to \([0,1]\) across the entire population.

This measure was designed with the following considerations:

NSGA-II’s \textbf{distance-to-neighbours} measure and [56]’s uniquely dominated hypervolume measure can only be applied to non-dominated ranks, which do not exist in decision-variable space.

SPEA2’s [115] \textbf{\( k^{th}\)-nearest-neighbour} measure is global, but \( k^{th}\)-nearest-neighbour uses the distance to just one (arbitrary) solution and the selection of \( k \) is problematic.

When \( k \) is low, niches are vulnerable to extinction. For example in \textbf{nearest-neighbour}, when \( k=1 \), if a single previously highly diverse solution is mutated slightly to produce a very similar solution, both solutions will suddenly have a very low diversity, measured to one another. Whilst nearest-neighbour (\( k=1 \)) distributes points excellently across
continuous spaces, it leaves small discrete feasible regions, with no room to spread out, vulnerable to extinction. As $k$ increases, the vulnerability to extinction decreases, but it becomes less useful in measuring individual diversity.

$D_D$ uses a weighted sum of distances to all other solutions to avoid the difficulty of selecting an appropriate $k$.

When $w_k=1$ and all other weights are 0, the weighted sum of distances is equivalent to $k^{\text{th}}$-nearest-neighbour.

When all the weights are equal, the population is pushed to the edges of the domain, and any point around the edge of the domain measured as equally diverse since an equal distance lost from one neighbour is gained to another.

When the weights decrease linearly from 1 to 0, the distribution is somewhere in between the desirable grid-like distribution of nearest-neighbour and the extreme edge distribution of equally-weighted-sum.

When the logarithmic scale in (9.5) is used, the distribution is even closer to the desirable nearest-neighbour distribution but, by placing a progressively smaller weight on more distant neighbours, avoids nearest-neighbour’s vulnerability.

During evolution, $D_D$ (decision-space diversity) is used in preference to the usual $D_O$ (objective-space diversity) to distribute the population through decision-variable space to avoid becoming stuck in local optima.
9.8.4 Blended rank

Blended rank, $R_B$, is calculated as follows:

$$R_B = \alpha D_D + (1- \alpha)(\alpha R_O + (1- \alpha)R_C) \quad (9.7)$$

or, alternatively:

$$R_B = \alpha D_D + (1- \alpha)(R_{B(old)}) ,$$

$$R_{B(old)} = \alpha R_O + (1- \alpha)R_C \quad (9.8)$$

where $\alpha = \frac{\text{num feasible solutions}}{\text{population size}}$ in both cases.

This differs from the old blended rank used in BSEA and [4], which is now part of the right-hand term in the new blended rank. The new blended rank is basically the old blended rank, blended again with diversity.

The new blended rank can be interpreted as:

The greater the feasible fraction, $\alpha$, the greater weight placed on diversification, and the less weight placed on the traditional constraint-satisfaction vs. objective optimisation trade-off.

Within that trade-off, the greater the feasible fraction, the greater the weight placed on objective optimisation, and the less weight placed on constraint satisfaction – this behaviour was previously established in BSEA.

The relationship between the feasible fraction, $\alpha$, and the weights on diversity, constraint satisfaction, and objective optimisation, is clearly illustrated graphically in figure 9.13.
Diversity weight: $\omega = \alpha$

Objective weight: $\omega = (1-\alpha) \cdot \alpha$

Constraint weight: $\omega = (1-\alpha) \cdot (1-\alpha)$

Sum of weights: $\omega = \alpha + (1-\alpha) \cdot \alpha + (1-\alpha) \cdot (1-\alpha)$

Figure 9.13. The diversity, constraint, and objective weights as functions of $\alpha$. All weights sum to 1.

In effect, by introducing diversity as a top-level term in the blended rank (as opposed to diversity usually being secondary to dominance), BREA places much greater emphasis on maintaining diversity in the population. Diversity is increased only as far as conditions allow (i.e., depending on the number of feasible solutions in the population), and is controlled adaptively in a feedback loop so that it does not spiral out of control or cause creep (see section 9.7.6.1).

This greater emphasis on decision-space diversity is intended to help avoid becoming stuck in local optima. Blended rank is used as the primary measure for comparing solutions during evolution.
$R_O$ and $R_C$ are both calculated using Pareto dominance in their respective spaces and the ranking procedure in [38]. To avoid bias, the rank values $R_O$ and $R_C$, and $D_D$, are all normalised to [0,1), causing the blended rank $R_B$ to also fall in [0, 1).

### 9.8.4.1 Two roles for $D_D$

The astute reader will notice that $D_D$ has been used twice: once just above as a weighted component in the blended rank $R_B$, and once in section 9.8.2 as a tertiary fitness measure (used only if the primary fitness measure, $R_B$, and the secondary fitness measure, constraint violation count, result in a tie when comparing solutions).

This is not redundant: note that, according to figure 9.13, when the population is entirely infeasible and $\alpha = 0$, there is no weight on diversity or objective optimisation. In this case $R_B$ will only reflect constraint rank, and so diversity still has a role to play as a tertiary fitness measure used to compare solutions.

Conversely, when $\alpha > 0$, $R_B$ will also reflect the continuous, fine-grained diversity measure, making it extremely unlikely that any two solutions will have the same $R_B$ and therefore extremely unlikely that the secondary and tertiary fitness measures will be necessary to differentiate solutions.

### 9.8.5 Extended-dominance rank and objective-space diversity

Extended-dominance rank, $R_E$, and objective-space diversity, $D_O$, are useful for selecting a diverse final solution set to present to the decision-maker.

$R_E$ is calculated using extended dominance ranking (see (9.1) in section 9.2.5.1).

Unlike NSGA-II, Pareto dominance is used in constraint space (rule 3) as well as in objective space (rule 1).
$D_o$ is calculated internally to each non-dominated rank, identically to NSGA-II [28]:

A solution’s diversity is the sum, over all objective dimensions, of the distance between the solution’s closest neighbours on either side of it in each objective dimension. All dimensions are normalized to $[0, 1)$. Solutions at the two extremes of each objective dimension are assigned an arbitrary diversity – in this case the largest representable real number, which has the effect of preserving the best and worst solutions in each objective dimension.

NSGA-II’s diversity measure is used in preference to the uniquely dominated hypervolume used previously in BSEA because uniquely dominated hypervolume did not perform significantly better than the normalised summed distance to neighbours, yet was much more computationally expensive.

### 9.8.6 Final solution set

A limitation of the blended space algorithm is that the blended rank actively avoids collapsing the population into a single non-dominated rank. This behaviour is useful during evolution, but is unfortunate for the final decision-maker, since most of the population is useless as a final solution set: most solutions are either infeasible or dominated.

To address this BREA recognises that, ultimately, extended-dominance-based algorithms produce a solution set with desirable properties: an entire population of feasible, non-dominated solutions, distributed evenly by objective-space diversity. Therefore, BREA reserves up to a certain fraction, $r$, of the population to be the final solution set:

$$r = \frac{\text{current generation}}{\text{total generations}} \quad (9.9)$$
This solution set is selected from the population using $R_E$ (instead of $R_B$) and $D_O$ (instead of $D_D$), which are the measures that reflect what is desirable in a final solution set. Only non-dominated ($R_E=1$) solutions are selected:

- If there are too few $R_E=1$ solutions to make up $r$, then no more are selected.
- If there are too many $R_E=1$ solutions to fit into $r$, then those with the highest $D_O$ are selected, up to $r$. The remainder of the population is selected as normal.

The final solution set grows linearly over the run of the algorithm, until when the run completes the entire population becomes the final solution set.

This reservation is unaffected by $\alpha$, which merely influences the blended rank; however the reserved feasible solutions are still counted in the population and still affect the dynamic feedback loop by forcing $\alpha$ to rise steadily with $r$. This relationship was found to be complementary: as the number of feasible solutions forcibly increases, $\alpha$ will shift $R_B$ more towards $D_D$ and $R_O$, allowing progressively more unconstrained exploration with the fewer remaining infeasible solutions.

### 9.8.7 Computational overhead

Blended ranking using objective space rank, constraint space rank, and decision space diversity has a computational complexity of $O((M+N+L)P^2)$ where $M$ is the number of objective dimensions, $N$ is the number of constraint dimensions, $L$ is the number of decision-variable dimensions, and $P$ is the size of the population.

BREA has a greater computational overhead than NSGA-II. It is effectively doing all the computation of NSGA-II during selection for the final solution set (ranking using extended-dominance plus objective-space diversity), plus:
• Ranking in objective space: every solution must be compared to every other solution \( (P^2) \), and each dominance comparison requires \( M \) objective score comparisons, for \( MP^2 \) complexity.

• Ranking in constraint space: every solution must be compared to every other solution \( (P^2) \), and each dominance comparison requires \( N \) constraint score comparisons, for \( NP^2 \) complexity.

• Diversity in decision-variable space: the distance from every solution to every other solution must be calculated \( (P^2) \), and each calculation requires \( L \) measurements, for \( LP^2 \) complexity.

This \( O((M+N+L)P^2) \) is compared to the computational complexity of NSGA-II which is \( O(MP^2) \). The linear increase in computational overhead is expected to be masked by evaluation functions in many real-world problems – particularly problems that require expensive simulations – and/or to be an acceptable price for solution quality.

9.9 Experiments

BREA was tested on the nonlinear crop rotation problem with the following parameters, which were identical for BSEA and NSGA-II:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>generations</td>
<td>2,000 and 200,000</td>
</tr>
<tr>
<td>population size</td>
<td>100</td>
</tr>
<tr>
<td>crossover points</td>
<td>2</td>
</tr>
<tr>
<td>crossover rate</td>
<td>0.95</td>
</tr>
<tr>
<td>mutation rate</td>
<td>0.025</td>
</tr>
<tr>
<td>mutation index</td>
<td>20</td>
</tr>
</tbody>
</table>

The fundamental evolutionary operators and parameter values were taken from [86].
The hypervolume of each solution set was measured using an arbitrary bounding point of (-5000000, 16000) which was selected to be greater in each dimension than any solution found in any of the runs displayed below.

### 9.9.1 Comparing algorithms

Figure 9.14 visually compares 10 runs of each algorithm over 2,000 generations. BREA clearly dominates NSGA-II, and performs very strongly compared to BSEA, and solutions sets from independent runs show greatly reduced variability. As previously seen, BSEA also tends to produce better solution sets than NSGA-II, but the variability between runs is very high and overlaps NSGA-II to a large extent.

BREA’s final solution sets have worse internal distributions than both BSEA and NSGA-II.

Table 9.4 gives the total hypervolume dominated by each solution set (averaged across each algorithm), and the standard deviation of these scores for each algorithm. These scores validate the above visual observations: BREA’s solution sets dominate much greater volumes of objective space than both BSEA and NSGA-II, and the standard deviation is greatly reduced.
Figure 9.14. 10 runs each of BREA vs. BSEA vs. NSGA-II over 2,000 generations.

Table 9.4. Mean and standard deviation of hypervolume metric from 10 runs of each algorithm. Best in bold.

<table>
<thead>
<tr>
<th></th>
<th>BREA 2,000</th>
<th>BSEA 2,000</th>
<th>NSGA-II 2,000</th>
</tr>
</thead>
<tbody>
<tr>
<td>mean</td>
<td>4.277377×10^{10}</td>
<td>2.839654×10^{10}</td>
<td>1.820166×10^{10}</td>
</tr>
<tr>
<td>stdv</td>
<td>3.804603×10^{9}</td>
<td>1.334271×10^{10}</td>
<td>7.126510×10^{9}</td>
</tr>
</tbody>
</table>

Table 9.5. Analysis of variance P values. Significant in bold.

<table>
<thead>
<tr>
<th></th>
<th>BSEA 2,000</th>
<th>NSGA-II 2,000</th>
</tr>
</thead>
<tbody>
<tr>
<td>BREA 2,000</td>
<td>0.0042</td>
<td>&lt; 0.0001</td>
</tr>
<tr>
<td>BSEA 2,000</td>
<td></td>
<td>0.0471</td>
</tr>
</tbody>
</table>

Table 9.5 shows analysis of variance P values. All algorithms are statistically significantly different from one another – BREA much more so to BSEA and NSGA-II than the latter are to each other.
Table 9.6. Mean and standard deviation of hypervolume metric from 5 runs of each algorithm. Best in bold.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Mean</th>
<th>Stdv</th>
</tr>
</thead>
<tbody>
<tr>
<td>BREA 200,000</td>
<td>$5.802992 \times 10^7$</td>
<td>$1.618070 \times 10^9$</td>
</tr>
<tr>
<td>BSEA 200,000</td>
<td>$3.845517 \times 10^7$</td>
<td>$6.086662 \times 10^9$</td>
</tr>
<tr>
<td>NSGA-II 200,000</td>
<td>$2.663374 \times 10^7$</td>
<td>$1.350547 \times 10^9$</td>
</tr>
</tbody>
</table>

Table 9.7. Analysis of variance P values. Significant in bold.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>BSEA 200,000</th>
<th>NSGA-II 200,000</th>
</tr>
</thead>
<tbody>
<tr>
<td>BREA 200,000</td>
<td>&lt; 0.0001</td>
<td>0.0009</td>
</tr>
<tr>
<td>BSEA 200,000</td>
<td></td>
<td>0.0329</td>
</tr>
</tbody>
</table>

Figure 9.15 visually compares 5 runs of each algorithm over 200,000 generations. Over this increased number of generations, BREA even more clearly dominates both BSEA and BREA, and the variability between runs is further reduced. BSEA and NSGA-II entirely overlap, and it is visually apparent that they are stuck in local optima (see section 9.6, particularly figure 9.10).
Table 9.6 validates the above visual observations, and table 9.7 confirms the statistical significance.

In both figure 9.14 and figure 9.15 it can be seen that NSGA-II’s solution sets have the best internal distribution of points, followed by BSEA, followed by BREA. This is explained by the following characteristics of the algorithms:

- In NSGA-II the selection pressure is the same over the duration of the run: specifically, the population is under strong pressure to collapse into a single nondominated rank, differentiated by diversity. This gives diversity a large scope for distributing the population, resulting in the excellent internal distribution of points. In contrast, both BSEA and BREA avoid collapsing into a single nondominated rank until the very end of the run, giving diversity less scope to distribute the population.

- All algorithms use objective-space diversity to differentiate solutions inside the same rank (which, as mentioned above, is a less frequent occurrence in BSEA and BREA). However, BREA additionally blends decision-variable-space diversity into the rank, 1) making it even less likely that solutions belong to the same rank, and 2) encouraging genetic diversity which does not necessarily correspond to objective-space diversity.

Both BSEA and BREA attempt to address this weakness by applying different selection pressure (actually the same as NSGA-II) to the final solution set, but these results show that this is not enough to make up the head-start NSGA-II has in using diversity to distribute the population.
9.9.2 Comparing progress
It is useful to know how the algorithms progress over time in order to draw conclusions about the algorithm’s ability to avoid local optima.

The following comparisons use the same solution sets as the previous section. Purely for visual clarity, only 5 (of 10) solution sets (selected randomly) are shown for each algorithm at 2,000 generations. However, all available solution sets are used for calculating the mean hypervolume, standard deviation, and statistical significance, to increase the statistical confidence.

A note on interpretation of scores: It is not obvious how progress should be quantitatively compared between algorithms. It is (relatively) straightforward to compare final solution sets between algorithms, but how should progress be compared? Should progress be measured in absolute units, or as a percentage of the initial measurement?

Consider that:

- The algorithms start and finish in different regions of objective-space.
- Progress is not expected to be uniformly easy to achieve across objective-space (absolute progress towards the Pareto front is expected to be easier to achieve when starting farther away from the Pareto front, and absolute progress to lower variability is expected to be easier when starting from a high variability).

This work suggests that progress towards the Pareto front should be compared by increased absolute volume, since comparing by percentage would greatly favour the progress made by poor-quality solution-sets where hypervolume scores are low and progress is easy even in absolute terms.
Conversely, this work suggests that progress in reducing variability between runs should be measured by percentage decrease in standard deviation, since comparing absolute standard deviations would greatly favour the large decreases possible from a high initial standard deviation (which may be impossible to match from a low initial standard deviation, considering that it cannot be reduced below 0!).

Nevertheless, as the above interpretation happens to rate BREA better than BSEA and NSGA-II, both absolute and percentage changes are given below for completeness but should be interpreted with caution.

Figure 9.16 - Figure 9.18 and Table 9.8 - Table 9.13 compare each algorithm against itself when allocated different numbers of generations.

**NSGA-II** shows a small absolute improvement in the mean hypervolume measure, but the variance between runs (measured by standard deviation) is actually much worse after 200,000 generations. Furthermore, the probability of the improvement in mean hypervolume being chance is 0.1311, well above the threshold of 0.05 for statistical significance. This is mostly due to the larger standard deviation.

**BSEA**’s absolute improvement in mean hypervolume over 200,000 generations is slightly better than NSGA-II’s, and BSEA shows an improvement in the variance between runs (in contrast to NSGA-II). However, after 200,000 generations the runs still exhibit a large variance, largely overlap with the runs from 2,000 generations, show signs of still being stuck in local optima (see section 9.6), and the probability of the improvement in mean hypervolume being chance is 0.0437, only just under the threshold of 0.05.
Table 9.8. Mean and standard deviation of hypervolume metric from all runs of each algorithm. Best in bold.

<table>
<thead>
<tr>
<th></th>
<th>NSGA-II 200,000</th>
<th>NSGA-II 2,000</th>
<th>difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>mean</td>
<td>$2.663374 \times 10^{10}$</td>
<td>$1.820166 \times 10^{10}$</td>
<td>0.84$\times 10^{10}$ (146%)</td>
</tr>
<tr>
<td>stdv</td>
<td>$1.350547 \times 10^{10}$</td>
<td>$7.126510 \times 10^{9}$</td>
<td>+6.38$\times 10^{9}$ (190%)</td>
</tr>
</tbody>
</table>

Table 9.9. Analysis of variance P values. Significant in bold.

<table>
<thead>
<tr>
<th></th>
<th>NSGA-II 2,000</th>
<th>NSGA-II 200,000</th>
</tr>
</thead>
<tbody>
<tr>
<td>NSGA-II 200,000</td>
<td></td>
<td>0.1311</td>
</tr>
</tbody>
</table>
Figure 9.17. 5 runs each of BSEA over 200,000 generations vs. BSEA over 2,000 generations.

Table 9.10. Mean and standard deviation of hypervolume metric from all runs of each algorithm. Best in bold.

<table>
<thead>
<tr>
<th></th>
<th>BSEA 200,000</th>
<th>BSEA 2,000</th>
<th>difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>mean</td>
<td>3.845517×10^{10}</td>
<td>2.839654×10^{10}</td>
<td>1.01×10^{10} (135%)</td>
</tr>
<tr>
<td>stdv</td>
<td>6.086662×10^{9}</td>
<td>1.334271×10^{10}</td>
<td>-7.26×10^{9} (46%)</td>
</tr>
</tbody>
</table>

Table 9.11. Analysis of variance P values. Significant in bold.

<table>
<thead>
<tr>
<th></th>
<th>BSEA 2,000</th>
<th>BSEA 200,000</th>
</tr>
</thead>
<tbody>
<tr>
<td>BSEA 2,000</td>
<td>0.0437</td>
<td></td>
</tr>
</tbody>
</table>
Figure 9.18. 5 runs each of BREA over 200,000 generations vs. BREA over 2,000 generations.

Table 9.12. Mean and standard deviation of hypervolume metric from all runs of each algorithm. Best in bold.

<table>
<thead>
<tr>
<th></th>
<th>BREA 200,000</th>
<th>BREA 2,000</th>
<th>change</th>
</tr>
</thead>
<tbody>
<tr>
<td>mean</td>
<td>5.802992x10^10</td>
<td>4.277377x10^10</td>
<td>1.52x10^10 (136%)</td>
</tr>
<tr>
<td>stdv</td>
<td>1.618070x10^9</td>
<td>3.804603x10^9</td>
<td>-2.19x10^9 (43%)</td>
</tr>
</tbody>
</table>


<table>
<thead>
<tr>
<th></th>
<th>BREA 2,000</th>
</tr>
</thead>
<tbody>
<tr>
<td>BREA 200,000</td>
<td>&lt; 0.0001</td>
</tr>
</tbody>
</table>

BREA shows the greatest absolute improvement in hypervolume, and a similar percentage reduction in variance to BSEA (both algorithms more than halve their standard deviation after 200,000 generations. However, although the reduction in standard deviation is similar to BSEA, when it is combined with the greater improvement in absolute hypervolume this means that all 200,000-generation runs dominate all 2,000-
generation runs; there is no overlap, unlike BSEA and NSGA-II. Finally, BREA has by far the lowest probability that the improvement in hypervolume is chance, at a virtual certainty of < 0.0001.

9.9.3 Comparing computational overhead
It was previously noted that the computational complexity of BREA is \( O((M+N+L)P^2) \), whereas the computational complexity of NSGA-II is just \( O(MP^2) \). Since the difference in complexity is only linear, this notation does not give much insight into the real-world computational cost where implementation details may mask any linear increase in cost. Therefore experiments were timed on a Pentium 4 3GHz (to the closest second) in order to determine whether the linear increase in cost is significant.

BREA takes 60s to evolve a population of 100 for 2,000 generations. (It is interesting to note that 40s, or \( 2/3 \)rd of that time is spent calculating decision-space diversity, suggesting large gains could be made by optimising this subroutine.)

NSGA-II takes just 5s to evolve a population of 100 for 2,000 generations. It takes 50s to evolve a population of 100 for 20,000 generations, and 58s to evolve a population of 800 for 2,000 generations.

Clearly the additional measurements used by BREA come at a high linear cost, even if the algorithm can be optimised. Therefore, it is useful to look at the behaviours of the algorithms when given a similar computational budget – 60 seconds.
Figure 9.19 shows that even over a similar computational budget, BREA tends to find superior non-dominated fronts compared to NSGA-II.

However NSGA-II’s final solution sets have much better internal distributions of points. This may be attributed to two causes:
• The NSGA-II runs have either many more generations or a much greater population size.

• NSGA-II is inherently better at distributing the population across a single rank in objective-space (see section 9.9.1).

It should be noted that the number of generations and population sizes above were chosen to fit inside a computational budget for the crop rotation problem, which is a computationally cheap function optimisation problem. Therefore the times closely reflect the amount of computation performed by the search algorithms, not the function evaluation. In particular, NSGA-II’s 10-fold increase in generations or 8-fold increase in population size inside the same computational budget, seen above, cannot be generalised to other problems with more expensive fitness evaluations.

9.10 Conclusions

Section 9.5 illustrated the difference in constraint handling behaviour between the extended-dominance-based NSGA-II and the blended-space-based BSEA, using a modified form of the CTP-8 benchmark problem. BSEA demonstrated the ability to more easily cross infeasible regions. This is due to blending objective-space rank and constraint-space rank, allowing infeasible solutions to play a greater role in evolution.

In section 9.6, on the crop rotation problem, BSEA tended to find better quality non-dominated fronts. However, the variability between runs for both BSEA and NSGA-II was disconcertingly high. The results suggested that both NSGA-II and BSEA were trapped in local optima.
In general, this chapter characterises CMEAs in the context of the strategy they use to reconcile the conflicting goals of constraint satisfaction and objective optimisation.

BREA was developed to make use of different measures in different spaces, as appropriate for the desired behaviour in each space. In particular, BREA’s strategy is to maintain an adaptive trade-off between genetic diversity, constraint satisfaction, and objective optimisation. (This is in contrast to the common extended dominance strategy, which rigidly prioritises constraint satisfaction, then objective optimisation, then diversity.)

BREA also makes an explicit distinction between the evolving solution set and the final solution set, and applies different selection pressures to each. The fraction of the population subject to the final solution set’s selection pressure increases progressively over the duration of the run, until at the end of the run the entire population forms the final solution set. All of this is accomplished without additional parameters.

BREA offers three key advantages in problems that are highly constrained with many local optima, all realised through its ability to cross infeasible regions and avoid local optima:

- Solution sets with better proximity to the Pareto front.
- Statistical confidence in the high quality of any given solution set when the true Pareto front is unknown.
- Solution sets will continue to evolve toward the true Pareto front, given additional generations to work with, rather than wasting time stuck in local optima.
9.10.1 Proximity
BREA consistently produced the solution sets with the closest proximity to the Pareto front. BREA’s solution sets dominated almost every other solution found in these experiments (the exception being in figure 9.14, in which just 8 solutions from BSEA were non-dominated by any solutions from BREA). In the 200,000 generation comparison, the worst solution set from BREA completely dominated every solution set from BSEA or NSGA-II.
BREA’s mean hypervolume scores were consistently the best in all experiments and comparisons.

9.10.2 Confidence
One of the goals in improving BSEA was to reduce the variability between independent runs (ideally towards the better side of the variance!) in order to allow a greater confidence in the quality of any single run. BREA’s standard deviation in hypervolume scores is nearly half that of NSGA-II (53%) after 2,000 generations; just 12% after 200,000 generations; and only slightly higher (107%) than NSGA-II when NSGA-II was given a population size of 800. Overall, BREA exhibits excellent (low) variability between runs.

9.10.3 Evolutionary progress
Given a 100-fold increase in generations, both NSGA-II and BSEA show small improvements in proximity to the true Pareto front. However, combined with the variability between runs there is no certainty that a 200,000-generation run will produce a better solution set than a 2,000-generation run. In contrast, BREA both improves the
proximity of the fronts and reduces the variability, resulting in every 200,000 generation run dominating every 2,000-generation run.

This is reflected in the analysis of variance of hypervolume scores, which reveals that the probability of any observed progress being due to chance is $P = 0.0437$ for BSEA and $P = 0.1311$ for NSGA-II, but a virtual certainty of $P < 0.0001$ for BREA.

### 9.10.4 Distribution

The only disadvantage to BREA observed in these experiments was a poorer distribution of solutions in the final solution set (see section 9.9.1). This may be addressed if necessary, but in these experiments the distribution was mostly irrelevant given that there was such a difference in proximity to the Pareto front between all algorithms, and even between runs of the same algorithm. Distribution is irrelevant when one solution set completely dominates another solution set, and this was often the case observed in these experiments.

More effort could go into improving the distribution of BREA’s final solution sets. One of the reasons identified for BREA’s poor distribution is that the population actively avoids collapsing into a single non-dominated rank until the very end of the run (controlled by the reserved fraction, $r$), limiting the ability of diversity to effectively distribute the final solution set. This could be addressed simply by running the algorithm for longer after $r$ reaches 1.0, creating a final stage of evolution devoted to improving the final solution set. That was not investigated here.

### 9.10.5 Computational overhead

BREA has a large linear increase in computational overhead over NSGA-II, so experiments where both algorithms had the same computational budget were performed.
Even in this comparison, BREA had a higher mean hypervolume than the best version of NSGA-II, and BREA produced solution sets that completely dominated every solution set from NSGA-II, although NSGA-II had a slightly lower standard deviation.

NSGA-II was allowed to use up its computational budget either by increasing the number of generations (from 2,000 to 20,000) or increasing the population size (from 100 to 800). Note that the 8-fold increase in population size is less than the 10-fold increase in generations because the computational complexity of $O(MP^2)$ (per generation) implies a linear increase in cost as generations increase, but specifies an exponential increase in cost as population size increases.

It is interesting that NSGA-II performed much better with the larger population than with the greater number of generations (although this is unsurprising as previous experiments had shown that even a 100-fold increase in generations did not greatly improve NSGA-II’s solution sets). This is particularly relevant to computational complexity because to reliably improve its solution sets NSGA-II must increase its population size (at exponential computational cost) whereas BREA can increase its generations (at linear computational cost).

It should be noted that these times were recorded for the crop rotation problem, which is a computationally cheap function optimisation problem, with trivial objective functions and constraint functions. The cost of BREA’s evolutionary operations – most notably the cost to calculate diversity in the 39-dimensional decision-variable space – dwarfed the cost of fitness evaluation.
The crop rotation problem, with its high dimensionality and cheap evaluation, is something of a worst-case scenario for BREA. The 10- and 8-fold increases in generations and population size possible for NSGA-II on this problem cannot be generalised to problems with more expensive fitness evaluations; the more expensive the fitness evaluation, the smaller the increases possible for NSGA-II under the same computational budget.

In many real-world problems, particularly those requiring simulation, the cost of fitness evaluation can make the cost of evolutionary operations pale into insignificance. For example, section 7.5.1 showed that the cost of a single simulation of the inverted pendulum (itself a relatively simple simulation) took ~0.1653s. Assuming a total of 200,000 evaluations (2,000 generations and 100 population-size), the cost of fitness evaluation is 33,060s, or roughly 3 orders of magnitude greater than the cost of BREA’s evolutionary operations for the same number of fitness evaluations.

One should question when computational budget is important in the first place; arguably it is most often important when expensive fitness evaluations are necessary. When the cost of evolutionary operators is masked by the cost of fitness evaluation, it is reasonable to select an algorithm that produces good results with fewer total fitness evaluations, and that makes every fitness evaluation count (e.g., makes use of information from all spaces, and does not waste evaluations in local optima). BREA arguably fits these criteria well and so, counter-intuitively, the most computationally expensive algorithm may be the most appropriate choice for computationally expensive problems!
9.10.6 Publications

From this research, the BSEA algorithm was published in [107], and the BREA algorithm was published in [108].
9.11 Nonlinear crop rotation problem

The nonlinear crop rotation model is given in [88], and the description below is quoted from [87]:

“Assume that a country grows a wide variety of crops in different seasons, and it has different types of lands. The yield rate, the cost of production, and the return from crop are functions of soil characteristics (fertility and other soil factors), region, the crop being produced, cropping pattern and method (crop being produced and their sequence, irrigation, non-irrigation, etc.). For a single-cropped land, there are a number of alternative crops from which the crop to be cultivated in a year may be chosen. Similarly there are many different combinations of crops for double-cropped (two crops in a year) and triple-cropped (three crops in a year) lands. Different combinations give different outputs. The utilisation of land for appropriate crops is the key issue for the crop-planning problem. The problem is to provide an annual crop production plan that determines the area to be used for different crops while fulfilling the demand, land, capital, import and region limitations.

“There are more than 100 crops in the country under consideration. The decision makers are interested in an aggregate planning model to make high level planning decisions. Hence all the crops are divided into 10 major crop groups. The number of crop combinations identified for single-, double- and triple-cropped lands is 10, 17 and 6 respectively according to the present cropping pattern. Any of the 10 major groups/crops can be produced in a year in the single-cropped land. There are 17 pairs of crops that can be produced (one after another of the pair) in a year in double-cropped lands while 6
combinations (3 crops in each group, one after another in a year) in triple-cropped lands.

The model consists of 68 variables and 45 constraints.

“In this study, we consider two objectives (contribution maximization and cultivation cost minimization) and similar constraints. However, we applied variable /constraint reduction technique which reduces the model to 39 variables and 15 constraints.

...  

“As many people would oppose the idea of considering linear models as test cases, we have solved another instance of the problem which allows nonlinear yield for triple-cropped land. In this instance, the variables from 27 to 32 were changed from linear to non-linear (with power of 1.1) in objective 1 and constraints 1, and 3 to 7.”

9.11.1 Decision-variable domains

\[ 0 \leq x_1, \ldots, x_8 \leq 500 \]
\[ 62.8604 \leq x_9 \leq 169.5437 \]
\[ 27.3913 \leq x_{10} \leq 134.0746 \]
\[ 0 \leq x_{11}, \ldots, x_{33} \leq 500 \]
\[ 0 \leq x_{34}, \ldots, x_{38} \leq 5 \]
\[ 0 \leq x_{39} \leq 500 \]

9.11.2 Objective functions

\[ f_1(x) = -(1134x_1 + 116x_2 + 2773x_3 + 2195x_4 + 291x_5 - 702x_6 + 8125x_7 + 7090x_8 + 47722x_9 + 16761x_{10} + 3907x_{11} + 3329x_{12} + 1425x_{13} + 9359x_{14} + 2311x_{15} + 407x_{16} + 8241x_{17} + 4968x_{18} + 4064x_{19} + 2071x_{20} + 10898x_{21} + 1493x_{22} + 9285x_{23} - 411x_{24} + 7381x_{25} + 7423x_{26} + 15215x_{27}^{1.1} + 6102x_{28}^{1.1} + 4198x_{29}^{1.1} + 12023x_{30}^{1.1} + 4366x_{31}^{1.1} + 2362x_{32}^{1.1} + 10196x_{33} - 5940x_{34} - 5940x_{35} - 5940x_{36} - 5940x_{37} - 4609x_{38} - 12341x_{39}) \]

\[ f_1(x) = 4.313x_1 + 3.450x_2 + 4.702x_3 + 7.557x_4 + 4.518x_5 + 5.233x_6 + 2.090x_7 + 1.355x_8 + 1.680x_9 + 12.486x_{10} + 9.015x_{11} + 11.870x_{12} + 8.831x_{13} + 6.403x_{14} + 11.007x_{15} + 7.968x_{16} + 5.540x_{17} + 12.259x_{18} + 9.220x_{19} + 9.935x_{20} + 6.792x_{21} + 12.790x_{22} + 8.912x_{23} + 9.715x_{24} + 5.873x_{25} + 7.323x_{26} + 3.445x_{27} + 16.572x_{28} + 13.533x_{29} + 11.105x_{30} + 17.492x_{31} + 14.453x_{32} + 12.025x_{33} \]
9.11.3 Constraint functions

\[ g_1(x) = -(0.46x_1 + 0.46x_{11} + 0.46x_{12} + 0.46x_{13} + 0.46x_{14} + 0.46x_{28}^{1.1} + 0.46x_{29}^{1.1} + \ 0.46x_{30}^{1.1} + x_{34} - 227.4566) \]

\[ g_2(x) = -(0.41x_2 + 0.41x_{15} + 0.41x_{16} + 0.41x_{17} + x_{18} - 89.2498) \]

\[ g_3(x) = -(0.7x_3 + 0.7x_{11} + 0.7x_{18} + 0.7x_{19} + 0.7x_{20} + 0.7x_{21} + 0.7x_{28}^{1.1} + 0.7x_{29}^{1.1} + 0.7x_{30}^{1.1} + 0.7x_{31}^{1.1} + 0.7x_{32}^{1.1} + 0.7x_{33} + x_{36} - 878.5105) \]

\[ g_4(x) = -(1.056x_4 + 1.056x_{12} + 1.056x_{15} + 1.056x_{18} + 1.056x_{22} + 1.056x_{23} + 1.056x_{28}^{1.1} + 1.056x_{31}^{1.1} + x_{37} - 710.2647) \]

\[ g_5(x) = -(0.75x_5 + 0.75x_{13} + 0.75x_{16} + 0.75x_{19} + 0.75x_{24} + 0.75x_{25} + 0.75x_{29}^{1.1} + 0.75x_{32}^{1.1} + x_{37} - 118.0728) \]

\[ g_6(x) = -(0.658x_6 + 0.658x_{20} + 0.658x_{22} + 0.658x_{24} + 0.658x_{26} + 0.658x_{31}^{1.1} + 0.658x_{32}^{1.1} + 0.658x_{33} + x_{39} - 28.7016) \]

\[ g_7(x) = -(0.921x_7 + 0.921x_{14} + 0.921x_{17} + 0.921x_{21} + 0.921x_{26} + 0.921x_{27}^{1.1} + 0.921x_{30}^{1.1} + 0.921x_{33} + x_{39} - 407.4360) \]

\[ g_8(x) = -(0.935x_8 + 0.935x_{23} + 0.935x_{25} + 0.935x_{27} - 41.7513) \]

\[ g_9(x) = -(9.453x_9 - 594.2200) \]

\[ g_{10}(x) = -(0.506x_{10} - 13.8600) \]

\[ g_{11}(x) = -(772.7000 - (x_1 + x_2 + x_3 + x_4 + x_6 + x_7 + x_8 + x_9 + x_{10})) \]

\[ g_{12}(x) = -(961.5000 - (x_{11} + x_{12} + x_{13} + x_{14} + x_{15} + x_{16} + x_{17} + x_{18} + x_{19} + x_{20} + x_{21} + x_{22} + x_{23} + x_{24} + x_{25} + x_{26} + x_{27})) \]

\[ g_{13}(x) = -(238.7000 - (x_{28} + x_{29} + x_{30} + x_{31} + x_{32} + x_{33})) \]

\[ g_{14}(x) = -(17000 - (4.313x_1 + 3.450x_2 + 4.702x_3 + 7.557x_4 + 4.518x_5 + 5.233x_6 + 2.090x_7 + 1.355x_8 + 1.680x_9 + 12.486x_{10} + 9.015x_{11} + 11.870x_{12} + 8.831x_{13} + 6.403x_{14} + 11.007x_{15} + 7.968x_{16} + 5.540x_{17} + 12.259x_{18} + 9.220x_{19} + 9.935x_{20} + 6.792x_{21} + 12.790x_{22} + 8.912x_{23} + 9.715x_{24} + 5.873x_{25} + 7.323x_{26} + 3.445x_{27} + 16.572x_{28} + 13.533x_{29} + 11.105x_{30} + 17.492x_{31} + 14.453x_{32} + 12.025x_{33})) \]

\[ g_{15}(x) = -(5.0 - (x_{34} + x_{35} + x_{36} + x_{37} + x_{38})) \]

\[ g_{16}(x) = -(196.9350 - (x_9 + x_{10})) \]
Chapter 10. Summary

This thesis has presented research into three techniques relevant to the application of evolutionary algorithms to fuzzy systems: multidimensional encoding; cooperative coevolution; and multiobjective optimisation. The inverted pendulum problem was used as to investigate the evolution of fuzzy systems throughout most of this thesis.

The final research chapter departed from the use of fuzzy systems to focus solely on constrained multiobjective optimisation, which is applicable to many real-world problems. This thesis presented Blended Rank Evolutionary Algorithm, a novel constrained multiobjective EA. BREA was applied to the non-linear crop rotation problem.

10.1 Evolution of fuzzy systems

10.1.1 Multidimensional encoding

Encoding bias results from arbitrarily encoding a solution’s decision-variables into a form expected by an EA. Encoding bias is expressed through the crossover operator. This work considered two types of encoding bias: epistatic bias and dimensional bias.

Epistasis is the effect one decision-variable has on the contribution of another decision-variable’s contribution to the fitness evaluation. Drastically changing the value of one of a pair of epistatic decision variables (as can happen during crossover) can have an unpredictable effect on fitness evaluation, making optimisation difficult due to the apportionment of credit problem. Therefore, encoding and crossover is usually designed
to exploit epistasis: by reducing the probability that epistatic decision-variables will be disrupted by crossover, the search can be greatly simplified.

**Dimensional bias** can be inadvertently introduced when a solution that is naturally represented by a multidimensional structure is encoded as the traditional single-dimensional vector of decision-variables commonly used in EAs. This causes some epistatic links to be preserved in the encoding (and able to be exploited) and other epistatic links to be broken (compounding the apportionment of credit problem).

A **multidimensional encoding**, together with a **multidimensional crossover operator**, can preserve epistatic links so that they can be exploited, whilst simultaneously avoiding arbitrary dimensional bias. As a fully specified fuzzy rulebase is a naturally multidimensional structure, this work hypothesised that multidimensional encoding and crossover would perform better than the traditional single-dimensional encoding of rulebase consequents.

Experimental results on function approximation problems supported this hypothesis. For a fully specified fuzzy rulebase a multidimensional encoding and crossover is more appropriate than a traditional vector encoding and \( n \)-point crossover, due to the elimination of dimensional encoding bias. Uniform crossover, which eliminates *all* encoding bias regardless of encoding, also performed very strongly, achieving better results than both multidimensional crossover and \( n \)-point crossover on the function approximation problems. The inverted pendulum control problem, with a much greater variability between runs, failed to show statistically significant differences between encodings and crossover operators, despite the prediction that the control problem would be more sensitive to exploitation of epistasis.
The overall conclusion is that encoding and crossover should be designed with consideration to what can be reasonably assumed about the problem domain; the practitioner should be wary of unsubstantiated, arbitrary design decisions that could result in undesirable bias. Better to adapt the algorithm to the problem, than the problem to the algorithm.

10.1.2 Cooperative coevolution of hierarchical rulebases
Hierarchical rulebases are a way of reducing the size of a fully-specified rulebase and also introducing modularity into the system. Rather than having a single flat rulebase that covers all inputs, a hierarchical rulebase is a number of smaller rulebases each covering a subset of inputs, with the output of one rulebase typically input to another. The assignment of inputs and outputs to rulebases defines the topology of the hierarchical rulebase. Unfortunately, there is no general method for designing topologies, but a rule of thumb is that interdependence between the inputs assigned to different rulebases should be minimised.

Cooperative coevolution is a way of reducing the size of an EA’s search space. Rather than having a single individual encode all decision-variables of a solution, cooperative coevolution defines the individuals of a population to encode only a subset of decision-variables, and multiple populations are evolved in parallel to search all decision-variables simultaneously. One individual is selected as a representative from each population, and the combined representatives form an entire solution. Unfortunately, there is no general method for decomposing a solution across multiple populations, but a rule of thumb is that epistasis between decision-variables in different populations should be minimised.
Hierarchical rulebases and cooperative coevolution share very similar goals and difficulties, and it was natural to combine the two techniques to use cooperative coevolution to evolve hierarchical rulebases.

10.1.2.1 Cooperative coevolution
This work tested the hypotheses that cooperative coevolution would make faster progress, and that cooperative coevolution produces better final solutions.

Experimental results on the inverted pendulum problem clearly showed that cooperative coevolution makes faster progress due to a reduced search-space.

The results also showed that coevolution does not necessarily produce better final solutions due to greater modularity. All algorithms produced insignificantly different solutions when allowed to run for long enough. However, this is likely problem-dependent; the problem decomposition used here (a hierarchical rulebase decomposition) was already modular.

10.1.2.2 Hierarchical design
Examination of the hierarchical rulebases revealed that all of the EAs tested were surprisingly flexible in using the intermediate output linking the two layers in the hierarchical topology, whilst still producing the same overall control strategy. However, this flexible interpretation of the intermediate output proved delicate, with the controller crashing the pendulum if the conditions were changed even slightly from conditions the controller was trained on. This highlights the importance of selecting test-cases carefully.

The interpretation of the intermediate variable is expected to be less flexible, but more robust, if trained on more test-cases.
10.2 Parallelism

Almost all evolutionary algorithms are trivially parallel: an individual can be evaluated independently of all other individuals in the population, and the task of fitness evaluation can therefore be trivially distributed across multiple nodes. When the cost of fitness evaluation is expensive compared to the cost of evolutionary operations and communications overhead (as is the case in the inverted pendulum problem), a simple master-slave model is well-suited for parallelising an EA.

However, when fitness evaluation is very cheap, the speedup from using the master-slave model diminishes, and may be overtaken by rising communications overhead (for example from slow or heterogeneous networks). In such cases, a coarser-grained model of parallelism, such as cooperative coevolution, can still realise speedup by minimising communications overhead.

10.3 Multiobjective optimisation

Multiobjective optimisation, particularly constrained multiobjective optimisation, is important for many real-world problems that are most naturally formulated with conflicting, non-commensurable objectives and multiple constraints.

A multiobjective EA produces a set of final solutions with diverse characteristics for the final decision maker to select from subjectively. The investigation of hierarchical rulebases identified that a range of controller behaviours were possible. This work initially introduced multiobjective optimisation to investigate the full extent of possible controller behaviours for the inverted pendulum.
10.3.1 Compatibility with previous techniques

Hierarchical rulebases were avoided in this study of Pareto multiobjective optimisation because of the interdependence imposed upon state-variables, biasing a solution’s behaviour to a certain region of the Pareto front. However, hierarchical rulebases are not incompatible with multiobjective optimisation in general; this bias may be acceptable or even desirable.

Cooperative coevolution appears to be incompatible with Pareto multiobjective optimisation, because cooperative coevolution approaches the apportionment of credit problem by selecting just one (or a very small number) representative individual from each population. It is not clear how a single representative should be selected to represent the diversity of solutions on the Pareto trade-off surface.

10.3.2 Formulating a multiobjective problem

The single-objective formulation of the inverted pendulum problem (which was given in chapter 5 and used in chapters 6 & 7) had to be reformulated as a multiobjective problem. What at first seemed to be good candidates for constraints – the penalty terms in the single-objective formulation – actually turned out to be redundant. Something that seemed to be a good candidate for an objective – the survival time, one of the fitness measures in the single-objective formulation – actually turned out to be a constraint. This highlights the different way of thinking about a problem when using multiobjective optimisation.

Objectives and constraints must be extracted from the problem description. Careful attention should be paid to making sure that the objectives and constraints accurately describe what is desired of a solution.
The problem description must unambiguously specify what constitutes feasibility, and how feasibility is different from optimality. Constraints should be able to differentiate infeasible solutions to guide the search to feasibility, and objectives should be able to differentiate feasible solutions to guide the search to optimality.

10.3.3 Diversity measures
This work investigated two diversity measures used to differentiate solutions inside a non-dominated rank: distance-to-neighbours, which is a measure based on the perimeter of a hyperbox containing a given solution but no others, and uniquely dominated hypervolume, which is a measure of the volume of objective space that a given solution dominates but that no other solution dominates. It was hypothesised that the uniquely dominated hypervolume would provide selection pressure for the final solution-set to more closely match the true Pareto front, but that the distance-to-neighbours would provide selection pressure for the final solution-set to be more uniformly distributed.

Experiments on the inverted pendulum problem revealed that all sets produced by the uniquely dominated hypervolume and distance-to-neighbours diversity measures were incomparable to one another in the Pareto sense (i.e., no solution set completely dominated another solution set).

However, some conclusions could be drawn regarding the general behaviour of the two diversity measures. It was observed that the regions where the solution sets overlapped were almost entirely dominated by sets using the hypervolume measure. Almost all of the (very few) non-dominated solutions in sets using the distance-to-neighbours measure were found in regions that the hypervolume sets didn’t find at all.
Confirming the hypothesis, it is concluded that the hypervolume measure produces a solution set with closer proximity to the Pareto front, whereas the distance-to-neighbours measure produces a solution set with better diversity. However, as the solution sets were always overlapping – incomparable in the Pareto sense – it cannot be concluded that one measure is better than the other; however, the above observations may influence the choice of measure.

10.4 Blended rank evolutionary algorithm

After investigating the multiobjective inverted pendulum problem, this work turned its attention away from the evolution of fuzzy systems and focussed entirely upon constrained multiobjective evolutionary algorithms.

This work characterises CMEAs in the context of the strategy they use to reconcile the conflicting goals of constraint satisfaction and objective optimisation. The commonly used extended dominance strategy rigidly prioritises constraint satisfaction, then objective optimisation, then diversity. This selection pressure is appropriate for a final solution set, but leaves the search susceptible to local optima in highly constrained problems. Blended space is an alternative strategy that maintains an adaptive trade-off between constraint satisfaction and objective optimisation.

Preliminary experiments on highly constrained problems – the restricted CTP-8 and the nonlinear crop rotation problem – demonstrated the superiority of a simple blended-space algorithm at avoiding constrained local optima.

In light of this, Blended Rank Evolutionary Algorithm (BREA) was developed to make use of different measures in different spaces, as appropriate for the desired behaviour in each space. In particular, BREA uses the blended space strategy to reconcile constraint
satisfaction and objective optimisation, allowing infeasible solutions to play a greater role in evolution. BREA also makes an explicit distinction between the evolving solution set and the final solution set, and applies different selection pressures to each. The fraction of the population subject to the final solution set’s selection pressure increases progressively over the duration of the run, until at the end of the run the entire population forms the final solution set. All of this is accomplished without additional parameters.

BREA offers several advantages (compared to extended-dominance in general and NSGA-II in particular) in problems that are highly constrained with many local optima, all realised through its ability to cross infeasible regions and avoid local optima:

10.4.1 Proximity
BREA consistently produced the solution sets with the closest proximity to the Pareto front; BREA’s solution sets dominated almost every other solution found in these comparisons. BREA’s mean hypervolume scores were consistently the best in all comparisons. This result may be attributed primarily to BREA’s ability to avoid local optima.

10.4.2 Confidence
BREA exhibited a lower variability between solution sets produced by independent runs. This is important for problems where the global optimum (i.e., the Pareto front) is not known a priori, and it cannot be known whether the result from any single run is only a local optimum. By avoiding local optima, and thereby reducing the variance between independent runs, BREA allows a greater confidence in the quality of any given solution set.
10.4.3 Evolutionary progress
BREA was able to make better use of an increased number of generations, due to its ability to avoid local optima and continue evolution toward the true Pareto front. This allows BREA to make full use of an allocated number of generations, or to make use of additional generations to improve a currently unacceptable solution set, without fear of futilely spending generations trapped in local optima.

10.4.4 Distribution
BREA’s solution sets had poorer objective-space distributions than NSGA-II, even though both algorithms used the same objective-space diversity measure. Broadly, this is because BREA does not apply the same selection pressure to the final solution set (in which objective-space diversity is important) as it does to the evolving solution set (in which objective-space diversity is not important). Specifically, this is because BREA avoids collapsing into a single non-dominated rank until the very end of the run, and objective-space diversity is only meaningful internally to a rank. In contrast, NSGA-II applies strong selection pressure to collapse into a single rank throughout the run, and once this is achieved objective-space diversity becomes the differentiating measure. In comparison to NSGA-II this point is largely moot, because when one solution set completely dominates another solution set (as was often the case), the relative distributions of each set are irrelevant. However, this shortcoming may be addressed by modifying BREA to spend more effort on the final solution set.

10.4.5 Computational overhead
BREA has a relatively high computational overhead, particularly in calculating decision-space diversity (which accounted for 2/3rd of the overhead in the crop rotation problem).
This is due to the larger number of measures it takes from each space at each generation, in order to better guide evolution. This increased overhead is expected to be masked by many expensive real-world fitness evaluations, but even in the extremely adverse case of the cheap crop rotation fitness evaluation (for which NSGA-II was able to have 10-fold increase in generations or an 8-fold increase in population size) BREA still outperformed NSGA-II given the same computational budget. Furthermore, it is argued that due to BREA’s ability to find superior solution sets with fewer total evaluations, BREA may be the superior choice especially when computational cost is very important – if it is assumed that computational cost becomes important primarily when fitness evaluations are very expensive.

10.5 Future research

There are many questions that have been raised in the previous chapters that invite further research, including:

- When is it worth exploiting deliberate epistatic encoding bias (using, for example, multidimensional crossover) versus removing all encoding bias (using uniform crossover)? Possibly in highly epistatic problems.

- The standard explorative representative selection scheme, for cooperative coevolution, forms just one trial solution from randomly selected representatives. If this implicit parameter is made explicit, what is an appropriate value given the trade-off of increased fitness evaluation costs for (presumably) greater robustness?

- The cooperative coevolutionary decomposition corresponded to the hierarchical rulebase decomposition, but hierarchical rulebase decomposition itself lacks
problem-independent guidelines. What other approaches to rulebase decomposition exist, and are they compatible with cooperative coevolution, as fully-specified hierarchical decomposition is?

- An appropriate model for parallelism depends on the cost of fitness evaluations (parallel code), the cost of evolutionary computation (serial code), and the hardware implementation (speed of processing nodes and speed of communications). In the inverted pendulum problem, the cost of fitness evaluations masked the cost of evolutionary computation and communications costs, making the master-slave model appropriate. What real-world problems exist, where computational cost is a real limiting factor, for which the coarser granularity of cooperative coevolution is useful?

One question that is particular to the context of this work is the combination of cooperative coevolution and multiobjective optimisation:

- Cooperative coevolution requires a single (or a few) representative individuals from each population in order to evaluate a trial solution. Pareto multiobjective optimisation works toward a population of diverse, equally desirable individuals. This diversity makes the apportionment of credit problem difficult to address with a limited number of representatives. Is there a way to apply cooperative coevolution to multiobjective optimisation?

There are also undoubtedly many ways to improve the CMEA proposed in this work, BREA. One suggestion follows:

- BREA’s distribution of the final solution set is relatively poor. An improved distribution might be achieved, at the cost of introducing a new parameter, by
specifying a number of generations at the end of the run during which all selection pressure is placed on the final solution set.

And finally, BREA should be applied to different real-world and benchmark problems (for example [114]) to determine (and, hopefully, demonstrate) the worth of the blended-space approach to constrained multiobjective optimisation.
Bibliography

34. Eriksson, R. and Olsson, B., Cooperative Coevolution in Inventory Control Optimisation. in *Third International Conference on Artificial Neural Networks and Genetic Algorithms*, (Norwich, UK, 1997), Springer-Verlag.


72. Pareto, V. *Cours D'Economie Politique*, Rouge, Lausanne, Switzerland, 1896.
75. Potter, M.A. The Design and Analysis of a Computational Model of Cooperative Coevolution, George Mason University, 1997.
89. Sivakumar, G., Vadakkepat, P. and Al Mamun, A. Genetic Algorithm Based Heuristic Fuzzy Pattern Classifier.
91. Stonier, R.J., Evolutionary Learning of Fuzzy Logic Controllers over a region of initial states. in *Congress on Evolutionary Computation*, (Washington, 1999), 2131 - 2138.
96. Stonier, R.J. and Żajaczkowski, J., A Cooperative Co-evolutionary Algorithm For Hierarchical Fuzzy Control Of The Inverted Pendulum. in *International Congress on Evolutionary Computation*, (Vancouver, Canada, 2006).


107. Young, N., Blended Ranking to Cross Infeasible Regions in Constrained Multiobjective Problems. in *Computational Intelligence for Modelling Control and Automation*, (Vienna, Austria, 2005), IEEE-Press.

108. Young, N. and Stonier, R.J., Blended Rank Evolutionary Algorithm for the Constrained Multiobjective Crop Rotation Problem. in *Computational Intelligence for Modelling Control and Automation*, (Sydney, Australia, 2006), IEEE-Press.


110. Young, N. and Stonier, R.J., Multi-dimensional encoding to reduce bias in fuzzy knowledge-bases. in *7th Asia-Pacific Conference on Complex Systems*, (Cairns, 2004), ISBN 1 87667496 2, 196-206.


117. Zitzler, E. and Thiele, L., Multiobjective optimization using evolutionary algorithms - a comparative case study. in Fifth International Conference on Parallel Problem Solving from Nature (PPSN-V), (Berlin, Germany, 1998), Springer, 292-301.