Adaptive Techniques for Enhancing the Robustness and Performance of Speciated PSOs in Multimodal Environments

A thesis submitted for the degree of

Doctor of Philosophy

Stefan Bird B.AppSci. (Computer Science), B.AppSci. (Geomatics),

School of Computer Science and Information Technology,

Science, Engineering, and Technology Portfolio,

RMIT University,

Melbourne, Victoria, Australia.

14th February, 2008

Declaration

I certify that except where due acknowledgement has been made, the work is that of the author alone; the work has not been submitted previously, in whole or in part, to qualify for any other academic award; the content of the thesis is the result of work which has been carried out since the official commencement date of the approved research program; and, any editorial work, paid or unpaid, carried out by a third party is acknowledged.

[illegible]

Stefan Charles Bird

School of Computer Science and Information Technology

RMIT University

14th February, 2008

Acknowledgments

While the people who have helped me during this journey are too numerous to mention, I would like to give special thanks to the following:

- Dr. Xiaodong Li for being my primary supervisor and all the challenges that entails
- Dr. Vic Ciesielski for the feedback he provided on my various papers and this document
- My family and friends for the care, hope and support they have given throughout my studies

Credits

Portions of the material in this thesis have previously appeared in the following publications:

- S. Bird and X. Li. Adaptively choosing niching parameters in a PSO. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 3-10, 2006.
- S. Bird and X. Li. Enhancing the robustness of a speciation-based PSO. In *Proceedings of the Congress on Evolutionary Computation*, pages 843-850, 2006.
- S. Bird and X. Li. Informative performance metrics for dynamic optimisation problems. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 18-25, 2007.
- S. Bird and X. Li. Using regression to improve local convergence. In *Proceedings of the Congress on Evolutionary Computation*, pages 1555-1562, 2007.

This work was supported by the Evolutionary Computing and Machine Learning group at RMIT University.

Note

Unless otherwise stated, all fractional results have been rounded to the displayed number of decimal figures.

Contents

Abstract **1**

1 Introduction **4**

1.1 Motivation 6

1.2 Objectives 9

1.3 Methodology 9

1.4 Contributions 10

1.5 Thesis Outline 11

2 Literature Review **13**

2.1 No Free Lunch 14

2.2 Static vs Dynamic Environments 16

2.3 Measuring Performance 17

2.3.1 Measuring error 18

2.3.2 Measuring diversity 19

2.3.3 Measuring performance in static environments 21

2.3.4 Measuring performance in dynamic environments 22
Online performance 23
Collective Mean Fitness 24
Mean Tracking Error 24
Current Error 25
Offline Error 25
2.3.5 The Moving Peaks test suite 27
2.4 Optimisation Algorithms 28
2.4.1 Random Search 29
2.4.2 Local Search 29
2.4.3 Simulated Annealing 30
2.4.4 Genetic Algorithms 31
2.4.5 Differential Evolution 32
2.4.6 Evolutionary Strategies 33
2.4.7 Genetic Programming 33
2.4.8 Ant Colony Optimisation 34
2.4.9 Particle Swarm Optimisation 35
Constraining Velocity 37
Controlling Information Spread 39
2.4.10 Using Meta-models 42
2.4.11 Parameter Control 44
2.5 Dynamic Optimisation 45

2.5.1 Detecting change 45
2.5.2 Dealing with change 46
Reintroducing diversity 46
Maintaining diversity 47
Remembering past solutions 47
Multiple populations 48
2.6 Multiobjective Optimisation 48
2.7 Multimodal Optimisation 49
2.7.1 Crowding 50
2.7.2 Fitness Sharing 51
2.7.3 Clearing 52
2.7.4 The Island Model 53
2.7.5 Restricted Tournament Selection 53
2.7.6 Multinational GAs 54
2.7.7 Sequential Niching 55
2.7.8 Clustering 56
2.7.9 NichePSO 57
2.7.10 SPSO and SDE 59
2.7.11 FER-PSO 61
2.7.12 mQSO 63
2.8 Summary 64

3 Speciation-based PSOs 65

3.1 Overview . . . 66

3.2 Challenges of multimodal functions . . . 67

3.3 SPSO . . . 68

3.3.1 Variations . . . 69

Limiting the species size . . . 69

Stable species seeds . . . 72

Replacing the worst species . . . 73

Quantum particles . . . 73

3.3.2 Strengths . . . 74

3.3.3 Weaknesses . . . 75

3.4 Proposed speciated PSOs . . . 76

3.4.1 ESPSO . . . 77

Increasing the robustness of r . . . 77

Strengths . . . 81

Weaknesses . . . 84

3.4.2 ANPSO . . . 84

Strengths . . . 89

Weaknesses . . . 90

3.5 Summary . . . 91

4 Performance metrics for dynamic environments 92

4.1 Measuring convergence speed . . . 93

4.1.1 Advantages . . . 94

4.1.2 Disadvantages . . . 98

4.2 Relating peak cover and BKPE to offline error . . . 98

4.3 Improving algorithm performance . . . 100

4.4 Summary . . . 101

5 Experiments and analysis 102

5.1 Proposed speciation algorithms . . . 102

5.1.1 Experimental setup . . . 103

Testing robustness . . . 106

Higher-dimensional functions . . . 107

Comparing to SPSO . . . 109

5.1.2 Results and analysis . . . 109

Robustness of s . . . 110

Robustness of P_{max} . . . 112

Effect of population size . . . 112

Robustness of r . . . 118

Performance on low-dimensional functions . . . 122

Performance on higher-dimensional functions . . . 123

Overall . . . 124

5.2 Informedness of performance metrics for dynamic environments . . . 126

5.2.1 Experimental setup . . . 126

5.2.2 Results and analysis . . . 128

Correlation with offline error 129

Algorithm performance 132

Improving the algorithms 135

5.3 Summary 136

6 Improving local convergence 138

6.1 Using regression to locate optima 139

6.2 Advantages 145

6.3 Disadvantages 146

6.4 Summary 147

7 The effectiveness of regression 148

7.1 Experimental setup 149

7.1.1 Static Functions 149

7.1.2 Moving Peaks 150

7.2 Results 154

7.2.1 Static Functions 154

Low dimensional landscapes 155

High dimensional landscapes 155

Sensitivity to e 156

7.2.2 Moving Peaks 157

Increasing convergence speed 157

Number of peaks 160

Peak movement severity 160
Dimensionality 161
Peak shape 162
Asymmetric peaks 164
Adding local optima 165
Asymmetric peaks with local optima 165
Sensitivity to e 165
Comparing to mQSO 169
7.3 Summary 169

8 Conclusion 171

8.1 Addressing our research objectives 172
8.1.1 Reducing the reliance on speciation parameters 173
8.1.2 Designing a metric to exclusively measure convergence speed 174
8.1.3 Performing a comparative study of several algorithms 175
8.1.4 Using performance metrics to profile an algorithm 175
8.1.5 Improving local convergence speed 176
8.1.6 Extending the Moving Peaks benchmark 177
8.2 Future directions 178
8.2.1 Speciation algorithms 178
8.2.2 Performance metrics 179
8.2.3 Regression 179
8.3 Summary 180

A Static test functions **181**

Bibliography **183**

List of Figures

2.1 Similarity of fitnesses for nearby points 15

2.2 Fitness vs. distance error. 18

2.3 Calculating peak cover 20

2.4 Error over time. The environment is changing every 5000 evaluations. 23

2.5 Moving Peaks 27

2.6 How the different algorithms are related 28

2.7 Common neighbourhood topologies. 41

2.8 SPSO/SDE species seeds 60

3.1 Inverted Shubert 2D fitness landscape 67

3.2 Single funnel, multimodal fitness landscape 70

3.3 The effects of setting r too large in SPSO 76

3.4 ESPSO example, step 1 78

3.5 ESPSO example, step 2 78

3.6 ESPSO example, step 3 78

3.7 ESPSO example, step 4 79

3.8 ESPSO example, step 5 79

3.9 ESPSO example, step 6 79

3.10 ANPSO: Calculating r 86

3.11 ANPSO: Forming species 86

4.1 Calculating BKPE 95

4.2 Relating peak cover and BKPE to offline error 99

5.1 The fitness landscape of Branin RCOS 104

5.2 The fitness landscape of Himmelblau 105

5.3 The fitness landscape of Six-Hump Camel Back 105

5.4 The fitness landscape of Deb's First Function 106

5.5 The Modified Humps landscape 109

5.6 The robustness of s: percentage of successful runs (ESPSO) 111

5.7 The robustness of s: percentage of successful runs (ANPSO) 111

5.8 The robustness of s: number of evaluations (ESPSO) 113

5.9 The robustness of s: number of evaluations (ANPSO) 113

5.10 The robustness of P_{max}: percentage of successful runs (ESPSO) 114

5.11 The robustness of P_{max}: percentage of successful runs (ANPSO) 114

5.12 The robustness of P_{max}: number of evaluations (ESPSO) 115

5.13 The robustness of P_{max}: number of evaluations (ANPSO) 115

5.14 The effect of population size: percentage of successful runs (ESPSO) 117

5.15 The effect of population size: percentage of successful runs (ANPSO) 117

5.16 The effect of population size: number of evaluations (ESPSO) 119
5.17 The effect of population size: number of evaluations (ANPSO) 119
5.18 The effect of population size: success rate on Inverted Shubert 2D 120
5.19 The effect of population size: number of evaluations on Inverted Shubert 2D . 120
5.20 The robustness of r: percentage of successful runs (ESPSO) 121
5.21 The robustness of r: percentage of successful runs (SPSO) 121
5.22 The effect of r on ESPSO: number of evaluations 122
5.23 The effect of dimensionality: success rate 125
5.24 The effect of dimensionality: evaluations 125
5.25 How BKPE and peak cover correlate with offline error 127
5.26 Correlation between PB and offline error 130
5.27 The correlation of each metric for different numbers of peaks. 131
5.28 Correlation between peak cover and BKPE 131
5.29 Peak cover and BKPE: 10 peaks 133
5.30 Peak cover and BKPE: 100 peaks 133

6.1 Regression example, step 1 143
6.2 Regression example, step 2 145

7.1 Determining the peak shape using α_d 151
7.2 An example of an asymmetric peak 152
7.3 An example of an asymmetric peak with local optima 152
7.4 Number of evaluations needed for Humps 5D 157

7.5 Current error over time on Moving Peaks scenario 2 158
7.6 Current error over time after the first peak movement 158
7.7 Adding the regression substantially reduces offline error 161
7.8 The benefit of the regression for different movement severities 162
7.9 Effectiveness of the regression for higher dimensions 162
7.10 Offline error for different peak shapes. 163
7.11 Performance on symmetric peaks for different values of Max_β 166
7.12 Performance on symmetric peaks for different values of Max_γ 166
7.13 Performance on asymmetric peaks for different values of Max_β 167
7.14 Performance on asymmetric peaks for different values of Max_γ 167
7.15 Offline error for Moving Peaks Scenario 2 for different values of e. 168
7.16 Offline error for asymmetric peaks with waves for different values of e 168

List of Tables

5.1 Number of evaluations required to find all global peaks 123

5.2 Moving Peaks Scenario 2 parameters 128

5.3 Algorithm performance with 10 peaks 134

5.4 Correlation with offline error for 10 peaks. 135

7.1 Regression performance on low dimensional static functions 155

7.2 Regression performance on the high dimensional Humps function 156

7.3 Regression performance on asymmetric peaks. 164

7.4 Comparing against mQSO: Severity 169

7.5 Comparing against mQSO: Number of peaks 170

Abstract

This thesis proposes several new techniques to improve the performance of speciated particle swarms in multimodal environments. We investigate how these algorithms can become more robust and adaptive, easier to use and able to solve a wider variety of optimisation problems. We then develop a technique that uses regression to vastly improve an algorithm's convergence speed without requiring extra evaluations.

Speciation techniques play an important role in particle swarms. They allow an algorithm to locate multiple optima, providing the user with a choice of solutions. Speciation also provides diversity preservation, which can be critical for dynamic optimisation. By increasing diversity and tracking multiple peaks simultaneously, speciated algorithms are better able to handle the changes inherent in dynamic environments.

Speciation algorithms often require a user to specify a parameter that controls how species form. This is a major drawback since the knowledge may not be available a priori. If the parameter is incorrectly set, the algorithm's performance is likely to be highly degraded. We propose using a time-based measure to control the speciation, allowing the algorithm to define species far more adaptively, using the population's characteristics and behaviour to

control membership.

Two new techniques presented in this thesis, ANPSO and ESPSO, use time-based convergence measures to define species. These methods are shown to be robust while still providing highly competitive performance. Both algorithms effectively optimised all of our test functions without requiring any tuning.

Speciated algorithms are ideally suited to optimising dynamic environments, however the complexity of these environments makes them far more difficult to design algorithms for. To increase an algorithm's performance it is necessary to determine in what ways it should be improved. While all performance metrics allow optimisation techniques to be compared, they cannot show how to improve an algorithm. Until now this has been done largely by trial and error. This is extremely inefficient, in the same way it is inefficient trying to improve a program's speed without profiling it first.

This thesis proposes a new metric that exclusively measures convergence speed. We show that an algorithm can be profiled by correlating the performance as measured by multiple metrics. By combining these two techniques, we can obtain far better insight into how best to improve an algorithm. Using this information, we then propose a local convergence enhancement that greatly increases performance by actively estimating the location of an optimum.

The enhancement uses regression to fit a surface to the peak, guiding the search by estimating the peak's true location. By incorporating this technique, the algorithm is able to use the information contained within the fitness landscape far more effectively. We show that by combining the regression with an existing speciated algorithm, we are able to vastly improve the algorithm's performance. This technique will greatly enhance the utility of PSO on problems where fitness evaluations are expensive, or that require fast reaction to change.

Chapter 1

Introduction

Numerical *optimisation* is a process of finding a solution to a numeric problem [Wilde, 1964]. It may be locating the highest value of a mathematical function, finding the most profitable allocation of resources, discovering hidden trends within a dataset, classification tasks, or any number of other problem types. While the problem types and structures are extremely varied, they have two things in common: the quality of a potential solution can be rated, and the goal is to find the highest quality solution within the given constraints.

Evolutionary Algorithms (EA) use a population of candidate solutions to locate an *optimum* over many *timesteps* or generations [Michalewicz, 1996]. The fittest members of the previous generation are used to create the next; by continually combining and modifying the best individuals we are able to hone in on an optimum. Using a population allows the algorithm to explore multiple regions of the *decision space*[1] simultaneously, increasing the chance of finding the best solution.

[1]An N-dimensional space that represents every possible solution to a problem, also known as a *search space*.

Swarm Intelligence (SI) follows a similar approach to EA, except that each individual is autonomous. To decide where to explore next, the individuals make their own decisions using information gained from other members of the population. In Particle Swarm Optimisation (PSO) [Kennedy and Eberhart, 2001] each member travels around the decision space and is called a particle. Each particle represents the solution at its location in the decision space. To do this, in addition to their location, particles have a velocity term which defines where they will explore next. Each particle also has a set of neighbours with whom it communicates. The quality and location of its neighbours' best solutions influences each particle's velocity. PSO algorithms offer fast convergence while being resistant to becoming stuck on a non-global optimum.

Many problems have multiple optimal or near-optimal solutions. In these environments it is often desirable to locate and track the multiple optima. This may be to reduce the risk of having the entire population converge on a non-global optimum, or to present the user with a set of choices instead of a single answer [Mahfoud, 1995a].

Tracking multiple solutions is especially valuable when dealing with *dynamic* environments, that is problems that change over time. In this situation, maintaining individuals on local optima can prove advantageous if a local optimum becomes or leads to a global one at some point in the future [Li et al., 2006].

The basic Particle Swarm is designed to converge on a single optimum [Clerc, 1999]. One method of allowing a PSO to converge on multiple optima is speciation. In nature, separate species evolve to exploit different niches in the environment. For example tigers that live in snowy areas have evolved white fur to better blend with their environment. In

PSO, speciation dynamically divides the members into separate groups or *species*. Only members of the same species are able to communicate with each other, allowing each species to converge on a different optimum without interference.

This thesis proposes several methods to improve the performance of PSO on multimodal and dynamic environments. First we explore how the number of user-set parameters required by the algorithm can be reduced. Most current speciated PSO algorithms require the user to set parameters in order to control which particles belong to which species. The ideal parameter values vary from problem to problem and are often difficult to determine. By alleviating this burden, we allow the algorithm to be applied without requiring parameter tuning, greatly increasing its robustness.

We also propose a new method that allows an algorithm to be profiled, providing valuable insight into how an algorithm's performance can be increased. We then use this technique to improve an existing speciated algorithm. By adding a regression heuristic to estimate the shape of a peak, we can vastly reduce the number of evaluations needed to locate the optimum.

These techniques enhance the usefulness of speciated algorithms, allowing them to efficiently solve an even wider variety of optimisation problems.

1.1 Motivation

In many environments, both artificial and real-world, there exist multiple global or near-global optima. Often it is desirable to locate all of these, or at least a number of them. When solving a real-world problem, the model used to evaluate the fitness of a solution often has to be simplified – many aspects of the problem are too difficult or impossible to include.

For example, a vehicle manufacturer may want to reduce wind drag by modifying a car's body shape. However changing the body shape also changes its aesthetics. While performance is important, a car that is unappealing will not sell well. Determining the attractiveness of a car is generally beyond the model's scope, yet it is still an important factor. If we are able to present the user with a number of "good" solutions, he or she can then choose a solution by taking into account any factors external to the model [Parmee et al., 1994]. By developing a selection of body shapes with similar aerodynamic performance, the user can choose the one that is most visually appealing, without having to continually guide the computer on this aspect.

Speciation allows the algorithm to locate and present a number of different solutions. It also reduces the risk of *premature convergence*, that is where the entire population has become stuck in a local optimum [Goldberg et al., 1992]. While an individual species may still converge on a poor solution, it does not stop the rest of the population from exploring other areas. Speciation also provides information on the algorithm's performance as it runs. Some algorithms will deliberately kill a poorly performing species to avoid wasting evaluations on low quality solutions.

Many existing speciated algorithms require the user to set parameters a priori. An example of such a parameter is the species radius, that is how different a candidate solution can be from the species' centre while still being considered part of that species. Correctly setting this parameter requires detailed knowledge of the problem, knowledge which is usually only obtainable through extensive trial and error. If the parameter is set too large the algorithm will not be able to differentiate nearby peaks. If it is too small many or all of the species will become trapped in local optima, unable to locate any acceptable solutions at all. Even in cases where the desired optima are found, an incorrect parameter value can result in the algorithm requiring many more fitness evaluations to optimise the function, reducing its usefulness.

By developing algorithms that provide good performance without requiring these parameters to be set, we free the user from a great burden. An algorithm that does not require tuning for each problem is substantially more useful than one that does. It can be more rapidly deployed in real world situations, and requires far less user-training and domain specific knowledge to be successfully applied.

Another issue with speciated algorithms is that the population is divided between the multiple species. This means that the convergence speed on each optimum is reduced relative to the overall number of fitness evaluations used. In addition, smaller population sizes lower the PSO's effectiveness by reducing the amount of available information.

To increase performance further we need to better use the information we already have. In EAs and PSOs, knowledge about a point is often discarded as the population is evolved over successive timesteps. This is very wasteful as these points are likely to still be useful to the optimisation.

We can prevent this waste by remembering the fittest known points, even after the main algorithm has discarded them. The speed at which the optimum is located can be vastly improved by employing a regression method. The regression uses these points to estimate the shape of the peak, and thus the location of the highest point. This allows the optimal solution to be efficiently determined, even with the very limited population available to an individual species.

1.2 Objectives

The primary objective of this thesis is to improve the effectiveness of speciated PSOs by reducing the amount of tuning required and increasing performance. More specifically:

1. Investigate how the reliance on user-specified parameters can be reduced or removed, allowing a speciated algorithm to provide acceptable performance without being tuned.
2. Develop a new performance metric that exclusively measures convergence speed.
3. Perform a comparative study of several algorithms to determine their relative strengths and weaknesses.
4. Determine how metrics can be more effectively utilised, allowing us to see how best to improve an algorithm.
5. Establish how a speciated algorithm's performance can be improved to more quickly locate the multiple optima.
6. Extend an existing dynamic optimisation benchmark, increasing its flexibility to allow more challenging fitness landscapes to be created.

1.3 Methodology

As this thesis is primarily devoted to improving speciated algorithms, some means of evaluating performance is required. We have used a test-bed of well-known functions, chosen to represent the challenges a speciated algorithm is likely to face. These challenges include high multimodality, small catchment areas, clustered peaks, high dimensionality and dynamic environments.

In achieving our aims we will employ several techniques such as speciation, sorting and regression. We will evaluate the success of our algorithms by comparing the number of evaluations required to locate all of a test function's optima with existing algorithms, as well as the percentage of successful runs. For the dynamic optimisation problems we will use offline error to measure performance. Our proposed metric will be evaluated by the correlation between it and the standard offline error score for each test run. Finally we will measure the efficacy of our local convergence technique using offline error.

1.4 Contributions

The main contributions of this thesis are:

- The development of a new speciation based PSO algorithm that adaptively determines the species size parameter using population statistics.
- Enhancing an existing speciation based PSO algorithm by greatly reducing the sensitivity of its speciation parameter.
- Demonstrating that using a time-based method to control species formation allows an algorithm to become far more robust while still providing good performance on multimodal environments.
- The creation of a new metric to exclusively measure the convergence speed of an algor-

ithm.

- Demonstration of the strengths and weaknesses of several algorithms using the above metric in addition to an existing population diversity metric.
- The development of a way to profile an algorithm, showing how it can best be improved.
- The creation of a local search enhancement using regression, significantly improving the performance of an existing speciated PSO.
- Extending an existing test function suite, allowing the creation of more difficult and multimodal fitness landscapes.

1.5 Thesis Outline

Chapter 2 provides the background of numeric optimisation, focussing on evolutionary computing algorithms, speciation and performance metrics.

Chapter 3 proposes ESPSO and ANPSO, two speciated Particle Swarm algorithms whose parameters are very robust, reducing the user's burden.

Chapter 4 describes how the use of performance metrics that measure only specific aspects of an algorithm's behaviour can lead to improved understanding of the algorithm's strengths and weaknesses, as well as suggest how it can be best improved.

Chapter 5 shows the effectiveness of the algorithms and metrics presented in Chapters 3 and 4.

Chapter 6 proposes using regression to improve our tracking of a moving optimum by enhancing the local convergence speed.

Chapter 7 tests the regression method presented in the previous chapter, showing a vast performance improvement over the current state-of-the-art algorithms.

Chapter 8 summarises the methods presented in Chapters 3, 4 and 6, provides future research directions and our concluding remarks.

Chapter 2

Literature Review

Optimisation is a very important part of many disciplines. Often the user is faced with a problem they are unable to solve manually. It may be finding the best time to fire the spark plugs in a petrol engine, the best shape for an aircraft's wing or the most efficient way to pack a truck. In each case, there is a *search space* of possible solutions within which we want to find the optimal point.

Central to all optimisation methods is a feedback mechanism that tells the algorithm how good a particular candidate solution is. This mechanism is known as the evaluation function. Its purpose is to take a given candidate solution and return an evaluation of its quality or *fitness*. By analysing the relative quality of points throughout the search space we can visualise a fitness landscape with peaks and troughs representing the regions of good and poor quality solutions respectively.

The algorithm uses the fitness of each point to decide the most promising solution to try next. The evaluation function itself represents a black-box – it may be as simple as

$f(x) = x^2$, or as complex as an exact model of a physical process. In many cases evaluating a candidate solution is computationally expensive; using an algorithm that can quickly find an acceptable solution is highly desirable.

This thesis is directed at improving speciated algorithms in *multimodal* environments, that is where the problem to be solved has more than one global optimum. We will be proposing algorithms that reduce the number of parameters that must be set a priori, making them far more adaptive to change. We will also propose a technique that allows existing algorithms to converge far more rapidly on an optimum.

This chapter provides an introduction to optimisation methods and performance measurement, primarily discussing algorithms within the Evolutionary Computing (EC) umbrella. The final portion of this chapter is dedicated to algorithms that are able to locate multiple optima in parallel.

2.1 No Free Lunch

All optimisation algorithms make assumptions about the function they are optimising. The most common assumption is that similar candidate solutions will have similar fitnesses, as shown in Figure 2.1. The extension of this is that good candidate solutions indicate areas of high fitness and it is useful to focus the search there. An algorithm's assumptions are often implicit: for example because a random search does not use the knowledge gained from previous guesses in any way, it implicitly assumes that the fitness of a candidate solution does not provide any information about nearby points.

An algorithm will generally give the best performance on problems that match its assump-

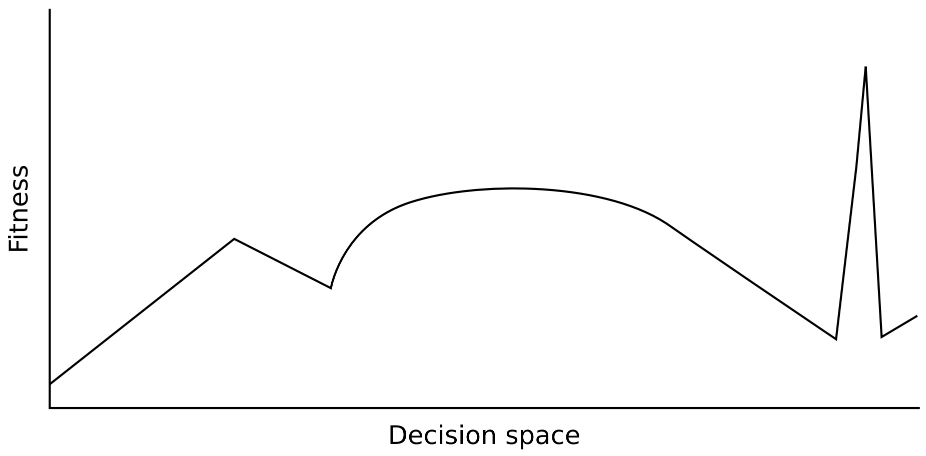


Figure 2.1: Adjacent points in the decision space tend to have similar fitnesses. The peak on the right is the true optimum, however since it is very narrow it is difficult to locate.

tions. Conversely an algorithm will usually perform poorly if the problem it is optimising behaves radically differently to what it expects. In the set of all problems with finite search spaces, we can find every possible fitness landscape. Therefore there are no assumptions that are valid for all landscapes. For example, an algorithm that assumed searching the fit areas would lead to the global optimum would perform poorly on Figure 2.1 – it is more likely to become stuck on the second best optimum in the middle of the landscape.

The No Free Lunch (NFL) theorem [Wolpert and Macready, 1997] states that when amortised across every possible problem, all optimisation algorithms perform equally. An algorithm that has good performance on some problems will have below average performance on others – the sum of all performances across all problems will be the same for every algorithm.

NFL is limited however. As it is only valid when taken across every possible fitness landscape, it says nothing when the domain is restricted to a certain class of problems. The vast majority of possible fitness landscapes are random; there is little to no correlation between points in the search space. These problems hold little interest for optimisation researchers as there is no exploitable information – a random search is likely to perform just as well if not better than any given algorithm.

Most real-world problems have much smoother landscapes – for example changing the spark plug timings by a millionth of a second is unlikely to make a noticeable difference to engine performance. This property allows us to ignore the difficulty of solving noise-like functions, but also nullifies NFL since we now only care about performance on a small portion of the possible problems. It still serves to remind us however that improving an algorithm's performance on a certain set of problems does not imply it will enjoy greater success across all other interesting problems. There is no "one true algorithm" – each method has its own advantages and disadvantages, and it is beneficial to know as much as possible about a given problem before deciding which algorithm would best solve it.

2.2 Static vs Dynamic Environments

The distinction between a static and a dynamic environment is simple: A static environment is one which does not change over time. The quality of a candidate solution in a static environment does not vary depending on when you measure it. In a dynamic environment, it can. This means that when performing the optimisation we are always racing against the clock; if we take too long to find an adequate solution, we will be unable to exploit it before the environment changes again.

A classic example of dynamic problem optimisation is the job scheduling problem [Branke, 2002]. In this problem there are a number of machines which must complete a set of jobs in an efficient manner. The nature of the problem changes over time as old jobs are completed, new jobs are added and machines break down or are repaired. In particular, the loss of a machine may require many jobs to be reassigned to other machines, drastically changing the schedule.

The cost of changing a schedule makes treating this problem as a static environment very inefficient. In many cases goods and items must be physically moved from one machine to another, requiring both time and labour. The larger the difference between two schedules, the more costly it is to move between them. By treating the schedule as a dynamic problem, we reduce these costs because the new schedule will usually be an adaption of the existing one and thus largely similar. If we were to treat it as a series of static problems, each new schedule is developed from scratch and may be very different from its predecessor. As a factory's scheduling efficiency can have a marked impact on both its output and profitability, this problem has wide commercial applicability and has been the subject of much research [Zhu and Wilhelm, 2006].

2.3 Measuring Performance

Whether the problem is static or dynamic, in order to determine the best algorithm to solve it we need some method of rating performance. Several metrics have been developed which measure different aspects of an algorithm's performance.

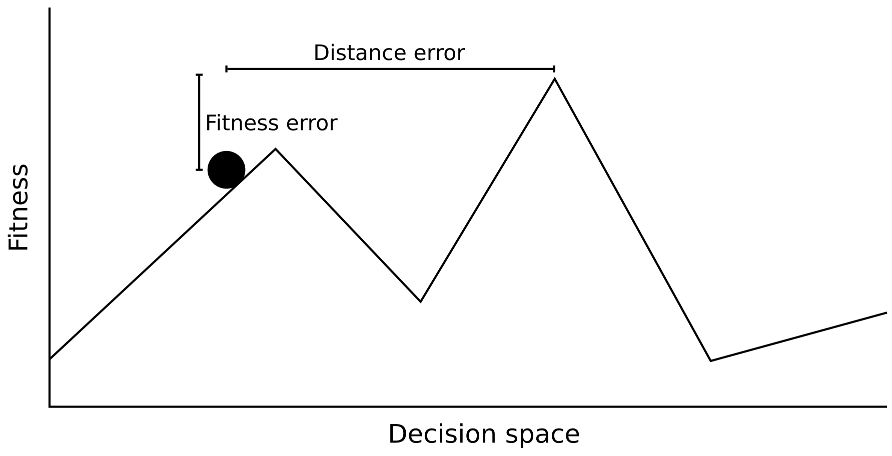


Figure 2.2: Fitness vs. distance error.

2.3.1 Measuring error

Most metrics operate on the basis of error – the difference between the global optimum and the best found solution. The error can be measured in one of two ways: fitness or distance. The fitness error measures the difference in terms of solution quality: in the presence of multiple "good" solutions, an algorithm will not be overly penalised for failing to find the best one, as long as the solution it does find is of similar fitness.

Distance error does the opposite: it is determined by how far apart the best-found and global solutions are in the decision space. To perform well an algorithm must find the correct solution, even in the presence of other solutions with similar quality. Converging on a solution on the other side of decision space to the global best will result in a large penalty regardless of the found solution's fitness, as in Figure 2.2.

2.3.2 Measuring diversity

In many EC algorithms, a population of candidate solutions is maintained. At each timestep or generation, the fitness of each individual is calculated. At the end of the generation the algorithm uses this information to determine the next set of points to try. These new points will form the population for the next generation. Using a population allows the algorithm to make more informed decisions – multiple areas in the search space can be explored simultaneously and information from one region can be used to improve the search in another.

The diversity of a population is how spread out the individuals are in the search space. It is important for an algorithm to maintain population diversity, especially when optimising dynamic environments. It is usually not desirable to have the entire population converged on a single optimum, as there is no guarantee the optimum is the best one. *Premature convergence*, that is when all of the individuals occupy the same local peak, is a difficulty all population-based algorithms must face.

A general measure of diversity is population entropy, as used in [Yao, 1993] for example. The entropy of a dataset is the number of bits, on average, that are needed to encode it [Cover and Thomas, 2006]. A large entropy value indicates the population is diverse. This measure is imperfect however, as it cannot distinguish between a population that is spread through the search space and one that consists of several tight clusters [Morrison, 2004]. Where the number and location of the optima is known, a simple yet informative measure of diversity is *peak cover* [Branke, 2002]. It reports what percentage of the optima were represented on average over the entire run. A peak is covered when there is at least one individual within

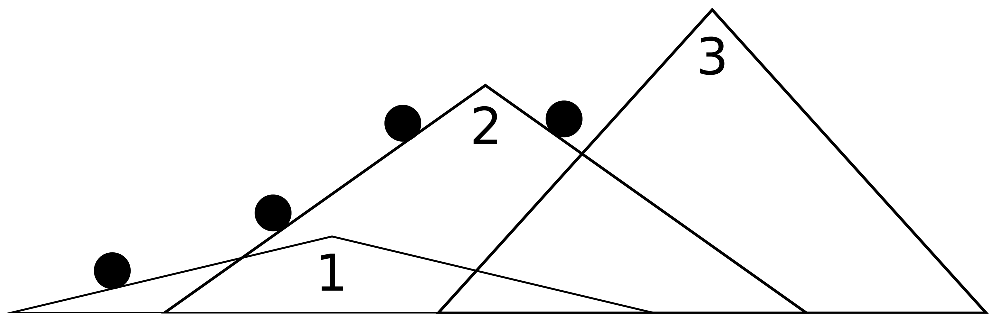


Figure 2.3: Half of the non-hidden peaks are represented by the population, giving a peak cover of 0.5 (50%). Peak 1 is ignored because it's hidden by Peak 2.

its catchment area. For evaluation functions where peaks can become submerged or hidden below other parts of the fitness landscape, the hidden peaks are ignored, as in Figure 2.3. This prevents the metric penalising an algorithm for not tracking a peak that isn't visible anyway. Peak cover is calculated at the end of each timestep using Equation (2.1).

$$\textit{peak cover} = \frac{\textit{covered peaks}}{\textit{visible peaks}} \tag{2.1}$$

Having a high peak cover is an advantage in many dynamic environments, since a currently suboptimal peak may lead to the global optimum in the future. An algorithm that tracks more peaks is better positioned to adapt once a change occurs; it has a higher chance of already covering the new best peak.

Peak cover is useful in that it isolates an aspect of an algorithm's behaviour that is not shown by most metrics. For problems where the global optimum makes large movements, maintaining population diversity is critical in order to quickly locate the optimum's new position. By measuring only this aspect of an algorithm, we are able to build a better picture of its behaviour, which problems it might be suited to and how it might be improved.

There is a disadvantage though, in that it is necessary to know the location of each significant peak. Even on static test functions, defining the catchment area of each peak can be difficult and may have to be done manually. On dynamic test functions this is often impractical as the environment changes are usually stochastic in nature. Also, not all peaks have the same height; for many problems it may not be worthwhile tracking every single peak since the lower peaks are unlikely to become global optima in the near future. Peak cover does not take this into account; an algorithm that tracks the fittest 5 peaks will score exactly the same as an algorithm that tracks the 5 least fit peaks.

Peak cover also makes no distinction between a peak that is represented by a single individual at the very edge of its catchment area or by a group of individuals near its peak. This can lead to algorithms that achieve good peak covers but are unable to effectively exploit the diversity.

Despite these issues, it is still a useful metric for certain test functions. In Chapter 4 we will introduce peak cover's complement, a metric that measures only convergence speed.

2.3.3 Measuring performance in static environments

Arguably the most important metric in a static environment is success rate, that is how often the algorithm is able to locate an optimum to a given accuracy. The accuracy is measured in terms of either distance or fitness error and expressed as a percentage or the actual number of successful or failed runs.

The most commonly used metrics show performance once a specified condition has been satisfied. This condition may be the error being reduced to a certain level, or a set number of fitness evaluations having been performed. For the former condition, papers usually report the number of fitness evaluations to that point. This measures how quickly the algorithm is able to converge on the global optimum, and is generally suited to applications where the goal is to find a good enough solution using minimal resources.

Where the condition is the number of fitness evaluations computed, the error at that time is usually reported. This is more applicable to situations where it is desirable to find the best solution within a given computing budget. By using evaluations as the time measure it becomes possible to make meaningful comparisons between runs with different population sizes.

When testing speciated algorithms on multimodal environments, it can be useful to report the number of significant optima found as a measure of population diversity. Again this is either reported as the number of evaluations required to locate all significant optima to within a given error or the number of optima located after a certain number of evaluations.

2.3.4 Measuring performance in dynamic environments

The metrics presented above are not generally useful for measuring performance on dynamic problems. Since the environment is constantly or periodically changing, it is insufficient to simply locate the global optimum towards the end of a run; the algorithm must be able to track the optimum throughout.

Various metrics have been proposed to measure algorithm performance in dynamic environments. Many report the fittest value at each generation; an algorithm's performance is analysed by graphing these values and comparing with other algorithms at each genera-

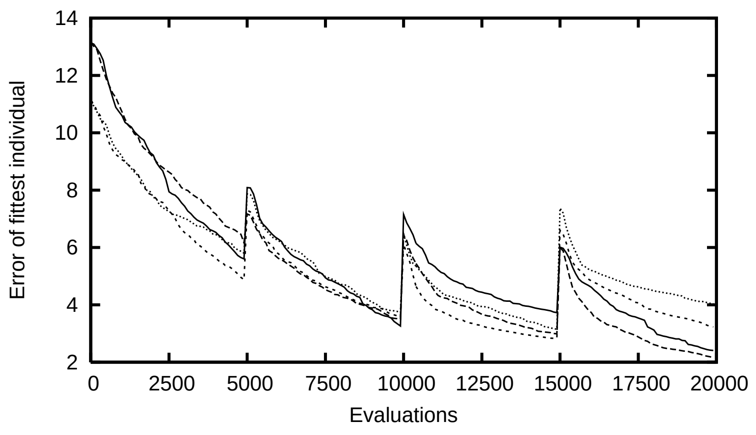


Figure 2.4: Error over time. The environment is changing every 5000 evaluations.

tion [Bäck, 1998; Grefenstette, 1999]. The resulting charts can be compared by looking at how quickly and to what level each algorithm's error falls after an environment change. An example of this type of chart is shown in Figure 2.4.

Depending on what the researcher is interested in these graphs can be quite informative – they allow the viewer to identify the algorithm that best suits their needs. Some algorithms recover very quickly but to a less fit solution, others take longer but more accurately locate the global optimum. Which is more appropriate depends on the task – if a "good" decision must be made as quickly as possible after a change an algorithm that recovers quickly is essential, otherwise it might be better to use one that will deliver a better solution if given more evaluations.

Online performance

Online performance [De Jong, 1975] averages the fitness of all the individuals at each generation. This is in some ways a convergence measure – an algorithm that is fast to converge on an optimum will be rated better by this metric than one that more accurately discovers the optimum but is not as converged. This penalises algorithms that keep individuals in suboptimal areas of the decision space to preserve diversity – the suboptimal individuals drag the average population fitness down.

Collective Mean Fitness

Collective Mean Fitness [Morrison, 2003] works by averaging the fitness of the best individual in each generation over many iterations. By measuring over a large number of generations, this metric provides a stable value with which performance can be compared. This metric can be difficult to interpret when the fitness of the optimum varies greatly during or between runs; this can act as noise, masking an algorithm's actual performance.

Equation 2.2 describes how collective mean fitness is calculated. F_T is the mean fitness, F_{BG} is the fitness of the best individual of each generation, G is the number of generations and M is the number of runs performed. F_T becomes constant as G tends to infinity.

$$F_T = \frac{\sum_{m=1}^{M} \left(\frac{\sum_{g=1}^{G} (F_{BG})}{G} \right)}{M} \tag{2.2}$$

Mean Tracking Error

Mean Tracking Error [Morrison, 2004] measures the average distance in the search space between the optimum and the best known solution at the end of each generation, as shown in Equation 2.3. This indicates how quickly the algorithm moves towards the correct peak. Algorithms which do not track the correct peak are penalised by this metric, even if they have converged on an area of near-optimal fitness.

$$E_t = \frac{\sum_{g=1}^{G} \left(\sum_{m=1}^{M} \sqrt{\sum_{i=1}^{N} \left(x_{i,BG} - x_{i,BP}\right)^2} / M \right)}{G} \tag{2.3}$$

The mean tracking error is represented by E_t, and N is the number of decision variables. The locations of the best individual and peak in dimension i are denoted by $x_{i,BG}$ and $x_{i,BP}$ respectively. M and G have the same meaning as in Equation 2.2.

Current Error

Current error is the fitness difference between the best solution and the global optimum [Branke and Schmeck, 2003]. Current error is computed as $\epsilon_t = opt(t) - F_{Bt}$, where ϵ_t, $opt(t)$ and F_{Bt} are current error, optimal fitness and the fitness of the best known-solution at evaluation t respectively. By monitoring current error we can analyse an algorithm's instantaneous performance, producing graphs such as the one shown in Figure 2.4.

Offline Error

Branke's modified offline error [Branke and Schmeck, 2003] provides an intuitive measure of an algorithm's performance, that is how well it is able to track the global optimum as it moves around the decision space. This incorporates population diversity, convergence speed and recovery after a landscape change. For functions where the global peak's location can drastically change, population diversity is also being measured; an algorithm that has maintained individuals near the new global optimum will take less time to locate it.

Offline error is calculated by averaging the current error over an entire run, as in Equation 2.4:

$$\epsilon^* = \frac{\sum_{t=1}^{T} \epsilon_t}{T} \tag{2.4}$$

The primary advantage of offline error is that it provides an overall indication of an algorithm's performance by combining the exploration, exploitation and recovery aspects into a single number. As it measures the average error per evaluation and not per timestep, meaningful comparisons can be made between algorithms using different population sizes. By using the fitness error instead of the distance error, this metric avoids overly penalising an algorithm for converging on a near-optimal location away from the global optimum.

While offline error is effective in comparing algorithms, it does not indicate how an algorithm can be improved. It is possible for two algorithms with wildly different characteristics to obtain the same offline error score. For example, algorithm **A** may converge very quickly but maintain poor diversity while algorithm **B** does the opposite. **A** locates its peak very quickly but often the peak it locates is not the global optimum. **B** takes a long time to accurately locate an optimum, but in most cases it will converge on the global optimum. Assuming both algorithms achieve similar offline error scores, which is better? Without further analysis of the individual runs, it is impossible to tell. By combining the exploration and exploitation aspects of an algorithm we discard information that could provide useful analysis. In Chapter 4 we show how by using offline error in conjunction with other metrics it is possible to determine the most effective way to improve an algorithm.

To calculate the error, the fitness of the global optimum must be known at all times. Most test functions can be generated so that the global optimum is easily determined, however it

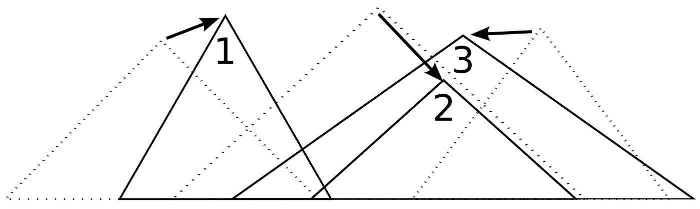


Figure 2.5: The difficulty of tracking peaks: Peak 2 used to be the global optimum however it is now covered by Peak 3. At some point in the future Peak 2 may re-emerge and become the highest again; algorithms with insufficient population diversity are unlikely to rediscover this peak.

is seldom available when optimising real world problems. This limits the use of offline error on these tasks.

2.3.5 The Moving Peaks test suite

To evaluate the algorithms we will propose in this thesis, we will use the Moving Peaks test suite [Branke, 1999]. Moving Peaks is a highly configurable fitness landscape generator, and is a commonly used benchmark for dynamic optimisation algorithms, for example in [Janson and Middendorf, 2004], [Ayvaz et al., 2006] and [Wang et al., 2007]. It consists of a number of peaks which move around the decision space, changing height and width as they do so. For simplicity, a conic peak shape is generally used, although this is by no means a requirement.

This problem is challenging because of the nature of the peak movement – since the peaks are changing in both height and location, a currently sub-optimal peak may become the global best at some point in the future, as in Figure 2.5. To achieve good performance it is necessary to both track as many peaks as possible and reconverge as quickly as possible after a peak movement.

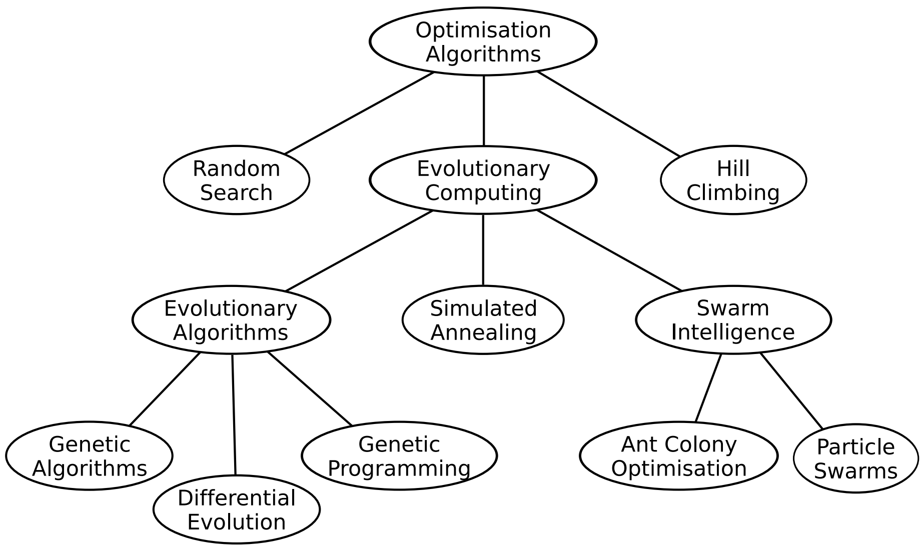


Figure 2.6: How the different algorithms are related

2.4 Optimisation Algorithms

Many optimisation algorithms have been developed to tackle the wide variety of problems found in the real world. Each algorithm has its advantages and disadvantages; certain algorithms are better suited to certain problems than others. Without running actual tests it is difficult to say which algorithm should be used for any given problem, although there has been research [Langdon and Poli, 2005] to determine the types of fitness landscape each algorithm is best suited to. This section describes random search, gradient decent and the commonly-used algorithms in the EC and SI families. Figure 2.6 shows the relationships between the algorithms.

2.4.1 Random Search

Stochastic optimisation methods incorporate a degree of randomness in the search process. If a stochastic algorithm fails to find a global optimum on a particular run it is possible to try again. Deterministic algorithms do not have this property – if they become stuck in a local optimum, something about the system must be changed in order to achieve a different result.

The simplest stochastic method is random search, where points within the problem space are chosen randomly and tested. Random search does not use the information gained from previous samples to guide its guesses, and thus does not perform as well as more sophisticated methods on problems where there are local similarities. Most of the "interesting" problems have local similarities, so most techniques have been designed to concentrate the search on areas with promising points.

2.4.2 Local Search

Local search methods are designed to rapidly locate an optimum once its general area has been found. These methods are susceptible to becoming trapped in a local optimum, meaning that they are most effectively used once the peak's location is already approximately known. Most local search methods do not use a population – at each timestep only one point is sampled.

The most intuitive local search method is hill climbing. This method works by continually sampling the decision space around the best point found so far [Michalewicz and Fogel, 2000]. At each iteration, a point somewhere near the current best is selected and evaluated. If the new point is better than the current best it replaces it, otherwise the new point is discarded. By repeating this process many times we "climb" the peak of the initial starting point, usually chosen randomly.

A variant of hill climbing is gradient ascent, which works by using the derivative of the evaluation function to guide the search direction [Gottfried, 1973]. The next point chosen to search is one that is close to the last point evaluated, but in the direction of the steepest ascent. Use of this algorithm is limited to evaluation functions where the gradient at each point can be computed.

2.4.3 Simulated Annealing

Simulated annealing [Kirkpatrick et al., 1983; Ingber, 1993] imitates the process of a material cooling. At the start the material is very hot and the molecules within are free to reposition themselves. Over time the temperature drops and the molecules lose energy; their movement eventually decreases to the point where they settle in their respective positions.

Simulated annealing mimics this to avoid becoming stuck in a local optimum. It maintains a single current solution and tries other candidate solutions in the current solution's neighbourhood. If the tried solution is better than the current one it becomes the new current solution. If the tried solution is equal or worse than the current one, it may still be accepted with a probability determined by T and how much better the current solution is. T in this case represents the temperature and is controlled by the cooling schedule, usually an exponential curve. When T is high, worse solutions are more likely to be chosen. As T tends to 0 the algorithm becomes more risk adverse, rarely moving to a worse solution.

If run over an infinite amount of time, simulated annealing is guaranteed to find a global optimum [Geman et al., 1993]. The difficulty is that we do not have an infinite amount of time in which to run the algorithm, and on many problems simulated annealing takes much longer to locate an optimum than the other techniques presented in this chapter.

2.4.4 Genetic Algorithms

Genetic Algorithms (GA) [Holland, 1992] act as a simplified version of Darwin's theory of evolution. A number of individuals are created, each having a random set of genes, or genotype. The genotype encodes the individual's solution to the problem in some manner, usually as a binary string. The evaluation function is used to rate each individual, and a selection of the best individuals is then bred together to produce the next generation. By repeating this process many times the system is able to hone in on the best solution.

At the end of each generation, the relative fitness of the individuals is compared and some are selected for breeding. The fitter an individual is, the higher chance it has of being selected. In GAs that use elitism, the fittest individual will always survive to the next generation unaltered.

Once the individuals are selected, pairs are randomly chosen and combined using a process called crossover [Fogel, 1994]. The purpose of crossover is to combine the genotypes of the two parents, hopefully propagating their beneficial genes. There are a number of different crossover methods, such as one-point, two-point and uniform. In one-point crossover, a cut point is randomly chosen along the length of the genotype. To produce the offspring, the mother's and father's genes are swapped at that point. For example if the mother's genotype was 00000 and the father's 11111, cutting between the 3rd and 4th bits would result in the offspring 00011 and 11100.

The offspring created during crossover become the new generation. To introduce diversity into the population, at each timestep there is a small probability the offspring undergo mutation, where part of the genotype is randomly modified. In the case of binary strings, usually a bit is flipped. The probability that a particular individual will undergo mutation is called the mutation rate. If the mutation rate is high the algorithm will often take a long time to converge, however it is less likely to become trapped in a local optimum.

2.4.5 Differential Evolution

Differential Evolution (DE) is also a population-based optimisation algorithm. Rather than being used with binary strings, genotypes in a DE are generally points within an $\Re^n$ space, where n is the number of decision variables [Storn and Price, 1997; Corne et al., 1999].

At the end of each generation, the DE chooses 3 distinct individuals and adds the difference in position between two of them to the position of the third. A new individual is then generated using a combination of the calculated position and some parent individual; for each dimension the algorithm will decide randomly whether to use the coordinate from the parent or the calculated position. Once the new individual has been created it will be compared with its parent; if it is fitter it will replace the parent in the next generation's population, otherwise it will be discarded and the parent will survive.

2.4.6 Evolutionary Strategies

Evolutionary Strategies (ES) [Bäck, 1996] share similarities with both GA and DE. Individuals are represented as a vector of real numbers as with DE, but like GA are created using combination and mutation.

In addition to encoding their respective candidate solution, individuals in an ES also have *strategy parameters* that control their mutation. These parameters are also combined and mutated in the same way as the individual's position within the decision space. By doing this, the ES is able to adapt to the problem at hand – the individuals with parameters that are better suited to the problem will generally have better fitness, and therefore be able to pass these parameters onto the next generation. This is a form of parameter control, which will be discussed in Section 2.4.11.

2.4.7 Genetic Programming

Many tasks are better tackled by creating a program than evolving a static solution for each new set of input variables. Genetic Programming (GP) [Koza, 1992] allows us to evolve programs that generate the desired output for any given input. To measure the fitness of a generated program, it is given several input sets and its output is compared against the expected result. The closer the output is to what was expected, the fitter that program is.

GP works similarly to GA in that there is a population of candidate solutions to which selection, crossover and mutation are repeatedly applied. The solutions themselves are represented differently though – in GP each solution is a program tree consisting of operators and terminals. Operators occupy the branch levels of the tree and require at least one child operator or terminal as an argument. Common operators include $+$, $-$, $\times$ and $\div$. Terminals occupy the leaf level of the tree and do not accept any arguments. Terminals are either constant numbers, callable functions or input variables.

Crossover works by swapping randomly selected branches between individuals, mutation by replacing a node with a random subtree. The tree structure of GP allows arbitrary changes like this without the need to check that the generated children are valid – when implemented correctly it is impossible to generate programs such as "5 + +" by either crossover or mutation. This property is known as closure. To ensure this property is maintained, conditions such as division by 0 and $\sqrt{-1}$ must be handled, usually by replacing the relevant operators with safe ones that return a specified value if the operation cannot be performed.

2.4.8 Ant Colony Optimisation

Ant Colony Optimisation (ACO) and Particle Swarm Optimisation (PSO, described below) are both members of the Swarm Intelligence paradigm. Whereas standard EAs have a central controller which decides who should live, die and breed, in SI each agent interacts with its neighbours to decide where to move next. By giving the population some simple interaction rules to follow, quite complex behaviours can emerge as the individuals interact with each other.

ACO mimics a colony of ants in search of food to solve combinatorial problems; that is where the goal is to find the most efficient path through a graph [Dorigo and Caro, 1999]. At any point on the pathway between the nest and the resource, the ant must choose which way to go without the aid of an external map or knowledge of the terrain. To do this it lays a pheromone trail as it searches; this trail allows it to find its way back to the nest. Once the ant has found food it walks back, laying another trail with a differnet pheromone. When a subsequent ant discovers the return trail it may or may not follow it, depending on the strength of the pheromone. Over time, more and more ants will follow the return trail when looking for food; as they return to the nest they will add their scent to the trail, further reinforcing it.

Since a shorter path allows more ants to complete traversal in a given amount of time, the most efficient path will have the strongest scent – it is the one that has been walked over the most. The pheromone evaporates over time, allowing the ants to "forget" inefficient paths or paths to food sources that have since been exhausted.

ACO sends out a number of ants to explore the paths around a graph, looking for the best solution. Each ant keeps track of where it has been and may have access to heuristics that help it find the goal. Once it has found the goal, it marks the path it took with pheromone with a strength proportionate to the cost of its solution – low cost solutions are marked with more pheromone than high cost ones. Over time a single path with strong pheromone will develop, with most of the ants following this path.

2.4.9 Particle Swarm Optimisation

Particle Swarm Optimisation [Kennedy and Eberhart, 1995; 2001] mimics a flock of birds searching for food or water. As with ACO, in a PSO individuals do not breed. Instead they move around the search space looking for promising areas. Each individual i at time t has a location $\vec{x}_{i,t}$, velocity $\vec{v}_{i,t}$, and memory of the location of the best point $\vec{p}_{i,t}$ it has seen so far. It also has a number of neighbours with whom it can communicate, the fittest of which is denoted $\vec{p}_{g,t}$.

To choose its next location, it adds its velocity to the current location, as in Equation (2.5):

$$\vec{x}_{i,t+1} = \vec{x}_{i,t} + \vec{v}_{i,t+1} \tag{2.5}$$

Equation (2.6) shows the original velocity update equation used by Kennedy. $\vec{r}_1$ and $\vec{r}_2$ represent vectors whose components are random numbers in the range of $[0, 1]$. $\otimes$ represents pointwise vector multiplication, where each element in the first vector is multiplied by the corresponding element in the second vector.

$$\vec{v}_{i,t+1} = \vec{v}_{i,t} + \varphi_1 \vec{r}_1 \otimes (\vec{p}_{i,t} - \vec{x}_{i,t}) + \varphi_2 \vec{r}_2 \otimes (\vec{p}_{g,t} - \vec{x}_{i,t}) \tag{2.6}$$

The right hand side can be seen as three parts added together. The first part models the inertia each particle feels – as in the real world, things cannot instantly change direction. The middle and last sections represent the cognitive and social components respectively. The particles are attracted towards two points in the search space: the best point they have found and the best point found by a neighbour. By subtracting the current position from each of these points, we get the direction and distance to that point. Rather than being attracted directly to the points however, a random vector $\vec{r}$ is applied so that the particle explores nearby regions. In the original PSO model [Kennedy and Eberhart, 1995], φ_1 and φ_2 were both set at 2 so the particle often overshoots the attractor points and explores on the other side.

Constraining Velocity

One difficulty with the original model is that it is common for the particles to violently oscillate around an optimum. The first attempt to solve this problem was simply to limit each component of $\vec{v}$ to a specified value V_{max}. While limiting global exploration, this prevented the velocity from reaching ridiculous levels. Particles still had difficulty converging though, and two main methods have since been developed to counteract this: inertia weight and constriction.

The inertia weight PSO model [Shi and Eberhart, 1998a] modifies Equation (2.6) by adding an additional term ω, as shown in Equation (2.7):

$$\vec{v}_{i,t+1} = \omega\vec{v}_{i,t} + \varphi_1\vec{r}_1 \otimes (\vec{p}_{i,t} - \vec{x}_{i,t}) + \varphi_2\vec{r}_2 \otimes (\vec{p}_{g,t} - \vec{x}_{i,t}) \tag{2.7}$$

The balance between exploitation and exploration can be controlled by varying ω. Large values increase the momentum each particle has, causing it to explore over a wider area. Small values make it easier for the particles to change direction, allowing them to converge on an already-known optimum more quickly. In their initial work, Shi and Eberhart set ω to between 0.8 and 1.4, finding that values close to 1 were the most reliable[1]. They also found that when the runs that did not locate the optimum are ignored, using a smaller ω setting greatly improved convergence speed.

These characteristics lead to the recommendation of adopting an inertia weight of 0.9

[1]An ω value of 1 is equivalent to the original PSO model.

at the start of the run and linearly reducing it over time to 0.4 [Shi and Eberhart, 1998b]. This causes the system to be exploratory at the start of a run and exploitative at the end, reducing the risk of premature convergence while still allowing the optimum to be quickly located – the same approach used by simulated annealing.

The constriction PSO works similarly to inertia weight in that a term is added to dampen the particle's velocity [Clerc, 1999], to the point that the inertia weight PSO can be transformed into the constriction model as shown by Clerc. The difference between the two models is that constriction is applied over both the previous velocity and the attraction force as in Equation (2.8):

$$\vec{v}_{i,t+1} = \chi \left(\vec{v}_{i,t} + \varphi_1 \vec{r}_1 \otimes (\vec{p}_{i,t} - \vec{x}_{i,t}) + \varphi_2 \vec{r}_2 \otimes (\vec{p}_{g,t} - \vec{x}_{i,t}) \right) \tag{2.8}$$

The constriction factor is represented by χ and its value depends on $\varphi = \varphi_1 + \varphi_2$ and κ, usually set at 1. It is calculated using Equation (2.9):

$$\chi = \frac{2\kappa}{\left| 2 - \varphi - \sqrt{\varphi^2 - 4\varphi} \right|} \tag{2.9}$$

For the constriction model φ_1 and φ_2 are usually set at 2.05, giving $\varphi = 4.10$ and $\chi \approx 0.7298$. For χ to be defined, φ must be greater than 4. The velocity is prevented from collapsing to 0 by the changing sizes of $\vec{r}_1$ and $\vec{r}_2$; at each timestep the particle is pulled towards a different point. Clerc and Kennedy [2002] also proved that this model guarantees that the swarm will eventually converge. This is an extremely important property as it enables the algorithm to accurately locate an optimum.

It has since been found that Clerc's constriction model provides the best performance when the velocity in each dimension is limited to the decision space width in that dimension. This is slightly different to V_{max} in that it does not assume that the decision space is the same size in all dimensions. When combined with the velocity limit, the constriction model generally outperforms the inertia weight model [Eberhart and Shi, 2000].

Controlling Information Spread

So far, it has been assumed that every particle can communicate with every other particle, that is the best point discovered by any particle is known by the entire population. This causes the entire population to rapidly converge on the best known point, which may or may not be desirable. Where the fitness landscape is largely unimodal this is an advantage – the concentration of particles allows the PSO to accurately locate the optimum very quickly. Multimodal landscapes present more of a challenge though – if the algorithm concentrates only on the currently known best and does not explore the search space enough, it may become trapped in a local peak and fail to locate the true optimum.

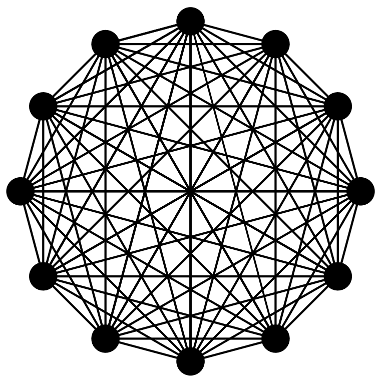
By reducing the number of connections between particles, we can slow the information spread. As an analogy using human behaviour, say we have 3 people: **A**, **B** and **C**. **A** and **C** do not know each other, but are both friends with **B**. **A** has just witnessed a new celebrity scandal and tells **B** about it, who then tells **C**. Since there is no direct link between **A** and **C**, **C** must wait until **B** spreads the word. As **A**, **B** and **C** tell other friends, who in turn tell their friends, the information spreads throughout the entire population. If every member of the population could talk to every other member, the information will spread much faster.

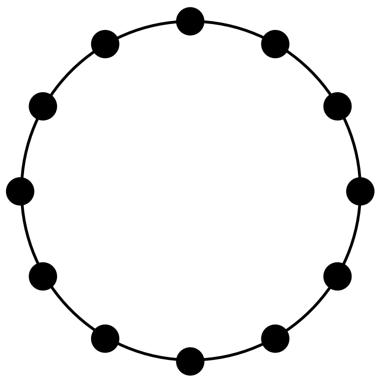
An example of this is the news – a broadcaster is able to communicate with every person that watches their news programme. People who watch or read the news are generally far more up to date than those who do not, simply because they are better connected with the news source.

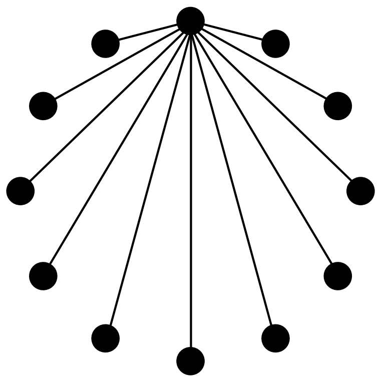
In the case of PSO the information is the location of a fit point; a particle will not be attracted there unless either it or one of its neighbours has discovered it. As successive particles are attracted to and discover an optimum, the information about it spreads through the population.

The population is able to explore multiple optima simultaneously when the number of neighbours each particle has is low compared to the total number of particles. As a particle is always attracted to the fittest point known by any of its neighbours, eventually the whole system will converge on the best-known optimum. By slowing the information spread, we cause the population to explore the decision space more thoroughly, increasing our chances of finding the global optimum [Kennedy and Eberhart, 2001].

Several neighbourhood topologies have been explored; the common ones are shown in Figure 2.7. It has been shown [Kennedy and Mendes, 2002] that circle and wheel do not perform well compared to other better connected topologies. These topologies do not allow much collaboration, meaning that each particle is searching in relative silence – it cannot quickly recruit other particles to help it explore a promising area. Kennedy and Mendes concluded that of all the neighbourhood topologies they tested, von Neumann provided the best overall performance: it was connected enough that the PSO could search locally promising areas effectively, but not so connected as to cause the population to converge on


(a) Global: Every particle is connected with every other.


(b) Circle: Each particle is connected with n neighbours in a ring structure. $n = 2$ is shown.


(c) Wheel: One central particle is connected with every other.

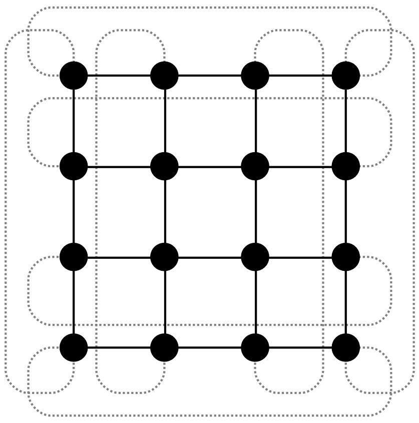

(d) Von Neumann: Particles are connected in a 2 dimensional lattice.

Figure 2.7: Common neighbourhood topologies.

the first optimum it found.

This concurs with earlier research by Hillis [1991], where GA individuals were placed in a von Neumann topology and only allowed to mate with others that are nearby. This allowed multiple solutions to coexist in the population, something that would not normally occur with unrestricted mating.

2.4.10 Using Meta-models

In some applications, fitness evaluations are exceedingly expensive. They may need human interaction, take many hours to compute, or require external facilities to be used, potentially costing real money. To reduce the number of fitness evaluations required, a meta-model can be used [Jin, 2005]. A meta-model creates an approximate model of the evaluation function. The real evaluation function is only used sparingly – for most evaluations the meta-model is used instead. The reduced cost of evaluating the meta-model allows the optimisation to proceed far more quickly.

In cases where a meta-model cannot be created beforehand, it can be dynamically generated from the evaluation function. One approach is by using clustering, as in [Kim and Cho, 2001]. In this case the population is divided into clusters and a single representative evaluation is performed for each cluster. Fitnesses are assigned to the individuals based on their distance from the representative sample. The number of fitness evaluations per timestep is controlled by the number of clusters created, allowing the evaluations budget to be specified independently of the population size.

A meta-model can also be created by fitting a surface to the points that have already been evaluated by the evaluation function. This model is then used to evaluate new points instead of the real evaluation function. For example Regis and Shoemaker [2004] use a model of the local landscape features to decide which solutions to evaluate with the real function. For each individual, the k nearest points that have already been evaluated are used to build a local landscape model. The individuals are evaluated according to their respective models, and only the ones with a high expected fitness are retained – the rest are discarded. This creates a filter, reducing the cost of the optimisation by preventing the algorithm from evaluating points that are expected to have a low fitness. The optimisation algorithm then proceeds as normal, using only the filtered points.

One particularly interesting EA combining fitness approximation and local search is EANA (Evolutionary Algorithms with N-dimensional Approximation) [Liang et al., 2000], where polynomial regression was used to approximate the fitness landscape. The experiments of EANA on a wide range of test functions have demonstrated the effectiveness of this approach. Nevertheless, EANA was designed to locate a single global optimum (not multiple global optima), and the test functions were all static test functions. To the best of our knowledge, no EAs using fitness approximation and local search have been tested for locating multiple global optima. It is even more difficult when tracking multiple moving peaks in a dynamic environment. In Chapter 6 we propose a PSO incorporating fitness approximation and local search methods, designed specifically for environments with multiple global optima.

2.4.11 Parameter Control

All of the optimisation algorithms discussed in the previous sections have user-specified parameters. These parameters must be specified before the algorithm is run and can greatly affect its success and performance. In many cases it takes much tuning to find the optimal parameter values for a problem. In addition, the parameter settings that are most effective at the start of an optimisation are likely to be different to the ones at the end, where the population has already converged and is fine-tuning its solution.

There has been much research into this problem, and the many proposed solutions are collectively called parameter control [Eiben et al., 1999]. As the different algorithms function in different ways, most parameter control methods are specific to a particular optimisation algorithm. For example, ES (see Section 2.4.6) uses parameter control by adaptively modifying the mutation rate, however this cannot be applied to algorithms such as PSO and DE because they do not use mutation.

An example of parameter control as applied to PSO is [Shi and Eberhart, 2001]. Fuzzy logic is used to dynamically adjust the inertia weight of the particles, based on how close their fitness is to the estimated optimal fitness. As discussed in Section 2.4.9, the inertia weight controls the balance between exploration and exploitation.

By using fuzzy logic the particle swarm is able to move to exploration mode as soon as it has approximated the optimum, rather than having to wait for the inertia weight to fall over time as with the standard PSO. It also means that the algorithm will not force the particles into exploitation mode before the area of the optimum has been determined. The authors showed a significant performance improvement in their tests, demonstrating the advantage of using parameter control over statically set values.

2.5 Dynamic Optimisation

Dynamic environments present additional challenges for all optimisation algorithms. Because the problem is changing over time, the algorithm cannot assume that the peaks it discovers will remain in the same place forever. Optima may move or disappear entirely, only to be replaced by new peaks in other areas. There are two parts to handling change: detecting it and dealing with it.

2.5.1 Detecting change

When optimising in a dynamic environment, if the algorithm maintains a memory it is important for it to be able to detect changes. Changes in the environment have the potential to invalidate any assumptions the algorithm may have made, including the fitness of a particular solution. To detect these changes, the algorithm must monitor the environment.

The most common method is by the use of sentries [Carlisle and Dozler, 2002]. At each timestep, the algorithm reevaluates the fitness of one or more points in the decision space. If the point's fitness becomes different to the value stored in memory the algorithm knows the environment has changed. The point or points can either be at a fixed or random location as in [Carlisle and Dozier, 2000], or the current population members as used by Li and Dam [2003].

2.5.2 Dealing with change

One of the simplest strategies for dealing with environment changes is simply to restart the algorithm, as in [Raman and Talbot, 1993] for example. This means that the dynamic environment is treated as series of unrelated static problems by the algorithm. This technique is appropriate in situations where the changes are so severe that the new environment is either not at all or only minimally similar to the old.

For many problems the old and new environments are substantially the same, or at least similar enough that the algorithm's existing knowledge is still useful. In these cases performing a restart is quite wasteful as the existing knowledge is discarded. If the algorithm is not restarted however, to avoid making decisions based on stale data we must ensure that any memory it has of the previous environment is cleared. For a PSO, this means that we either reevaluate the fitness of each particle's personal best, or set the personal best to the particle's current location.

When there is a change in the environment, we want the algorithm to start looking for new solutions again. There are several approaches to this, which Jin and Branke [2005] have grouped into 4 categories, described below.

Reintroducing diversity

Whenever a change is detected, the algorithm responds by increasing the diversity of the population. In EAs this is often achieved using hypermutation [Cobb, 1990], where the mutation rate is temporarity increased to reintroduce diversity into the population. PSOs may do this by re-randomising a proportion of the population [Eberhart and Shi, 2001; Hu and Eberhart, 2002].

In both cases, reintroduction of diversity results in the replacement of old knowledge with random points. How advantageous this is depends on how similar the old and new fitness landscapes are. If the landscape has only changed slightly, adding too many random points will most likely hinder the search. If the changes were large, adding too few random points may result in the algorithm not finding a new global peak. This conundrum relates back to No Free Lunch, discussed in Section 2.1.

Maintaining diversity

Another approach is to continually maintain population diversity. An algorithm that prevents its population from fully converging is in a better position to react to changes than one that does not. One of the most straightforward ways is to continually re-randomise a small portion of the population, as was done in [Grefenstette, 1992]. This ensures the algorithm never stops looking for new optima.

Remembering past solutions

In some dynamic problems the environment may revert to a state similar to a previous state. For example in a job scheduling problem, a broken down machine may have been repaired, allowing a previously developed solution to be reused. In these cases it can be useful to store an archive of previously-fit solutions, for example [Louis and Johnson, 1997] and [Mori et al., 2001].

This strategy has a number of difficulties, including deciding which solutions to store and when [Branke, 1999]. Reseeding the population with previous solutions may also result in premature convergence to a local optimum. This can happen when the stored individuals are better than any of the current population, but only cover suboptimal parts of the decision space. This problem can be reduced by also increasing population diversity in some other way.

Multiple populations

The fourth group of strategies involves maintaining multiple populations in different areas, the major focus of this thesis. By having multiple subpopulations or species, the algorithm is able to maintain population diversity while exploring multiple areas of the decision space simultaneously. Several approaches to this are described in Section 2.7.

2.6 Multiobjective Optimisation

In real world optimisation we often need to achieve multiple goals simultaneously. For example we might want to design a truck that is both fuel efficient and powerful. Fuel-efficiency and power represents a trade-off – a more powerful engine is generally heavier and uses more fuel in normal operation. In these cases there is no one perfect solution: is a powerful but fuel-inefficient truck better or worse than a vehicle that uses little petrol but can only carry small loads?

Multiobjective optimisation (MO) allows us to optimise both goals simultaneously. It presents a range of solutions representing different points in the tradeoff. The user can then choose the most powerful design that has a given efficiency or the most efficient design that meets their power requirement.

To do this, the algorithm evaluates each individual based on the *non-dominated* front it belongs to [Deb et al., 2002]. A non-dominated front is a set of solutions which are neither better nor worse than each other. Solution **A** dominates solution **B** if two conditions are met:

- **A** is fitter than **B** in at least one objective
- There are no objectives for which **B** is fitter than **A**

The pareto front is the set of solutions that are not dominated by any other solutions in the decision space. These represent the best tradeoff available – the truck cannot be made more efficient without sacrificing power or given more power without making it use more fuel.

2.7 Multimodal Optimisation

Many real-world fitness landscapes consist of many optima, either local or global. These additional optima make it more difficult to locate an acceptable solution and increase the likelihood of premature convergence. When used with only a single objective, evolutionary algorithms have a tendency to converge on a single point. By preventing the entire population converging in one area, an algorithm is less prone to becoming stuck in a local optimum. In this section we outline a number of representative techniques that improve performance in these environments and allow an algorithm to locate multiple optima.

2.7.1 Crowding

The crowding method [De Jong, 1975] was developed to reduce the risk of premature convergence in GAs. A percentage G of the population, known as the generation gap, is selected for breeding. For each of the new individuals created, a number of the existing individuals are chosen to compare against. This number is called the crowding factor and written CF. The member of the chosen individuals that is most similar to the new individual is discarded and the newly-generated individual is put in its place.

This technique ensures that areas represented at the start of a run will generally still be represented towards the end. It was later found that crowding has difficulty maintaining individuals on multiple optima [Deb and Goldberg, 1989]. This occurs when none of the chosen CF individuals occupy the same peak as the new individual. This causes replacement error, where the new individual replaces an individual on some other peak, losing diversity.

Deterministic crowding [Mahfoud, 1992] solves this problem by only comparing an individual to its parents. An offspring is highly likely to be similar to its parents, thus replacing a parent with its offspring causes only very minimal replacement error. As with the original crowding method, the parent individual is only replaced if the child has a higher fitness, otherwise the child is discarded.

One drawback of this approach is reduced selection pressure. Since each individual only competes with its parents, it is possible for groups of individuals with low fitness to persist indefinitely. These individuals contribute very little to the search, and the wasted evaluations increase the time needed for the algorithm to locate the desired optima.

2.7.2 Fitness Sharing

Fitness Sharing [Goldberg and Richardson, 1987; Sareni and Krahenbuhl, 1998] encourages diversity by reducing the apparent fitness f_i' of an individual i according to how many other individuals are nearby. This is done using Equation (2.10):

$$f_i' = \frac{f_i}{m_i} \tag{2.10}$$

The niche count m_i is a measure of how many individuals are in i's area and how close they are. It is computed by summing a sharing function $sh(d_{ij})$ for each individual j of the population N, as in Equation (2.11). The decision space distance between individuals i and j is denoted as d_{ij}, although it is not necessarily the Euclidean distance. In binary GAs, the Hamming distance is most often used.

$$m_i = \sum_{j=1}^{N} sh(d_{ij}) \tag{2.11}$$

The sharing function shown by Equation (2.12) returns a value in the range $[0, 1]$ depending on how close the individuals are. Having two individuals on the same point is represented by a value of 1, and 0 is returned if they are further than the user-specified parameter σ_s. Intermediate values are returned for distances less than σ_s. The value of α is usually set at 1; using other values allows the user to increase or decrease the sharing penalty for very close individuals.

$$sh(d_{ij}) = \begin{cases} 1 - (d_{ij}/\sigma_s)^{\alpha}, & \text{if } d_{ij} < \sigma_s \\ 0, & \text{otherwise} \end{cases} \tag{2.12}$$

Fitness sharing has a disadvantage shared by many speciated algorithms – the user is required to set a "similarity measure" parameter, in this case σ_s. Setting this to an appropriate value requires a priori knowledge of the function being optimised, specifically the expected distance between optima. For many functions this information cannot be obtained directly – the user has to run the algorithm multiple times and use trial and error to find the correct parameter value. This can be a computationally expensive process.

2.7.3 Clearing

Clearing [Pétrowski, 1996] is an elitist specialisation of fitness sharing. Instead of nearby individuals sharing the available fitness resource, clearing uses a winner-takes-all approach. The individuals are first sorted from fittest to least fit. For each individual in the list we scan all of the subsequent members; any that are within σ of the current individual are demoted, that is their fitness is set to the minimum possible value. These individuals are then removed from the sorted list to prevent them from demoting others.

The parameter σ is equivalent to σ_s as used by sharing. The effect of this algorithm is that each optimum is only represented by the fittest individual; all others have their fitness reset to the minimum value. This is a form of elitism which always preserves the fittest individual. Elitist methods are more susceptible to premature convergence as they concentrate on the best known solutions. If these are in local optima then the algorithm may become stuck, unable to locate the global optimum.

2.7.4 The Island Model

Instead of having one large population, the island model uses several smaller populations [Tanese, 1987; Bessaou et al., 2000]. These populations develop independently, potentially concentrating on different areas of the decision space. Periodically, individuals from one island will be shared with another. This increases the diversity in the target island and encourages the development of positive traits.

Where multiple computers are available to run the optimisation, distributed genetic algorithms are often used to decrease the time needed by the algorithm. Many distributed GAs use the island model, where each computer manages a separate population [Cantú-Paz, 1998]. As with the standard island model, individuals are periodically transferred between computers to increase population diversity.

Many of the ideas presented below act similarly to the island model, but rather than specifically creating a set of separate populations they group the individuals adaptively. These groupings are made based on the behaviour of the system, most often by grouping the population into sets of individuals who are near each other in the decision space.

2.7.5 Restricted Tournament Selection

Restricted Tournament Selection (RTS) modifies the standard GA model to encourage competition between nearby individuals [Parmee et al., 1994]. In RTS, a newly created individual **A** does not automatically become part of the next generation. Instead, a number N of individuals from the current population are randomly sampled and the one whose genotype is most similar to **A**'s is chosen. Let us call this individual **B**. The fitter of **A** and **B** becomes part of the next generation. The weaker individual is discarded.

One issue with RTS is that the value of N determines how many solutions will be returned. In many cases the user does not know how many optima there are, so setting an appropriate value can be difficult. Adaptive Restricted Tournament Selection (ARTS) overcomes this problem by removing the parameter [Roy and Parmee, 1996]. Instead the population is divided into clusters, and the new individual is compared against the closest member of the closest cluster. The clustering algorithm determines the number of clusters, and therefore optima found, freeing the user from this task.

2.7.6 Multinational GAs

The Multinational GA [Ursem, 1999; 2000] divides the population into a set of nations, each one representing its own optimum. Each nation consists of a population, a government and a policy. The government consists of the fittest individuals within the population, the policy represents the centroid of the government individuals' locations.

A hill-valley function is used to determine which nation each individual should belong to. The hill-valley function samples several points between the individual and a given nation's policy. An individual is considered part of a nation if all of the sampled points have a higher fitness than the individual or policy – that is there are no valleys between the two points.

At the end of every generation, each individual is checked to see whether it should remain in its current nation. If there is a valley between the individual and its nation's policy, the algorithm checks whether the individual should migrate to each other nation in turn. The individual will migrate if no valleys are detected between the individual and the prospective nation's policy. If no suitable nation can be found, the individual founds its own nation.

At the end of the migration phase, nations with low populations are replenished using poor-fitness individuals from overpopulated nations. The policies of the nations themselves are now compared with the hill-valley function. If no valleys are detected between two policies, it is assumed that the nations are converging on the same peak and are merged.

Ursem showed that while this technique is effective on both static and dynamic test functions, it is not always successful at maintaining population diversity. As discussed in Section 2.3.2, maintaining sufficient diversity is very important when optimising complex dynamic problems.

2.7.7 Sequential Niching

Sequential Niching[2] [Beasley et al., 1993] performs multiple optimisation runs to locate multiple optima. To discourage a subsequent run from converging on an already-known peak, the areas around previous solutions are derated in a similar manner to fitness sharing. As with fitness sharing, the niche radius r must be specified. If little is known about the problem it can be very difficult to find a suitable value for this parameter. The algorithm is rerun until the desired number of optima have been discovered.

In practise, Sequential Niching presents several difficulties in addition to finding appropriate parameter settings [Mahfoud, 1995b]. Since the actual size and shape of an optimum's catchment area is unknown, it cannot be cleanly "removed" from the landscape without

[2]The words niche and species are synonymous and may be used interchangeably.

leaving ridges and troughs. These artifacts can create gradients that lead away from the undiscovered optima, reducing the chance of the optima being discovered. If the peaks' catchment areas are different sizes, the algorithm is more likely to converge on the peak with the largest catchment area. This can result in many runs locating the same solution, despite it having been derated.

2.7.8 Clustering

Speciation can also be performed by using a k-means clustering algorithm to divide the population [Kennedy, 2000]. At the start of each generation, the following process is used to identify the clusters:

1. A number of individuals equal to the desired number of clusters C are selected.

2. The initial cluster centres are set to these individuals' locations.

3. Assign each member of the population to its closest cluster centre.

4. Compute each cluster's new centre as the centroid of its members' locations.

5. Repeat steps 3 and 4 three times to allow the clusters to stabilise.

The particles move according to the normal PSO model, except that each particle's personal best location is replaced by its cluster's centre. Kennedy also tried setting the global best position to the centre of the fittest particle's cluster but found this to reduce performance.

A similar method was presented by Passaro and Starita [2006]. Rather than modifying the particle's personal best, the clusters are used to restrict which other particles each individual can communicate with. This allows each cluster to concentrate on its own local area without disturbance from remote particles.

The main difficulty with both methods is setting the optimal value for the parameter C – the ideal value depends on the number of optima we expect to find. Setting it too low means we will not detect all of the optima; setting it too high increases the computational cost and can limit the interaction between particles, resulting in reduced performance.

2.7.9 NichePSO

NichePSO [Brits et al., 2002; 2003] differs from a normal PSO in that the unniched particles do not interact with each other, performing individual searches instead. If the fitness variance of a particle over the last 3 steps was less than a predefined amount δ, a subswarm is created consisting of that particle and its nearest topological neighbour.

Each subswarm "owns" an area of the decision space, defined as a hypersphere centred on the fittest particle. The hypersphere's radius is equal to the distance between the fittest particle and the furthest member of the subswarm. Any other particles that move to a point within the hypersphere permanently join the subswarm. If the hyperspheres of neighbouring subswarms overlap, it is assumed they are converging on the same optimum and are merged.

Initially the particles have very few neighbours. Before any subswarms have been created, none of the particles have any neighbours at all. This means that each particle's neighbourhood best is the same as its personal best. The particle now only has one attractor instead of two and will rapidly converge on it. If the particle's velocity drops to 0 it is no longer searching – it is simply repeatedly sampling its personal best.

To overcome this problem, NichePSO uses the Guaranteed Convergence PSO (GCPSO) [van den Bergh and Engelbrecht, 2002] to ensure the particles keep searching. GCPSO achieves this by applying a different movement rule to the fittest particle in the swarm. Instead of moving around normally as the remaining particles do, the fittest particle performs a local search by randomly trying points within a certain distance d of its personal best location.

The value of d is determined adaptively. As the particle tries its surrounding points, the algorithm remembers the number of times in a row it was successful or not in finding a fitter location. If the number of successes in a row exceeds a threshold, the search becomes more aggressive by doubling d. Likewise if the number of failures reaches a threshold it halves d, thereby searching in a smaller area.

GCPSO was modified by Peer et al. [2003] to make it better suited to complex neighbourhood topologies and speciated PSOs. Rather than having a single value of d for the entire swarm, each particle maintains its own value which is adjusted based on the particle's own successes or failures. All particles that are fitter than their neighbours use the GCPSO movement rule. In the case of NichePSO, this means the fittest particle of each species.

To improve NichePSO's performance, Engelbrecht and van Loggerenberg [2007] investigated limiting the situations under which a particle or subswarm would join another niche. It was found that by using scatter merging, whereby the weaker subswarm's particles are reinitialised, the algorithm was better able to explore the search space, increasing the number of optima it was able to locate. Exploration was improved further by only allowing tightly-clustered subswarms to absorb new particles.

NichePSO still has unresolved issues. Of the questions posed in the original paper's conclusion, only scalability up to 4 dimensions has been answered. In particular, it is still unknown how robust the value of δ is, and whether the algorithm is effective in dynamic environments.

2.7.10 SPSO and SDE

Speciated PSO (SPSO) [Li, 2004] and Speciated DE (SDE) [Li, 2005] are adaptions of clearing to PSO and DE respectively. Instead of modifying an individual's fitness, the population is divided into explicit species. Only individuals in the same species can communicate; each species performs an independent search in its area of the decision space. In this section we will only discuss SPSO, however most of this description applies equally to SDE.

The allocation of each individual works similarly to clearing. All individuals that are not within r (equivalent to σ in clearing) of a fitter population member are defined as species seeds. The rest of the population are then allocated to the fittest seed within r of their position. Each species is defined as the seed and the other individuals allocated to it, as in Figure 2.8.

SPSO has proved effective on a variety of multimodal optimisation functions and has used as a base for two of the algorithms presented in this thesis. It is able to maintain stable populations on multiple optima, provided they are further than r apart.

As with fitness sharing and clearing, the main difficulty with using this algorithm is setting the parameter r. If it is too large the algorithm cannot distinguish nearby optima. If it is set too small many species will form on local optima; often containing only the species

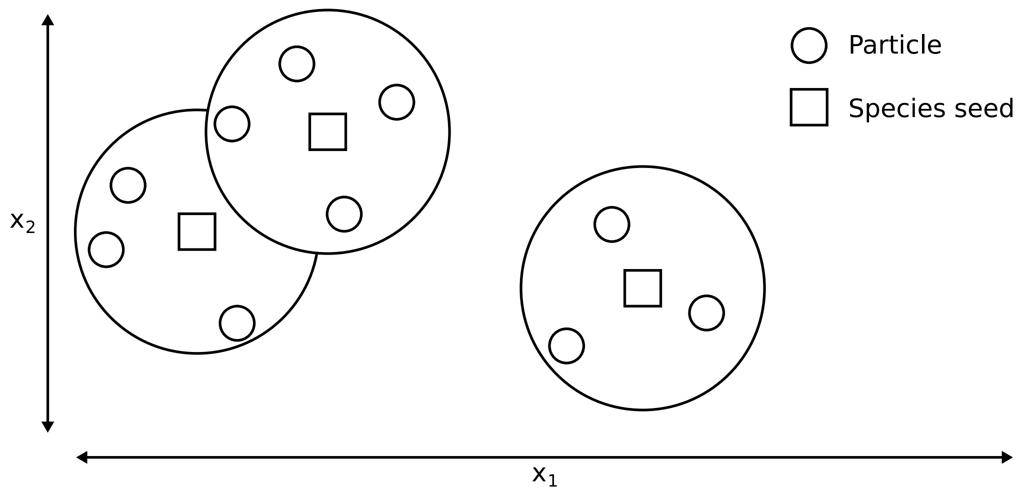


Figure 2.8: The species seeds represent the fittest individuals in their areas of the decision space. The middle seed is fitter than the one on the left, so its species area takes precedence where they overlap.

seed. An individual with no neighbours generally adds very little to the search; as discussed above it will simply converge on its personal best – even if this does not represent a local optimum. SPSO can be combined with GCPSO to allow these single-member species to find their local optimum, but escape to a higher peak is generally very difficult without inter-particle communication.

SPSO is quite sensitive to r, as are most methods that use decision space similarity measures. The next chapter discusses how this sensitivity can be reduced or removed, allowing multimodal functions to be effectively optimised without requiring the algorithm to be fine-tuned.

2.7.11 FER-PSO

FER-PSO is designed to create separate populations on multiple optima, but does not require the user to set how large each species should be [Li, 2007]. It is based on FDR-PSO [Veeramachaneni et al., 2003], which adds an additional term to the PSO velocity update equation (see Equation (2.8)) to produce Equation (2.13). The third term adds an additional attractor point $\vec{p}_{n,t}$ for each particle:

$$\vec{v}_{i,t+1} = \chi\left(\vec{v}_{i,t} + \varphi_1\vec{r}_1 \otimes (\vec{p}_{i,t} - \vec{x}_{i,t}) + \varphi_2\vec{r}_2 \otimes (\vec{p}_{g,t} - \vec{x}_{i,t}) + \varphi_3\vec{r}_3 \otimes (\vec{p}_{n,t} - \vec{x}_{i,t})\right) \tag{2.13}$$

For each dimension d, the location of the attractor point $p_{n,d,t}$ is set to the personal best $p_{j,d,t}$ of the particle j that has the maximum $FDR_{i,j,d}$, as calculated by Equation (2.14):

$$FDR_{i,j,d} = \frac{f(\vec{p}_{j,t}) - f(\vec{x}_{i,t})}{|p_{j,d,t} - x_{i,d,t}|} \tag{2.14}$$

The third term attracts the particle to the fittest point that is close to it, thereby reducing the PSO's tendency to converge on a single point.

FER-PSO is specifically designed to maintain multiple populations in multimodal environments. Equation (2.15) shows its velocity update equation:

$$\vec{v}_{i,t+1} = \chi\left(\vec{v}_{i,t} + \varphi_1\vec{r}_1 \otimes (\vec{p}_{i,t} - \vec{x}_{i,t}) + \varphi_2\vec{r}_2 \otimes (\vec{p}_{j,t} - \vec{x}_{i,t})\right) \tag{2.15}$$

Particle j is the particle for which Equation (2.16) is maximised:

$$FER_{i,j} = \alpha \frac{f(\vec{p}_{j,t}) - f(\vec{p}_{i,t})}{||\vec{p}_{j,t} - \vec{p}_{i,t}||} \tag{2.16}$$

A scaling factor α is applied to balance the effect of the fitness and distance differences between $\vec{p}_{i,t}$ and $\vec{p}_{j,t}$. It is calculated using Equation (2.17):

$$\alpha = \frac{\sqrt{\sum_{d=1}^{D} \left(x_d^u - x_d^l\right)^2}}{f(\vec{p}_g) - f(\vec{p}_w)} \tag{2.17}$$

where D is the number of decision variables, x_d^u and x_d^l are the upper and lower decision space boundaries in dimension d respectively. The fittest and worst-fit particles in the population are denoted $\vec{p}_g$ and $\vec{p}_w$ respectively. The effect of this is to calculate the ratio between the size of the decision space and the difference between the best and worst known points.

By choosing j to maximise $FER_{i,j}$, the algorithm causes the particle to communicate mostly within its own area of the decision space. Distant particles are not generally chosen unless there is a very high fitness difference because the $||\vec{p}_{j,t} - \vec{p}_{i,t}||$ term is too large. This results in the dynamic formation of species around optima. One advantage of FER-PSO is that particles never become totally isolated with no neighbours, as can happen with SPSO.

FER-PSO suffers from one difficulty however, in that the value of α is determined in part by the size of the decision space. This means that if the swarm is optimising an environment with a very large decision space but where all optima are clustered in a single area, the algorithm will select particles based mostly on the difference in fitness, preventing multiple populations from emerging.

2.7.12 mQSO

To improve PSO performance on dynamic environments, Blackwell and Branke [2006] developed mQSO. This is a combination of **m**ultiswarms with **Q**uantum **S**warm **O**ptimisation, a standard PSO which was modified in several ways to optimise its performance on the Moving Peaks benchmark.

Firstly, the population is divided into several *subswarms* to allow the PSO to converge on multiple optima simultaneously. This has an effect similar to using SPSO, however the subswarms (species in SPSO) are defined beforehand rather than based on the population. Should two subswarms move too close, the particles in the less fit swarm will be reinitialised – it will be forced to find a new peak.

The subswarms are subject to an anticonvergence measure: if all of the subswarms have converged below a given radius, the least fit one is killed and its particles reinitialised. This prevents the system from wasting resources on an underperforming peak and becomes more and more critical as the number of peaks increases. A similar technique was used by Li et al. [2006] to improve SPSO's performance on the Moving Peaks benchmark.

Finally, to allow the PSO to react more quickly to peak movements, a *quantum* swarm is used. The population is divided into quantum and normal particles. Quantum particles are so called because their behaviour loosely resembles the behaviour of subatomic particles in quantum mechanics. At each iteration, the quantum particles are moved randomly within a hypersphere of radius r_{cloud} according to a uniform volume distribution. The hypersphere is centred on the fittest known point. The rest of the particles follow the normal particle movement equations (2.5) and (2.8).

The main difficulty with mQSO is setting the r_{cloud} parameter. The ideal value depends largely on the expected size of the landscape changes. When optimising real-world functions, this information is rarely available.

2.8 Summary

In this chapter we discussed the importance of optimisation and outlined the representative algorithms within the Evolutionary Computation family. The difficulty of optimising multimodal environments was shown, along with the techniques developed to handle them. Finally we have described the methods used to measure performance, especially in dynamic environments.

In the coming chapters we will detail how the performance of PSO on multimodal environments can be improved. The next chapter presents two new speciated algorithms, ANPSO and ESPSO. These algorithms are designed to reduce the sensitivity to user-set parameters. In Chapter 6 we will present a technique that can be added to most numerical optimisation algorithms to vastly improve local convergence speed.

Chapter 3

Speciation-based PSOs

In the biological world, countless species have developed. Each species exploits a niche within the environment; it represents an adaptation to both its habitat and circumstance. Individuals from different species cannot breed, allowing each species to become as effective as possible in its particular habitat by specialising. For example, most fish have developed fins – a definite advantage in an aquatic environment. In air though, fins provide no advantage so few land-based animals have them.

Particle Swarm Optimisation (PSO) algorithms do not implicitly contain a speciation mechanism – any individual is able to interact or breed with any other, even if indirectly. To contrast this with real life, it is impossible for a cat to breed with a dog.

In optimisation, speciation allows a group of individuals to specialise in a particular niche and represent a particular solution. Without interference from each other, different groups of individuals are able to occupy their respective niches within the decision space. The final result is that instead of the algorithm returning just one solution, it returns several.

An algorithm that returns multiple solutions provides the user with choice. In general, the algorithm has only partial knowledge of the domain it is optimising. Many aspects cannot be represented properly: for example if the optimisation task involves assigning jobs to people, how does one fully encode the politics of a group? Perhaps Bob and Sarah work well together, except if Fred is in charge. Or Jenny is excellent at making widgets, except on Thursdays when she is sore from her weekly physiotherapy appointment. Encoding all these minor details into the algorithm would be too time-consuming, however a human operator could use them to choose between two otherwise equally good solutions.

Speciated algorithms are also less likely to suffer premature convergence. By maintaining several species in different areas of the decision space, even if one species becomes trapped in a local optimum, the rest are still free to explore other areas. Algorithms can also be proactive in this regard – if a species has not improved in fitness for some time its individuals can be moved elsewhere – either to aid another species or redistributed over the entire decision space.

3.1 Overview

This chapter will discuss an existing speciated PSO algorithm, SPSO in detail and analyse its advantages and disadvantages. An overview of this algorithm was given in Section 2.7.10. We will then detail an enhancement to this algorithm in order to mitigate its primary weakness. We will also propose a new algorithm that does not require the user to specify any parameters, instead using population statistics to adapt to the given problem. We will only be presenting the theory here; the experiments and results supporting our claims are presented in Chapter 5.

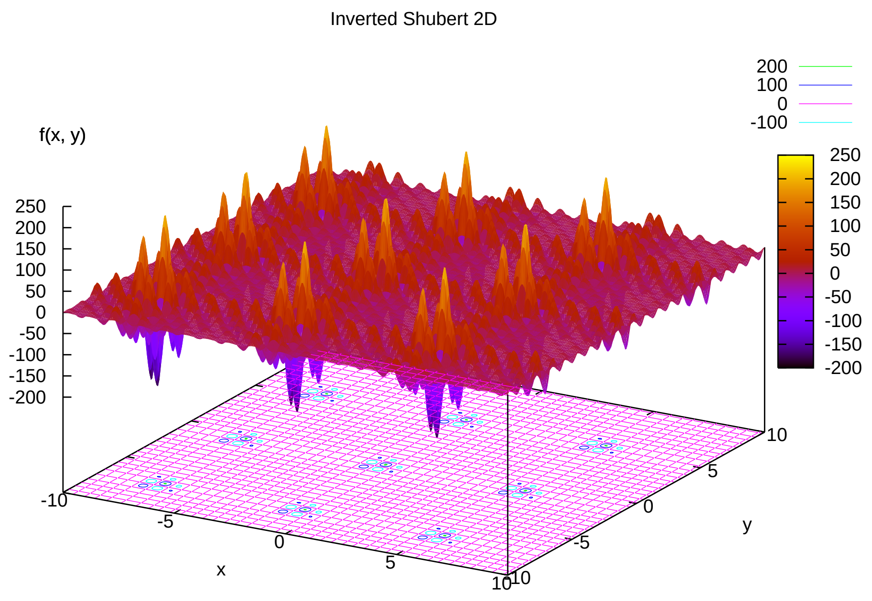


Figure 3.1: Multimodal functions contain many peaks. Pictured is the Inverted Shubert function in 2 dimensions. The 18 global optima are difficult to find amongst the many local peaks.

3.2 Challenges of multimodal functions

A fitness landscape is multimodal if there is more than one peak, as discussed in Section 2.7. Figure 3.1 shows an example of a multimodal function. Multimodal function optimisation is hard. It is the equivalent of searching the Himalayas for Mount Everest, blindfolded. Special care must be taken to avoid premature convergence, when the entire population exists within a local peak. Areas away from the peak are no longer searched and in most cases the algorithm is incapable of escape.

Achieving stable populations on multiple peaks requires some means of preventing information about one peak from reaching individuals converging on another. At the same time we still need some information flow, otherwise we risk having many individuals converging on local optima, greatly reducing the particles left searching for the global optimum. Speciation achieves this by dividing the population into subgroups, each searching a different peak.

3.3 SPSO

SPSO stands for Speciated PSO and groups particles based on their location within the decision space [Li, 2004]. Each group of particles is called a species, and the particles within each species are connected using the global neighbourhood topology. The species themselves are completely isolated – no communication occurs between particles of different species.

The algorithm borrows the concept of a user-specified species radius from the clearing procedure described in Section 2.7.3. In SPSO this parameter is known as r and is used similarly to the clearing procedure's δ. A species' fitness is defined as that of the fittest particle within it, known as the species seed. Each species controls an area of the decision space – any other particles within r of the species seed belong to that species. Particles can only belong to a single species – if there is contention between species for a particular particle, the particle is assigned to the fittest one.

In order to determine which particles belong to which species, at the start of each generation a list is created to store the species seeds. The particles are then iterated through in order from fittest to least fit. For each particle we check the species seed list in order. If the particle's current location is within r of any seeds, it is added to that seed's species. Otherwise it is added to the bottom of the species seed list, ensuring that the seeds are always checked from fittest to least fit. Each species acts as a subpopulation, its members connected using the global neighbourhood topology. A formal description is given in Algorithm 1.

```
$Seeds$ is the list of species seeds, initially empty
$P$ is the list of the population members, sorted from fittest to least fit

$Seeds_1 \leftarrow P_1$
for $i$=2 to $N$ do
    $found \leftarrow FALSE$
    for $j$=1; $Seeds_j \in Seeds \wedge \neg found$ do
        if $||P_i - Seeds_j|| < r$ then
            Allocate $P_i$ to species $j$
            $found \leftarrow TRUE$
        end
    end
    if $\neg found$ then
        $Seeds_{j+1} \leftarrow P_i$
    end
end
```

Algorithm 1: The procedure for allocating particles to species, run at the start of each generation

3.3.1 Variations

This section introduces the variations of SPSO that have been proposed – these have been designed to improve different aspects of its performance. The first three variations we will present are applicable to most problem types, both static and dynamic. The last is an enhancement purely for dynamic environments.

Limiting the species size

Fitness landscapes can be divided into two categories – single and multifunnel [Leopold et al., 1992]. A funnel refers to an optimum's basin of attraction or catchment area. Each funnel represents a general trend towards a significant optimum, and may contain other local optima as in Figure 3.2. PSO is able to perform well on single funnel environments because of its

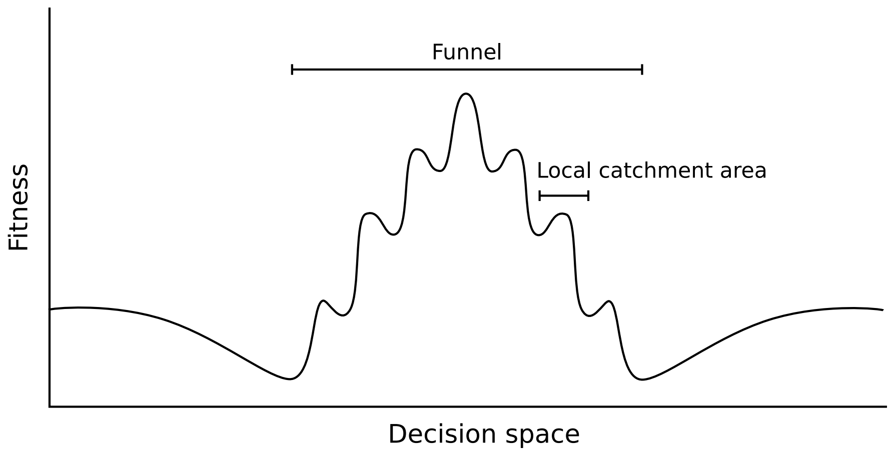


Figure 3.2: An example of a single funnel, multimodal landscape. The funnel for this peak is relatively large – the local optima are not wide or significant enough to overly distract from the global optimum.

tendency to overshoot an optimum – as it approaches each local optimum in turn, some of the particles will fly past and land in the next optimum's catchment area. Assuming they land high enough up the next slope, they will explore the area and eventually attract their neighbours, who themselves overshoot, repeating the process.

Multifunnel environments are more difficult to solve though. For particle swarms, the chance of locating the global optimum is directly related to the proportion of the initial population that is in the correct funnel [Sutton et al., 2006]. The optimisation process can be seen as a race between the funnels, and the speed that each funnel's optimum can be found is directly related to the number of particles searching within it. The funnel that has the current global best particle will slowly attract particles exploring the other funnels, diminishing the population available to those funnels. As more and more particles migrate to the currently winning funnel, it becomes harder and harder for the other funnels to take the lead and reclaim their particles. Eventually all of the particles have moved to one funnel, whether or not it represents the global optimum.

This problem is diminished in a speciated particle swarm as it is much harder for one funnel to steal another's particles. It is still possible for one funnel to "collect" most of the particles though, and quite likely when the funnels do not have similar-sized catchment areas. A larger funnel occupies more of the decision space, meaning that any given particle is more likely to have its personal best within the funnel's bounds. Even if there is a species representing each funnel, more particles will be attracted into the area of the largest funnel's species, eventually joining it. The uneven distribution of individuals may leave some species without sufficient particles to find their funnel's true optimum.

The largest funnel is not always the best one – there is no reason to expect a funnel's width correlates with its fitness. It is not advantageous to allow the largest funnel to capture most of the particles simply because it is the largest.

To avoid this issue, a new parameter P_{max} was added by Parrott and Li [2006], limiting how many particles can belong to each species. Before assigning a particle, the algorithm now checks how many particles are already in that species. If there are already P_{max} particles the particle is reinitialised, that is its velocity and location are reset to random values and the personal best memory wiped. The particle must wait until the next timestep to join a species.

Since we allocate the particles to species in order of fitness, we avoid throwing away our best solutions – the particles we reinitialise are the least fit particles that would have been allocated to that species. By stopping all the particles from collecting on the same optimum we are able to further reduce the risk of premature convergence. Reinitialising the least fit particles also helps explore the search space more efficiently.

Stable species seeds

In the original SPSO, species membership is determined by the current location of a particle. As the particles move around, so do the species boundaries. Particles may continually join or leave a species for no other reason than the particles' velocities have not yet reduced enough to fully converge. The particles are still exploring the same optimum and hopefully will eventually converge there, but continually changing a particle's neighbours means its neighbourhood best is also changing. For a particle to converge, its personal best and neighbourhood best locations must be close for several timesteps. If the neighbourhood best is changing wildly from timestep to timestep, this will not happen and the velocity will not decrease.

In order to make the neighbourhood best locations more stable, ESPSO (described in Section 3.4.1) modifies SPSO so that species are defined by the personal best instead of the current location – it makes sense to define the species centre as our best approximation to its optimum.

This also has the effect of making a species membership far more stable – allowing particles to explore further afield without risk of accidentally leaving the species. The only ways for a particle to leave a species is for the seed to move further than r from the particle, or for the particle to locate a better solution elsewhere.

Replacing the worst species

Assuming there are no particles entering or exiting it, a species acts as an independent swarm exploring some area of the decision space. As with any other swarm, it is susceptible to premature convergence. Speciated algorithms provide us with much more information than is normally available to particle swarms, allowing us to make decisions on both individual particles and whole species. It is rarely useful to keep species that have converged on unfit areas of the decision space. If we suspect that a species is unlikely to improve, and it represents an area of the decision space that is unlikely to become important in the near future, we can employ an anti-convergence measure to encourage the algorithm to explore other parts of the decision space.

SPSO was modified to achieve this by Li et al. [2006]. The convergence of each species is measured as the distance d_p between the seed particle and the furthest member of the species. At the end of each timestep, if the average convergence across every species in the swarm becomes less than a user-specified threshold Δ, we kill the worst species and reinitialise its particles. The reinitialised particles are then free to explore other optima, either converging on existing species or forming new ones.

Quantum particles

One of the challenges dynamic optimisation presents is how to handle a change in the fitness landscape. If the population has converged to a very small area, it will take some time for it to deconverge enough to start exploring again. To maintain a minimum level of diversity within each species, Li et al. [2006] adapted the idea of quantum swarms as described below. This idea was originally presented by Blackwell and Branke [2004] and discussed in Section 2.7.12.

In addition to the particles that follow the normal PSO velocity and location Equations (2.5) and (2.8), species can also replace some of their normal particles with quantum ones. Once a species' d_p drops below a threshold δ, its particles are positioned in a uniform shell distribution of radius r_{cloud} around the species seed. Half will be converted to quantum particles; these particles do not use a personal or neighbourhood best to decide where to move next, instead using the uniform shell distribution to choose a new location each timestep.

The quantum particles are included in the normal particles' neighbourhood and can become the species seed. The quantum particles do not converge tightly like the normal particles, meaning that if a quantum particle becomes the new species seed, it is likely to be relatively distant from the normal particles. Because a normal particle's velocity is dependant on how far it is from its neighbourhood best, the normal particles' velocities will increase much faster due to the larger distance, allowing them to change to exploratory mode far more quickly. This reduces the time needed to discover the optimum's new location.

3.3.2 Strengths

The primary strengths of SPSO are its simplicity and speed. It is simple to understand and to implement, and with the right setting of r is able to efficiently locate all global optima on a range of test problems. With the modifications detailed above, it is able to adapt effectively to the changes inherent in a dynamic environment, tracking multiple optima as they move around the fitness landscape.

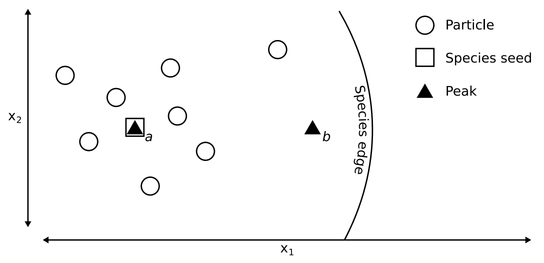
The SPSO algorithm is also fast. It consists of two stages, sorting and species allocation.

The computational complexity of the sorting pass is $O(NlogN)$ where N is the population size. The allocation pass is $O(Ns)$ where s is the number of species, which is in the range of $[1, N]$. Although this gives it a worst-case complexity of $O(N^2)$, this only occurs when every particle forms its own species. In general the number of species is small relative to the population size.

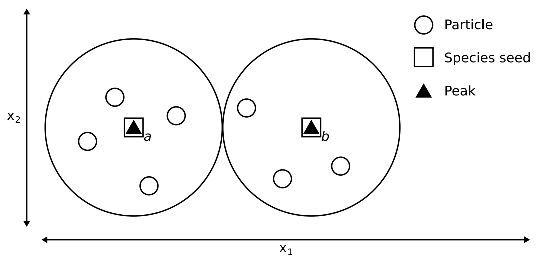
3.3.3 Weaknesses

The biggest weakness of SPSO is its sensitivity to the user-specified parameter r. If the average distance between particles is greater than r, there will be many species with only a single particle. Since a single particle has no neighbours to communicate with it will eventually converge on its personal best location, which is not necessarily even a local optimum. These stationary particles represent wasted resources – instead of exploring the landscape or even their local area, they are simply reevaluating the same point over and over again. The more multimodal a landscape is the more noticeable this becomes because the particles are less likely to come close enough to each other to join the same species.

On the other hand, if r is set too large the algorithm cannot differentiate between optima. As stated in Section 3.2, in most cases a particle swarm is unable to maintain stable populations on more than one optimum. Since each species acts as an individual particle swarm, the problems apply here too – the system is unlikely to converge on multiple optima if they are within r of each other, as depicted in Figure 3.3. In many cases the algorithm cannot converge on two optima that are within $2r$ of each other – one species will dominate the other and "steal" any particles that stray into it.



(a) Peak ***a*** has been located, but ***b*** is unlikely to be found because r is set too large; particles approaching peak ***b*** are attracted to ***a***.



(b) The swarm is able to locate both peaks because r has been set to the ideal value for this problem.

Figure 3.3: The effects of setting r too large in SPSO

The ideal radius size is one that is large enough to avoid having many small species but not so large as to make distinguishing optima difficult. The correct value depends entirely on the fitness landscape and it is impossible to tell what this should be for a given optimisation problem without multiple trial runs. For some problems such as the Shubert function depicted in Figure 3.1, there is no ideal value – the r value required to escape the local optima is too large to differentiate the peaks. SPSO is unable to effectively locate all of the optima on these problems.

3.4 Proposed speciated PSOs

SPSO's weaknesses make it difficult to use on arbitrary problems. It works well if we know how far apart the optima are, but this is extremely difficult to determine without in-depth knowledge of the environment in question. Often the only way to tune r is to perform several trial runs; a costly exercise in terms of both the number of evaluations and amount of human time required. In this section we propose two alternatives to SPSO. These algorithms are far easier to use as they do not require their parameters to be tuned prior to a run.

3.4.1 ESPSO

ESPSO [1] extends SPSO to reduce the sensitivity to r. It stands for Enhanced Speciated PSO and is designed to prevent established species from interfering with the discovery of nearby optima.

Increasing the robustness of r

While ESPSO still uses the r parameter, it is far less sensitive to the value. If the user has no idea what the ideal value is for a particular problem, he or she can set it to a large number knowing that the algorithm will still work effectively.

To prevent interference between species representing different optima, once a species has converged it is isolated from the main population. To determine whether a species has converged or not, a time-based measure is used. This measure is invariant to the size of the decision space, making it far more robust.

ESPSO uses the same method as SPSO for determining which species an individual should belong to, except that it uses the personal best instead of current location as discussed in Section 3.3.1. ESPSO classifies each particle as being in either a species or a subswarm. Species act the same as in SPSO – particles are free to join and leave as they update their personal best location. Subswarms do not interact in any way with the main swarm or other subswarms. Once a particle has been allocated to a subswarm it cannot leave, as depicted in Figures 3.4 to 3.9. This prevents information about an optimum from "leaking" into other populations.

[1] A paper on ESPSO by this author was published in 2006 [Bird and Li, 2006b].

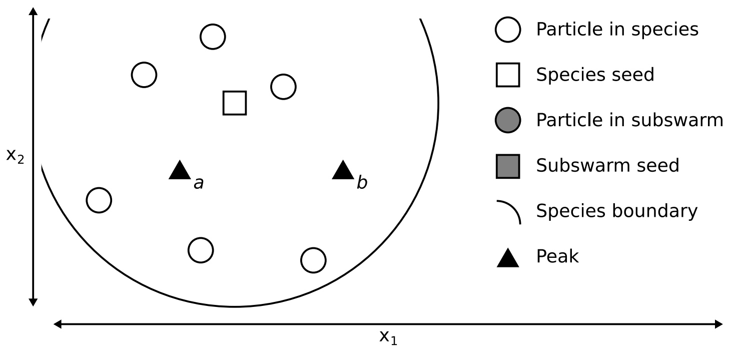


Figure 3.4: The particles have approximately located peak ***a****. The species seed (see Section 3.3) will keep moving as the particles hone in on the optimum.*

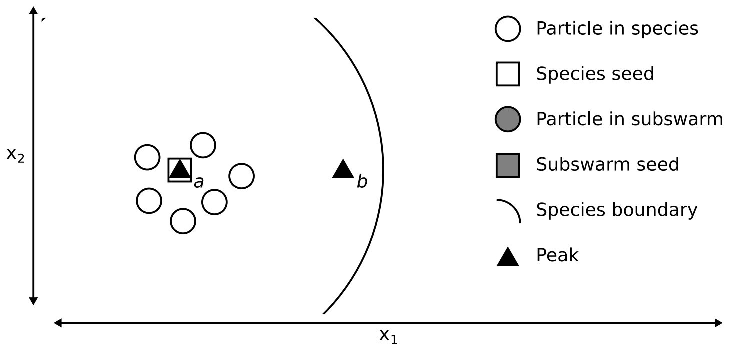


Figure 3.5: Peak ***a*** *has been located and the species seed has not moved for s steps. The best particles in the species will be placed into a subswarm that does not interact with the rest of the population. The other particles will be reinitialised.*

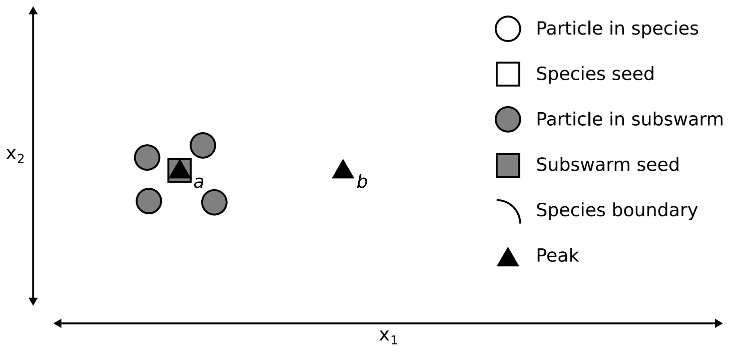


Figure 3.6: Immediately after the subswarm is created. As there is no longer a species in the area, other particles are free to come in and explore, eventually locating peak ***b****.*

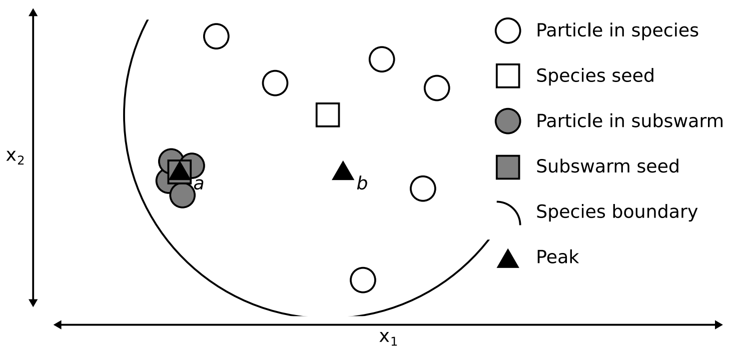


Figure 3.7: Other particles move in and start locating peak ***b****. As these particles do not interact with the particles on peak* ***a****, there is no interference from the existing optimum.*

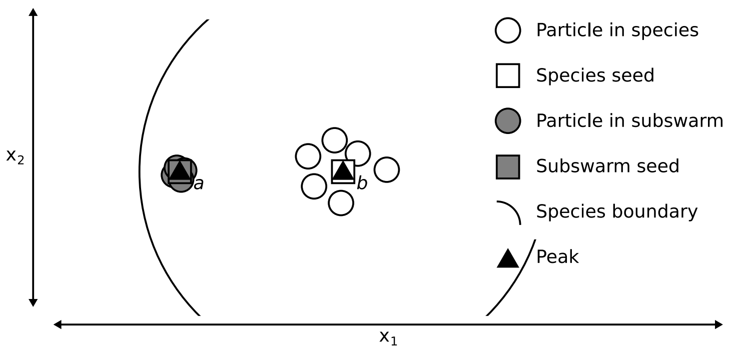


Figure 3.8: As happened with peak ***a****, the best particles in the species are about to be placed in a subswarm.*

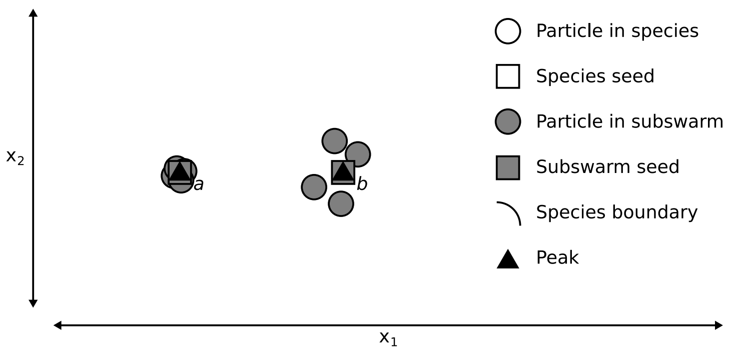


Figure 3.9: Immediately after the subswarm is created. The particles on peak ***b*** *will continue to converge.*

Once the particles that are not in subswarms have been allocated to species using the SPSO algorithm, we look for species that should be converted to subswarms. Species that have not yet accurately located an optimum will update their seed position frequently as their members find better solutions. As the optimum becomes better defined, locating fitter points becomes harder and harder, meaning that the seed position will change less often. This indicates the species has stabilised on a particular optimum. A species is converted to a subswarm if the location of its seed has not moved for s timesteps, as depicted in Figure 3.5.

The subswarm is created with P_{max} particles. If the originating species contains more than P_{max} particles, only the fittest P_{max} will be kept – the rest are reinitialised as shown in Figure 3.6. If there are fewer than P_{max} particles in the species, extra ones will be created with random locations and velocities within a hypersphere centred on the seed particle. The radius of the hypersphere depends on how many existing particles there are: if there is only one, the new particles are created within r of the species seed, otherwise they are created within d_p, defined in Section 3.3.1.

To avoid the situation of multiple subswarms converging on the same optimum, if the seeds of two subswarms are closer than a user-specified parameter δ, the least fit subswarm is killed. If necessary, new particles are created randomly around the decision space to restore the overall population to its initial level. While it is a distance-based measure, δ is easy for the user to set – it represents the minimum difference between two solutions that the user cares about. In real world problems there are usually limits on how finely a system can be tuned. When building a house there is no meaningful difference between a 2.000m and a 2.001m wooden beam – its day to day length will vary by more than this due to temperature

```
$Seeds$ is the list of species seeds, created by SPSO in Algorithm 1
$Told$ is the time there has been a seed at each point at the last timestep
$Tnew$ is the time there has been a seed at each point at this timestep, initially $\varnothing$

for $i \in Seeds$ do
    if $\exists Told_i$ then
        $Tnew_i \leftarrow Told_i + 1$
    end
    else
        $Tnew_i \leftarrow 1$
    end
end
$Told \leftarrow Tnew$
for $i \in Tnew$ do
    if $Tnew_i >= s$ then
        Create subswarm around $i$ (Algorithm 3)
    end
end
Run Algorithm 4 to remove duplicate subswarms.
```

Algorithm 2: Basic procedure for ESPSO

and humidity. A formal description of ESPSO is given by Algorithms 2, 3 and 4.

Strengths

ESPSO's main strength is the robustness of its parameters. While it introduces 3 new parameters, P_{max}, s and δ, P_{max} and s are shown in Chapter 5 to be robust to a variety of optimisation problems – the optimal value was the same or very similar in all of the tested cases. δ is intuitive for the user to set; the program can just have a setting saying "Do not show solutions that are more similar than δ".

Most importantly, ESPSO is robust to the setting of r. It removes the upper restriction detailed in Section 3.3.3. The algorithm is effective even if r is set to the width of the decision

```
$seed$ is the seed particle of the species we want to convert to a subswarm
$S$ is the species' member particles, sorted from fittest to least fit
$d_p$ is the distance between the species seed and the furthest particle
$P$ is the set of particles in the main population

if $|S| = 1$ then
    $d_p \leftarrow r$
end
else
    $d_p \leftarrow 0$
    for $i \in S$ do
        if $|seed - i| > d_p$ then
            $d_p \leftarrow |seed - i|$
        end
    end
end
while $|S| > P_{max}$ do
    Reinitialise the least fit particle and remove it from $S$
end
for $p \in S$ do
    Remove $p$ from $P$
end
while $|S| < P_{max}$ do
    Create a new particle $p$ within $d_p$ of $seed$
    $S \leftarrow S \cup p$
end
```

Algorithm 3: Creating a new subswarm

```
$Subswarms$ is the list of subswarms
$S$ is an initially empty list
$initPop$ is the initial number of particles in the swarm
$P$ is the set of particles in the main population

for $i$=1 to $|Subswarms|$ do
    $found \leftarrow FALSE$
    for $j$=1; $S_j \in S \wedge \neg found$ do
        if $||Subswarm_i - S_j|| < \delta$ then
            for $k \in Subswarm_i$ do
                if $|P| < initPop$ then
                    Reinitialise $k$
                    $P \leftarrow P \cup k$
                end
                else
                    Kill $k$
                end
            end
            Remove $Subswarm_i$ from $Subswarm$
            $found \leftarrow TRUE$
        end
    end
    if $\neg found$ then
        $S \leftarrow S \cup Subswarm_i$
    end
end
```

Algorithm 4: Removing duplicate subswarms

space. In this case it no longer exhibits speciation as specified by the SPSO algorithm and acts more like sequential niching (see Section 2.7.7). While the optimal r setting varies depending on the fitness landscape, the performance penalty for choosing a larger than optimal value is low.

Weaknesses

If r is small and the landscape is multimodal the algorithm can cause a population explosion. This happens when many subswarms containing only a few particles are formed. Many extra individuals have to be created so that each subswarm has P_{max} particles, increasing the total population. The excess particles in unpromising areas waste evaluations, reducing performance. This can become critical in dynamic environments because the algorithm takes longer to react to a landscape change. The issue can be mitigated by killing the least fit subswarm as detailed in Section 3.3.1, although in this case the particles would not be replaced. The process can be repeated until the total population reduces to the desired level.

3.4.2 ANPSO

Instead of requiring the user to specify the species radius r, ANPSO (Adaptive Niching PSO)[2] adaptively determines it from the population statistics at each iteration. This method uses the intrinsic nature of the particles to converge on an optimum and creates species when they do so. When a particle discovers a local peak, its velocity will reduce and it will explore the area closely. If multiple particles explore the same area it is likely that it is a peak of interest, so a species is created with those particles as its initial members. Any other particles that

[2]A paper on ANPSO by this author was published in 2006 [Bird and Li, 2006a].

later converge on the peak will also join the species. The adaptive nature of r means ANPSO is a form of parameter control, as discussed in Section 2.4.11.

The set of particles is represented by P, which has N members. In **step 1**, for each member p_i of the population, we measure the distance d_i to the closest other particle in the decision space using Equation (3.1). We set r to the average of these distances, as shown in Equation (3.2) and Figure 3.10. $||p_i - p_j||$ measures the Euclidean distance between p_i and some other particle p_j.

$$d_i = \min \{||p_i - p_j|| \, ; \forall j \; p_i, p_j \in P \wedge p_i \neq p_j\} \tag{3.1}$$

$$r = \frac{\sum_{i=1}^{N} d_i}{N} \tag{3.2}$$

In **step 2**, an undirected graph g is created containing a node for each particle. Initially there are no edges between any of the nodes. **Step 3** finds every set of particles that have been close to each other for at least 2 steps. The algorithm finds the pairs of particles that are within r of each other, as shown in Figure 3.11. A counter is maintained for each pair – if the pair has been close for two or more steps, a species is formed. Other particles are then free to join the species if they stay within r of any of the existing members for a number of steps s, where s is usually set to 2. A formal description is given in Algorithms 5 and 6, which are invoked at every iteration.

Particles will leave a species if they are further than r from the nearest particle within their species for s consecutive steps. As a species is defined as a set of particles that are close together, it is impossible to have a species containing only one particle. If all of a species'

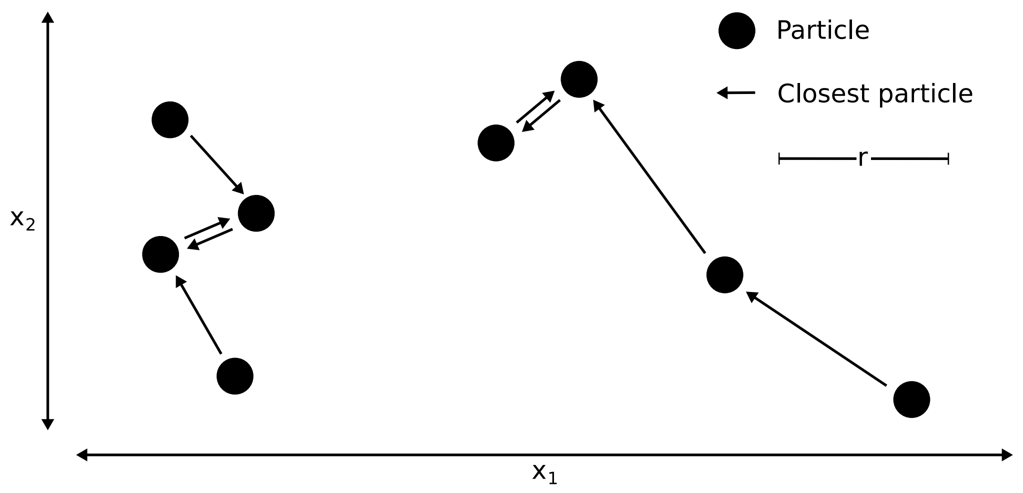


Figure 3.10: Showing the distance from each particle to the particle closest to it. r is calculated by averaging these distances.

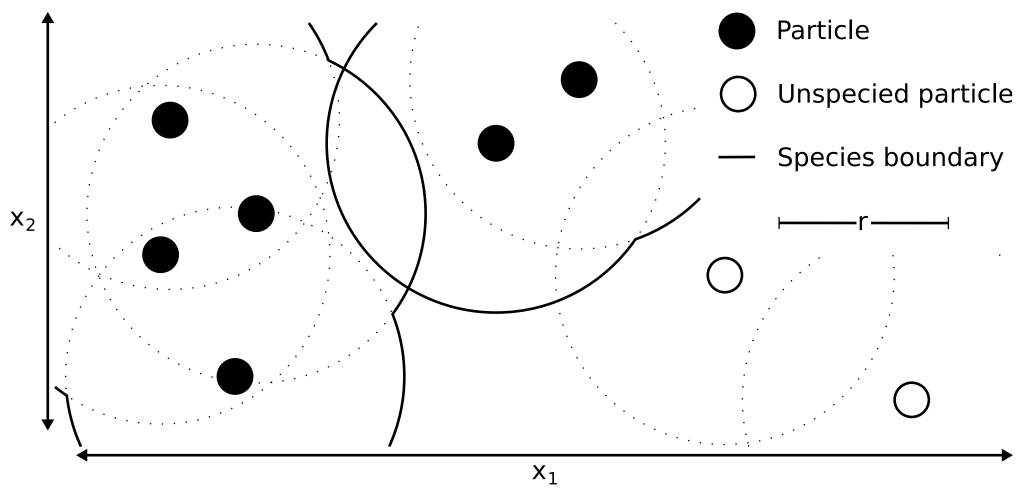


Figure 3.11: If a particle remains within r of another for s consecutive steps, the particles form a species.

particles leave, the species will disappear.

The algorithm tracks how many steps each particle has been within r of every other particle in an array called t. At each step, if two particles are closer than r, the counter is incremented for that pair. Every step that the two particles are apart decrements the counter. A particle is considered to be in the same species as another particle if the counter is greater than or equal to s. By requiring particles to be away for more than one step before allowing them to leave a species, the species are more stable.

The counter for each pair stays within the range of $[0, 2s]$ in order to allow particles to join or leave species at a reasonably fast pace. If $s = 2$ and two particles converge at step 10, normally they would form a species at step 12. Presumably they had been apart for the first 10 steps and the counter decremented every timestep they were apart. If the counter was allowed to go below 0, when they initially converged their counter would have been at -10. In this case they would have to wait an additional 10 steps before forming the species, just to return the counter to 0. The upper limit is 4 so as to allow a particle to leave a species if it has been distant for more than 2 consecutive steps.

As the number of particles in each species is generally low, the particles within each species use a fully connected topology. With very small populations most neighbourhood topologies act similarly to this; with even a moderately connected topology each particle is connected to almost every other. When using the global topology only one neighbourhood best needs to be computed per species; most other topologies require one computation per particle.

The unspecied particles are placed in a von Neumann neighbourhood which is updated

t is a N x N array that is created at the beginning of the run to track how many steps each particle has been near every other particle. Initially every element is 0.

```
1. Determine $r$ using equation (3.2);
2. Create an undirected graph $g$ containing a node for each particle, but no edges;
3. Calculate values for $t$ and add edges to $g$:
for $i=1$ to $N-1$ do
    for $j=i+1$ to $N$ do
        if $||p_i - p_j|| < r$ then
            Increment $t_{i,j}$;
            if $t_{i,j} > 2s$ then
                $t_{i,j} \leftarrow 2s$;
            end
            if $t_{i,j} >= s$ then
                Create an edge in $g$ from $p_i$ to $p_j$;
            end
        end
        else
            Decrement $t_{i,j}$;
            if $t_{i,j} < 0$ then
                $t_{i,j} \leftarrow 0$;
            end
        end
    end
end
```

Algorithm 5: The procedure for creating the species graph. s controls how long two particles have to be close to form a species. t is used to keep track of how long each particle has been close to every other particle.

```
After creating the graph in Algorithm 5:
Create a set variable $visited \leftarrow \emptyset$;
Create a set variable $reachable$;
for $i$=1 to $N$ do
    if $p_i \notin visited$ and $d_i < r$ then
        $reachable \leftarrow \{\forall j\; p_i, p_j \in P \wedge p_j$ reachable from $p_i$ in $g\}$;
        Create a new species $u \leftarrow \{p_i\}$;
        for $p \in reachable$ and $p \notin visited$ do
            $visited \leftarrow visited \cup \{p\}$;
            $u \leftarrow u \cup \{p\}$;
        end
    end
end
```

Algorithm 6: Creating the species from the species graph.

each step. This provides good results on a wide range of problems and encourages simultaneous convergence on multiple optima, as discussed in Section 2.4.9. As particles form species, they are removed from the unspecied network. This serves two purposes: it prevents a particle from being attracted away from its local optimum by other particles, and causes the rest of the population to "forget" about the newly found optimum, allowing it to explore other areas of the decision space.

Strengths

ANPSO's main strength is that it does not require the user to set any parameters. It replaces the distance-based parameter used by most speciated algorithms with a time-based parameter s. As will be shown in Chapter 5, this parameter has been found to be highly robust across all of the problems it has been tested on. The species size is now determined adaptively using the population statistics without requiring any problem-specific knowledge.

In addition, ANPSO uses the full power of the PSO to locate new optima by allowing all of the unspecied particles to communicate. PSO has been shown to be very effective at locating single optima by exploiting the collaboration between individuals, however most speciation algorithms do not make use of this, isolating the particles into small groups from the beginning.

Weaknesses

ANPSO has two main weaknesses: speed and the difficulty of joining species once most of the particles have converged. ANPSO is more computationally expensive than some other speciated algorithms. It requires N^2 computations to determine the value for r, and again to determine the steps-together array t. However, for most usages this is not a problem – the speed penalty is not generally noticeable except for very large populations of 1000 particles or more. This penalty is also offset by the adaptiveness – in most cases it is cheaper than running some other algorithm multiple times in order to determine the correct parameter settings.

The other weakness is that as the particles converge into species, r tends to 0. As it gets smaller and smaller, the particles must wait until they have converged more tightly before they are considered a species. As long as the particles remain in the unspecied population they will slowly drag the rest of the population to their optimum, reducing the algorithm's ability to locate multiple optima in parallel. While this does not prevent the algorithm from locating new optima, it does slow it down and wastes fitness evaluations. Our testing did not show this to be a significant problem.

3.5 Summary

In this chapter we discussed SPSO in detail and introduced two new speciation algorithms. ESPSO improves SPSO by reducing its reliance on the parameter r. The original SPSO algorithm is quite sensitive to the value of this parameter, and the ideal value varies from problem to problem. If an incorrect setting is used SPSO will either miss some of the peaks or not be able to locate any global optima at all.

The second algorithm we introduced, ANPSO, does not require the user to set any parameters at all, instead it adaptively determines the most appropriate value from the population statistics. In Chapter 5 we will analyse the performance of these new algorithms, using SPSO as our performance baseline. The next chapter presents a new means to analyse an algorithm's performance, providing valuable insights into how it can be improved for a given problem.

Chapter 4

Performance metrics for dynamic environments

Performance metrics allow us to compare how well different algorithms are able to solve various functions. By emphasising certain features in the fitness landscape, and comparing performance between algorithms we can gauge their relative strengths and weaknesses, guiding us in the search for new and improved algorithms [Langdon and Poli, 2005].

When optimising static problems, we tend to concentrate on how quickly the algorithm finds the global optimum or optima. Dynamic optimisation brings an additional challenge, that is handling landscape changes. Metrics designed to measure performance on static problems cannot show this aspect effectively, meaning that we need a separate set of metrics for these environments. Chapter 2 discusses this issue in more detail.

Most dynamic optimisation metrics only measure the overall performance of an algorithm without giving insight into its strengths and weaknesses. This information would be very useful to those trying to improve existing algorithms or develop new ones. With this knowledge efforts can be directed more effectively, leading to reduced research time and ultimately, better algorithms.

In Section 2.3 we discussed several existing metrics, including peak cover and offline error. In this chapter we will be proposing a new metric, Best-Known Peak Error (BKPE). BKPE measures convergence speed, that is how quickly an algorithm can accurately find an optimum once it has located its catchment area. By combining this with peak cover we are better able to analyse the strengths of an optimisation algorithm in dynamic environments. We will be comparing these two metrics to offline error, described below. We will then show how to determine the most effective way to improve an algorithm's performance by correlating the BKPE and peak cover of its runs with the offline error.

4.1 Measuring convergence speed

BKPE[1] measures the convergence speed of the algorithm once it has located the catchment area of a peak. The peak used for the calculation is called the *best-known peak*. This is the peak with the fittest individual that was covered for the entire time since the last peak movement. This metric is calculated similarly to offline error; it is the difference in fitness between the top of the peak and the fittest individual within the peak's catchment area. While the actual peak chosen has little effect on the metric, we chose the best-known peak as it is the one the algorithm is most likely to be concentrating its resources on.

At the end of each timestep, for each peak the fittest individual within that peak's

[1] A paper on BKPE by this author was published in 2007 [Bird and Li, 2007a].

catchment area is chosen, and the fitness difference between the individual and the peak is added to that peak's error (see Equation (4.1) and Figure 4.1). Immediately before the peaks are moved, the peak with the fittest individual and that was covered at the end of every timestep since the last move is selected, and its error added to the total. After this, the error for each peak is reset to 0. The BKPE is calculated by dividing the total error by the number of timesteps[2]. This is described in detail in Algorithms 7 and 8.

$$BKPE = \frac{1}{T}\sum_{t=1}^{T} e_{t,i} \tag{4.1}$$

T is the total number of timesteps, t is the current timestep number and i is the index of the best-known peak during that time[3]. $e_{t,i}$ represents the fitness error between the chosen peak i and its fittest individual.

4.1.1 Advantages

BKPE provides insight into a specific aspect of an algorithm's performance by exclusively measuring convergence speed. Since multiple metrics can be run in parallel it is possible to compare the BKPE, peak cover and offline error for the same run. In the example of the two similarly-performing algorithms given in Section 2.3.4, by also measuring the peak cover and BKPE we would be able to differentiate the algorithms, exposing the strength and weakness of both.

[2]If there was no peak covered for every timestep between two peak movements, the BKPE cannot be calculated for this period and the timesteps between the movements are ignored.

[3]As stated above, this can only be decided in retrospect once there has been an environment change.

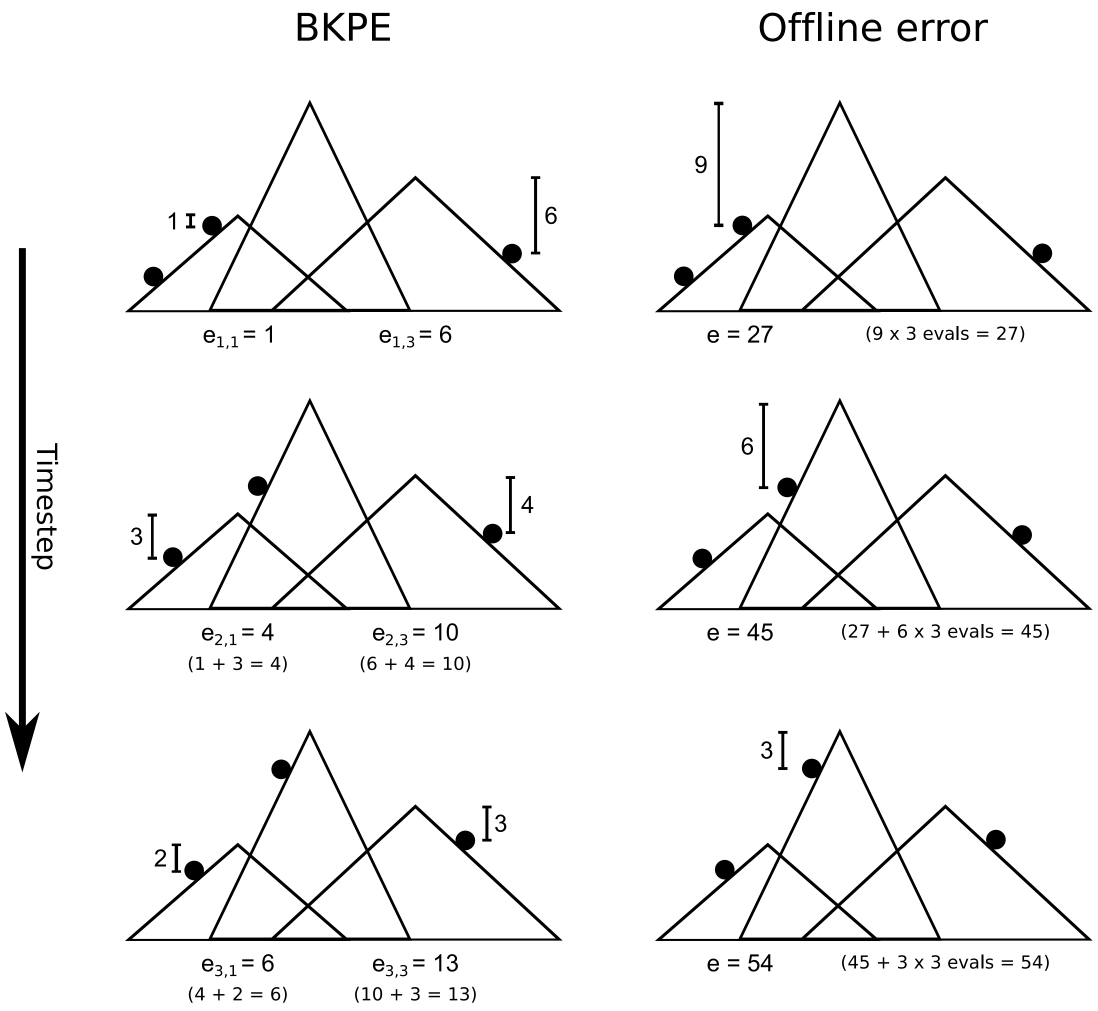


Figure 4.1: For BKPE, the current error is calculated for every known peak. Immediately before a peak movement, the error of the fittest particle on a known peak (in this case the right-hand peak) is added to the total error for the run. The errors for each peak are then reset to 0. The BKPE in the scenario above would be $\frac{13}{3\ timesteps} = 4.33$*. The middle peak is ignored because it was unknown during the first timestep. The offline error would be* $\frac{54}{9\ evaluations} = 6$*.*

```
$K$ is the set of peaks, $\vec{k_i}$ is the $i$th peak
$U$ is the set of unknown peaks
$P$ is the set of population members
$C$ is the set of population members covering peak $i$
$e_i$ is the error of peak $i$
$covering(\vec{k_i})$ returns the subset of $P$ that covers $\vec{k_i}$
$fitness(\vec{x})$ returns the fitness of point $\vec{x}$
$y$ is a temporary variable

At the end of each timestep:
for $\forall i; \vec{k_i} \in K \wedge \vec{k_i} \notin U$ do
    $C \leftarrow covering(\vec{k_i})$
    if $C = \oslash$ then
        $U \leftarrow U \cup \vec{k_i}$
    end
    else
        // Find the fittest individual on peak $\vec{k_i}$
        $y \leftarrow 0$
        for $\forall j; \vec{c_j} \in C$ do
            if $fitness(\vec{c_j}) > fitness(\vec{c_y})$ then
                $y \leftarrow j$
            end
        end
        $e_i \leftarrow e_i + fitness(\vec{k_i}) - fitness(\vec{c_y})$
    end
end
```

Algorithm 7: The procedure for calculating BKPE at each timestep.

```
$r$ is the cumulative total error, initially 0
$m$ is the number of ticks for which the metric has been computed, initially 0
$BKPE = \frac{r}{m}$
$t$ is the current tick
$l$ is the tick of the last peak movement
$K$ is the set of peaks, $\vec{k_i}$ is the $i$th peak
$U$ is the set of unknown peaks
$C$ is the set of population members covering peak $i$
$e_y$ is the error of peak $y$
$covering(\vec{k_i})$ returns the subset of $P$ that covers $\vec{k_i}$
$hidden()$ returns the subset of $K$ that is hidden by other peaks
$fitness(\vec{x})$ returns the fitness of point $\vec{x}$
$y$ and $f$ are temporary variables

**Immediately before a peak movement:**
if $\exists i; \vec{k_i} \in K \wedge \vec{k_i} \notin U$ then
    // Find the fittest individual on a non-covered
    // peak and note which peak it is on
    $f \leftarrow -\infty$
    $y \leftarrow 0$
    for $\forall i; \vec{k_i} \in K \wedge \vec{k_i} \notin U$ do
        $C \leftarrow covering(\vec{k_i})$
        for $\forall j; \vec{c_j} \in C$ do
            if $fitness(\vec{c_j}) > f$ then
                $y \leftarrow i$
                $f \leftarrow fitness(\vec{c_j})$
            end
        end
    end
    $r \leftarrow r + e_y$
    $m \leftarrow m + t - l$
end

**Immediately after a peak movement:**
$l \leftarrow t$
for $\forall i; \vec{k_i} \in K$ do
    $e_i \leftarrow 0$
end
```

Algorithm 8: The procedure for calculating BKPE when a peak moves.

4.1.2 Disadvantages

The metric assumes that the best-known peak is the one the algorithm is concentrating its population on, but this is not always the case. It may be that the best individual is isolated in a small species so that while the algorithm knows about the peak, it is unable to focus its search there by adding individuals. This adds a level of noise to the metric as the algorithm is sometimes penalised for "knowing" the best peak but exploring elsewhere.

Since BKPE is measured per timestep and not per evaluation, algorithms with larger populations have an advantage. An algorithm that uses a population of 100 has that number of guesses with which to locate the optimum each timestep. An algorithm with only 50 members gets half as many guesses and so is likely to have a higher BKPE. It is only valid to compare the BKPE scores of two algorithms when they both use the same population size.

4.2 Relating peak cover and BKPE to offline error

Peak cover and BKPE are orthogonal component parts of offline error. To obtain a good offline error score, the algorithm must converge quickly (measured by BKPE) and track as many peaks as possible (measured by peak cover). An algorithm with a low BKPE and high peak cover will achieve a low offline error, as shown in Figure 4.2. Conversely, an algorithm that has a high BKPE and low peak cover is likely to have a high offline error.

Let us imagine an algorithm possessing an oracle. The oracle knows the exact location of each peak at each timestep but not the height. It must keep an individual on each peak in order to return the global optimum. This algorithm would have a peak cover of 1 and a BKPE very close to 0. Since we track every peak, the best-known peak will always be the

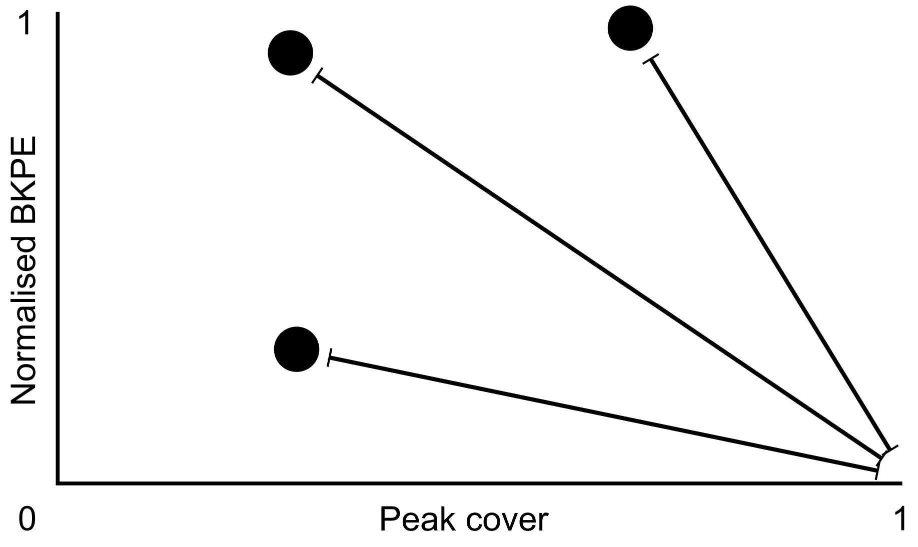


Figure 4.2: Algorithms that get a low BKPE and high peak cover will have a low offline error, thus offline error is related to the distance from the bottom right of the graph.

globally optimal peak, thus the BKPE and offline error will be almost[4] the same.

As the time needed by an algorithm to accurately locate an optimum increases, the BKPE gets larger. With each evaluation, a slower-converging algorithm incurs a larger error penalty, increasing its offline error score. The same effect can be observed as peak cover decreases. As the algorithm reduces its diversity it becomes less likely to have individuals in the area of a new global peak. It takes longer to locate the new optimum after a landscape change, if it does so at all. Again it receives a penalty with each successive evaluation, increasing its offline error.

[4]There will be a small difference – the offline error is computed every evaluation whereas the BKPE is calculated every timestep.

4.3 Improving algorithm performance

When optimising a computer program's speed, it is wise to first profile it to find where the bottlenecks lie. Doing so allows us to concentrate our efforts where they will provide the largest benefit. To this end there exist profilers for almost every computer language, and optimising a program without profiling it first is considered a waste of time.

Optimisation algorithms are the same; when trying to improve an algorithm we often make blind assumptions about what can be improved, without using any empirical evidence to guide us. For example, simply improving the peak cover of an algorithm may not improve its offline error. The algorithm may already maintain sufficient diversity to cope with most landscape changes, and by increasing exploration further we may spread the search too thinly.

It is possible to profile an algorithm by performing a number of runs and correlating the peak cover and BKPE with the offline error. Regardless of the algorithm's actual scores for the three metrics, we can see which aspect had the biggest correlation with offline error.

If there is a high correlation between BKPE and offline error, we can say that the runs where the algorithm converged faster generally scored better. This implies that if we are able to improve the convergence speed overall, we should see a decrease in the offline error. Conversely if there is little to no correlation between BKPE and offline error but a high correlation with peak cover, we can confidently say that improving the BKPE score is unlikely to affect the offline error much. The correlation between peak cover and offline error suggests we should be researching how to improve the diversity preservation aspects of the algorithm.

Having a peak cover of 1 or a BKPE of 0 means nothing if overall performance is still poor. By profiling an algorithm before attempting to improve it, we can spend our time on the aspects that are likely to provide the greatest benefit. An example of this is demonstrated in the next three chapters.

4.4 Summary

In this chapter we have described a commonly used metric, offline error. We have also detailed two metrics, peak cover and BKPE, that exclusively measure the exploration and exploitation aspects of an algorithm respectively. Finally we showed how by correlating these two metrics with offline error we are able to profile an algorithm, indicating how its overall performance can best be improved. In the following chapter we will use these metrics to evaluate the performance of several PSO variants, including the ESPSO and ANPSO algorithms proposed in Chapter 3.

Chapter 5

Experiments and analysis

In this chapter we will evaluate the effectiveness of the techniques presented in Chapters 3 and 4. It is divided into two parts. In the first part we will evaluate the robustness and performance of ESPSO and ANPSO. We will be using SPSO, described fully in Section 3.3, as our benchmark for comparing performance.

The second part of this chapter uses BKPE and peak cover to measure the performance of these three algorithms plus a baseline PSO on a dynamic test suite. We will measure how well these two metrics correlate with offline error and how they can be used to guide future improvements to the algorithms.

5.1 Proposed speciation algorithms

This section describes our testing methodology to determine how robust ESPSO and ANPSO's parameters are and whether they are effective optimisation techniques. We will then provide the experimental results and analysis along with recommended values for r, s and P_{max} –

the three speciation-related parameters of the algorithms.

5.1.1 Experimental setup

The performance of ANPSO and ESPSO will be tested on five multimodal test functions: Branin RCOS, Six-Hump Camel Back, Deb's First Function, Himmelblau and Inverted Shubert 2D. The equations for each are given in Appendix A. These functions represent a variety of different problem types:

- Branin RCOS [Branin, 1972] and Himmelblau [Beasley et al., 1993] both have peaks with large catchment areas. The two right-hand peaks of Himmelblau can cause SPSO problems if r is set too large, even if it is not set large enough to encompass both – the peak that is first located has a tendency to "steal" particles that are exploring the other. This tendency worsens as r is increased. These functions are shown in Figures 5.1 and 5.2 respectively.
- Six-Hump Camel Back [Michalewicz, 1996] has two global optima with relatively large catchment areas, however there are also 4 local optima for the particles to become trapped in. See Figure 5.3.
- Deb's First Function [Deb and Goldberg, 1989] is quite simple to solve, although the proximity of the peaks requires SPSO's radius parameter to be set carefully. See Figure 5.4
- Inverted Shubert 2D [Li et al., 2002] is the 2-dimensional form of Shubert. It is the most difficult to solve as it is highly multimodal. There are 18 global optima, however

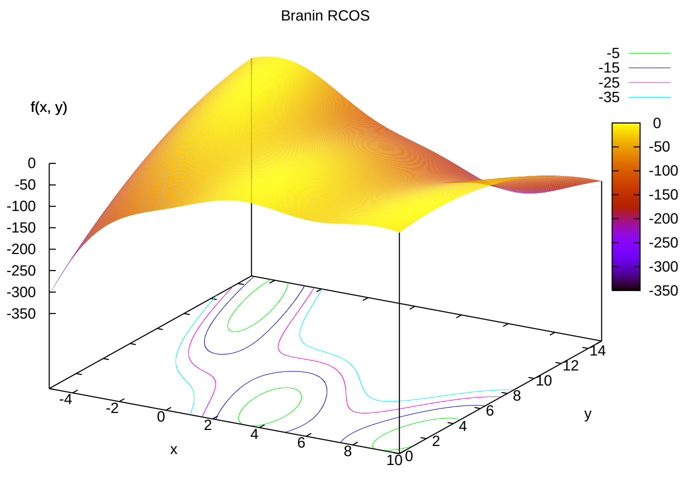


Figure 5.1: The fitness landscape of Branin RCOS

their catchment areas are extremely small. The optima are located in pairs which are evenly distributed throughout the search space. The optima in each pair are very close (see Figure 3.1), meaning that for SPSO to find them all r has to be set very small. However with this setting most of the particles become trapped in the many local optima.

A real world test case was not used due to the difficulty of determining the exact location and height of the global optima, as is required by the metrics we will be testing in the second part of this chapter. In addition, as the properties of real world problems vary significantly, the relative effectiveness of an algorithm on one real world problem does not provide much indication of how it will perform on another. The set of functions we will use were chosen because they have well-known properties and are easily visualisable.

For each experiment we performed 50 runs, recording the number of evaluations required to locate all of the global optima to a fitness error of $e = 0.00001$. The *success rate* was mea-

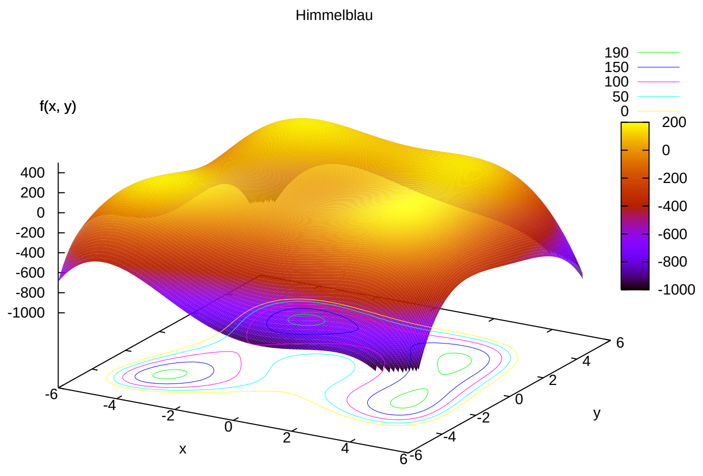


Figure 5.2: The fitness landscape of Himmelblau

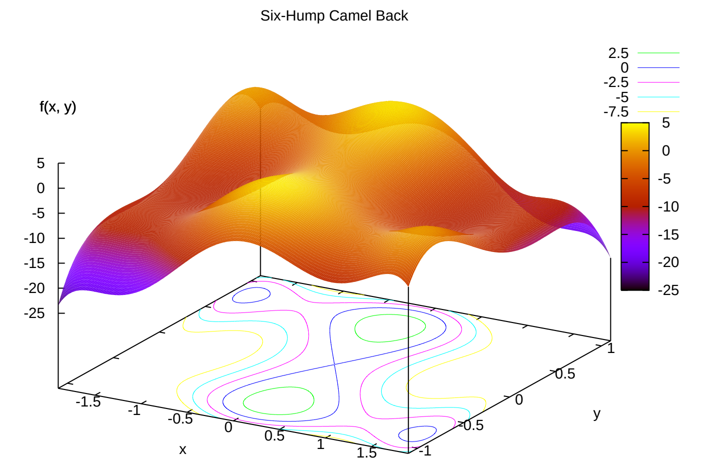


Figure 5.3: The fitness landscape of Six-Hump Camel Back

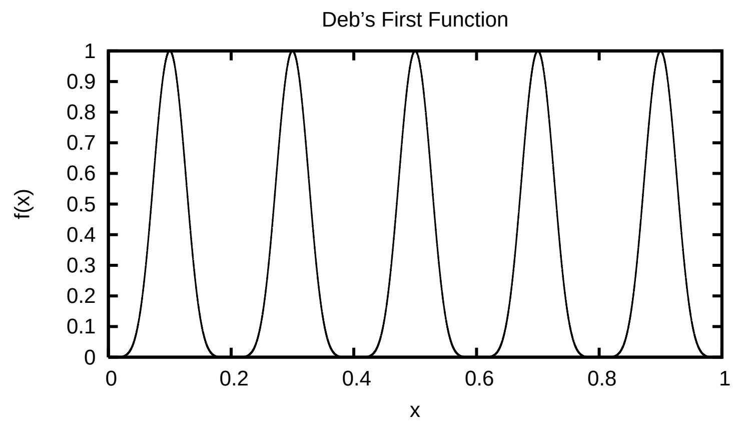


Figure 5.4: The fitness landscape of Deb's First Function

sured as the number of runs where every peak was located. A run is considered unsuccessful if the swarm failed to find all of the global optima within 2000 timesteps. Unless otherwise stated, for Inverted Shubert 2D we used a population size of 500, the other functions used a population of 30.

Testing robustness

ESPSO and ANPSO were both developed to alleviate the user of having to specify the optimal parameter settings. We therefore need to make sure the parameters they do have are robust across a range of fitness landscapes.

Both algorithms use the parameters s and P_{max}. Although the algorithms use them in slightly different ways, they have very similar meanings. For ANPSO, s controls how long two particles need to have been close to each other to form a species. For ESPSO it is the number of timesteps to wait before converting a species into a subswarm. P_{max} is used by ANPSO to specify the maximum number of individuals that can belong to a species. For ESPSO it defines the size of each subswarm, that is every subswarm has exactly P_{max} members.

We have tested s using the values 2, 3, 4, 5, 6, 8, 10, 12, 14, 16, 18 and 20. P_{max} was tested with the same set of values, excluding 3 and 5. We will also look at the effect of the total population size – for Inverted Shubert 2D we will test with populations from 100 to 1000; for the others we use populations from 10 to 100. The population tests were conducted with $P_{max} = 6$ and s equal to 2 and 3 for ANPSO and ESPSO respectively, as these values provided the best performance. For each test function, the ideal value of r for SPSO was found by trial and error. We define "ideal" to mean the value that allows all of the optima to be located using the least number of evaluations. Unless otherwise stated, for ESPSO we have set r to 2.5 times the ideal value for SPSO.

To test the robustness of r, it is set to $\frac{1}{2}$, $\frac{3}{4}$, 1, 2.5, 5, 7.5 and 10 times the approximate ideal radius for SPSO on each function (see Appendix A). The ideal r value for SPSO is somewhere between 50% and 90% of the distance between the closest two optima in the particular problem. This allows the largest catchment area for each species without causing undue interference between them.

Higher-dimensional functions

To test performance in environments where there are a larger number of decision variables, we have used a modified version of the Generic Hump Function proposed by Singh and Deb [2006]. The Hump function allows us to independently control the number of dimensions, the number of humps and the size and shape of those humps, making it ideal for our testing.

The Humps function consists of K humps rising from a flat surface. The height of the hump k at a given point X is shown in Equation (5.1):

$$f(x,k) = \begin{cases} h_k \left[1 - \left(\frac{d(X,k)^{\alpha_k}}{r_k}\right)\right], & \text{if } d(X,k) < r_k \\ 0, & \text{otherwise} \end{cases} \tag{5.1}$$

The distance from the centre of the hump k to X is denoted by $d(X,k)$. In Singh's testing, all of the humps have the same height, radius and shape; $h_k = 1, \alpha_k = 1$ for all k. The radius r_k was varied with the number of dimensions. The humps are placed so that they do not intersect, meaning that the distance between any two humps j and k is at least $r_j + r_k$. For our testing, we have removed the function's flat base so as to allow the algorithms to more easily find the peaks, as shown in Equation (5.2):

$$f(x,k) = h_k \left[1 - \left(\frac{d(X,k)^{\alpha_k}}{r_k}\right)\right] \tag{5.2}$$

The height of any given point in the decision space is the height of the tallest hump at that point, as in Equation (5.3):

$$f(x) = \text{Max}_{k=1}^{K} f(x,k) \tag{5.3}$$

An example of a landscape with 3 humps in two dimensions is shown in Figure 5.5. We have tested this function with 20 peaks and between 5 and 10 decision variables. The r_k for each test was set to 0.27. All of these tests used 300 particles and SPSO's r parameter was set to $2r_k$, that is twice the hump radius.

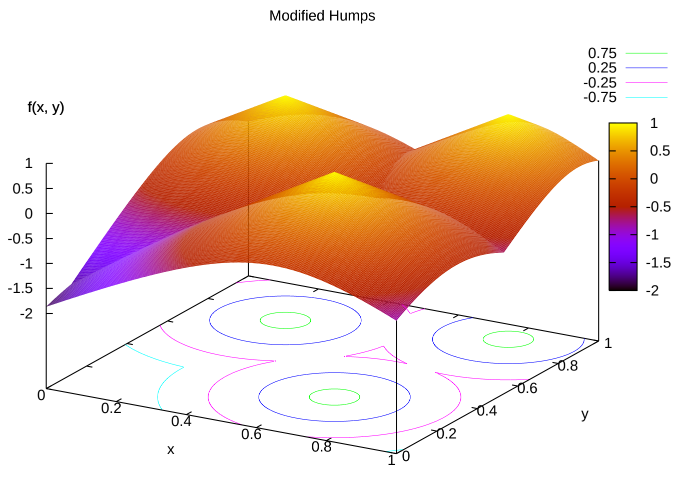


Figure 5.5: An example of the Modified Humps landscape using 3 humps in two dimensions

Comparing to SPSO

We will be using SPSO as our benchmark to allow comparison of ESPSO and ANPSO's performance with an existing algorithm. For SPSO the r parameter has been set to the optimal value for each evaluation function – the largest value that did not cause nearby peaks to interfere with each other. On most of the functions this was around half the distance between the two closest optima. To better measure the effect of ESPSO's enhancements, we have assigned particles to species using their personal best location rather than their current location. This is one of the modifications of ESPSO, described in Section 3.3.1.

5.1.2 Results and analysis

We will begin by discussing the effect of s and P_{max} on ESPSO and ANPSO and determine the ideal value for these parameters. We will then analyse the performance of these algorithms using different population sizes, before showing how ESPSO improves SPSO by making r far more robust. The final subsection will give a comparison between the performance of SPSO, ESPSO and ANPSO.

Robustness of s

Figures 5.6 and 5.7 show the success rates using different values of s for ESPSO and ANPSO respectively. For clarity we have omitted the error bars, however in all cases the variance was small. A run is only considered successful if all of the global optima have been located to within the acceptable fitness error e. Both algorithms achieved a 100% success rate on most of the functions. The only exceptions were Inverted Shubert 2D for ESPSO and Branin RCOS and Himmelblau for ANPSO. In these cases, choosing a lower value of s made the algorithm more reliable, that is the percentage of successful runs was higher.

The number of evaluations required by the algorithms to solve each of the test functions can be seen in Figures 5.8 and 5.9. Both algorithms show their best performance when s is set to a low value. For ESPSO we recommend setting $s = 3$, for ANPSO $s = 2$ is ideal for most of the functions tested. These are the values that have been used for the remainder of this thesis.

The parameter s can be seen as an "aggressiveness" control. Large values make ESPSO behave almost identically to SPSO – a species seed has to be still for a very long time before the enhancements are activated. As ESPSO becomes more like SPSO, its efficiency when using large values for r reduces. Eventually, for very large values of s, ESPSO will be unable to locate all peaks because the species are never converted to subswarms. Similarly, small values of s make ESPSO very aggressive in converting species to sub-populations. ANPSO

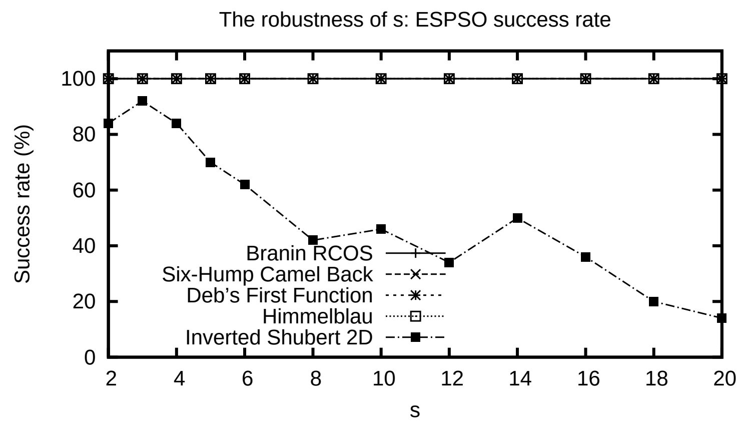


Figure 5.6: The percentage of successful runs ESPSO with different values of s. r is set to 2.5 times the optimum value for SPSO for each problem, $P_{max} = 6$.

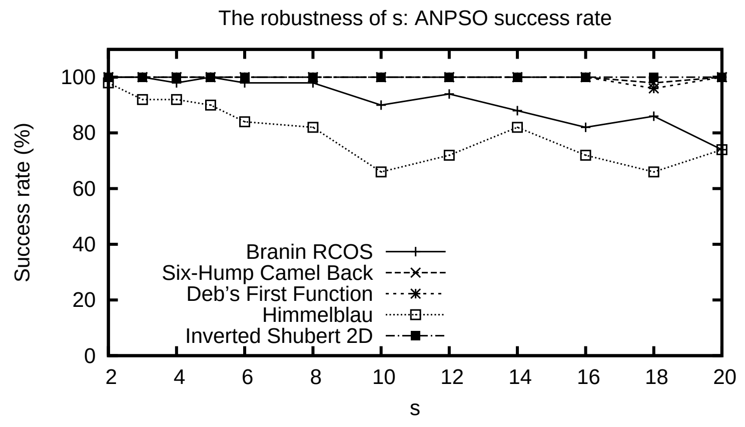


Figure 5.7: The number of successful runs for ANPSO with different values of s. $P_{max} = 6$.

performs in a similar way. Increasing s causes the particles to wait longer before they can join a species, giving them more time to attract their neighbours.

Overall it can be seen that this parameter is very robust. The time-based nature of s makes it invariant to the size of the decision space and the relative heights of the optima. This allows both algorithms to perform effectively on all of the problems tested without individual tuning.

Robustness of P_{max}

Figures 5.10 and 5.11 show that P_{max} has a much larger effect on the success rate of the algorithm. The rapid drops in success rate are caused by the algorithms running out of particles. Because the population size for the functions other than Inverted Shubert 2D is 30, large species sizes can quickly exhaust the population limit. If too many species have converged on local peaks there may not be enough particles left in the main population to locate the global peaks. A more gradual curve can be seen on Inverted Shubert 2D. Because Shubert is extremely multimodal and r is still relatively small (1.875) compared to the search space, ESPSO forms many species on local optima. As the number of particles in the sub-populations increases it takes fewer sub-populations on local optima to use up all the particles. We recommend setting P_{max} to 6 for both algorithms. This provides sufficient particles, but does not force the user to use large population sizes to locate all optima.

From Figures 5.12 and 5.13 it can be seen that P_{max} does not have a large effect on the number of evaluations required to locate all of the optima. Values smaller than 6 cause an increase in the time taken as the sub-populations formed do not have enough particles to quickly locate a peak. As the ideal value of s is very small, the sub-populations are created when the particles are still largely in exploratory mode and have not yet converged. A larger number of particles is likely to locate the peak more quickly, converging in fewer timesteps.

Effect of population size

Population size does not have a large effect on the reliability (Figures 5.14 and 5.15), provided there are enough individuals to cover all of the global optima. Since $P_{max} = 6$, this means

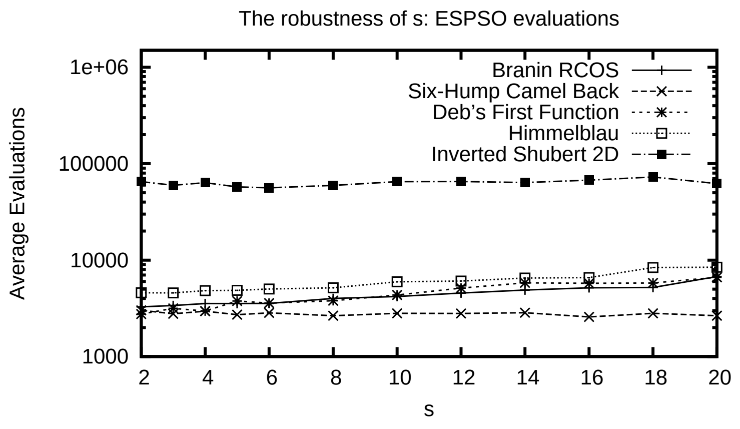


Figure 5.8: The number of evaluations required for ESPSO to locate all optima with different values of s.

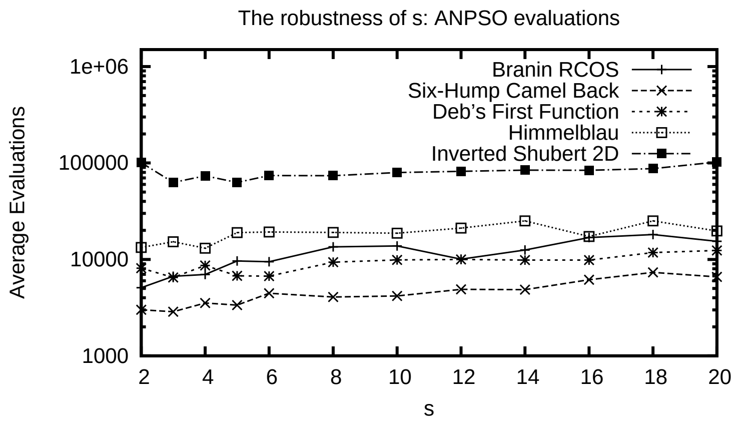


Figure 5.9: The number of evaluations required for ANPSO to locate all optima with different values of s.

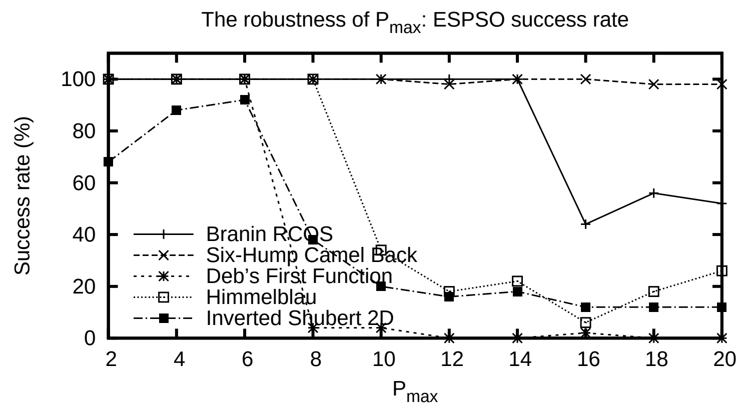


Figure 5.10: The success rate of ESPSO locating all optima with different values of P_{max}. r is set to 2.5 times the optimum value for SPSO for each problem, s = 3.

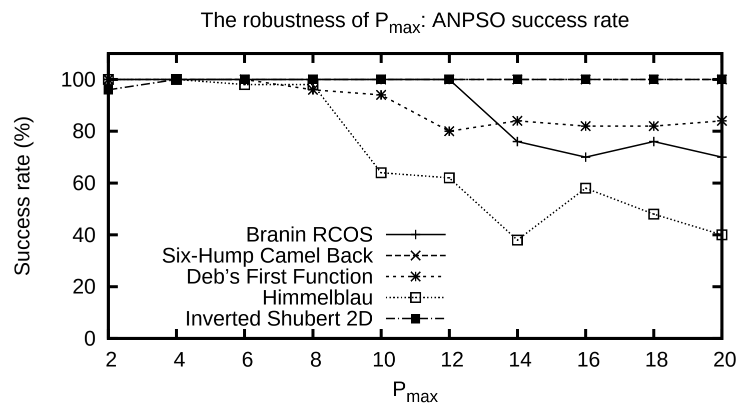


Figure 5.11: The success rate of ANPSO locating all optima with different values of P_{max}. s = 2.

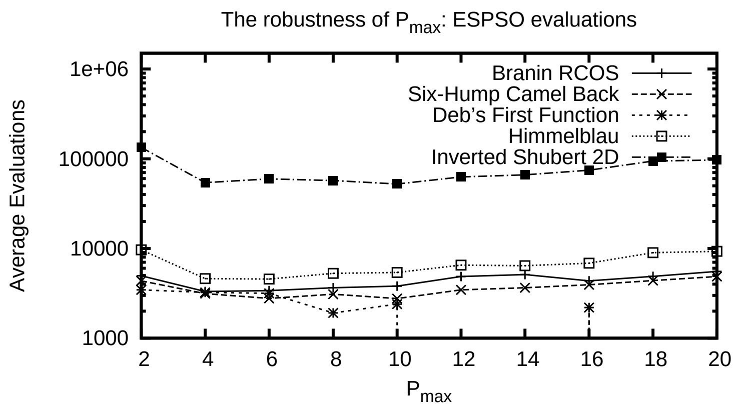


Figure 5.12: The number of evaluations required for ESPSO to locate all optima with different values of P_{max}. $s = 3$.

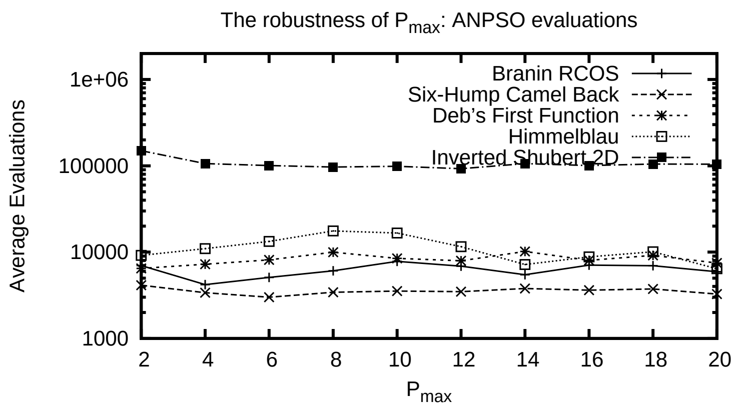


Figure 5.13: The number of evaluations required for ANPSO to locate all optima with different values of P_{max}. $s = 2$.

the algorithms required approximately 6 particles for each optimum, although they achieved 100% reliability using slightly less than this on most of the evaluation functions. The reason for this is that there are usually a few particles left in the main population after most of the optima are discovered; these were able to locate the last optimum.

Figures 5.16 and 5.17 show that larger populations use more evaluations; adding unneeded particles hinders optimisation performance. For every combination except ANPSO with Branin RCOS, the best performance was achieved with very small populations. The ideal population size for the other three test functions was the minimum that still allowed the algorithms to reliably locate the optima.

Figure 5.18 compares the population sizes needed by SPSO, ESPSO and ANPSO to locate all of the optima for the Inverted Shubert 2D function. ANPSO and ESPSO achieved 100% reliability at 200 and 300 particles respectively. SPSO required more than 600 particles to reach a 100% success rate. The population is so large because there is no value of r that enables SPSO to differentiate the global optima without causing most of the population to become trapped in local optima. Extra particles must be added to compensate.

The cost of having too many particles is illustrated again in Figure 5.19. For ESPSO, the difference between 100 particles and 1000 particles is almost an order of magnitude. For ANPSO it is only double, however ANPSO requires far more evaluations when using small populations than ESPSO. The wastefulness of adding particles to compensate for SPSO's trapped individuals is apparent here. While the number of evaluations required by the three algorithms at each population size is similar, it should be remembered that SPSO is unreliable below 500 particles. By comparing ESPSO and SPSO at the populations they first achieve

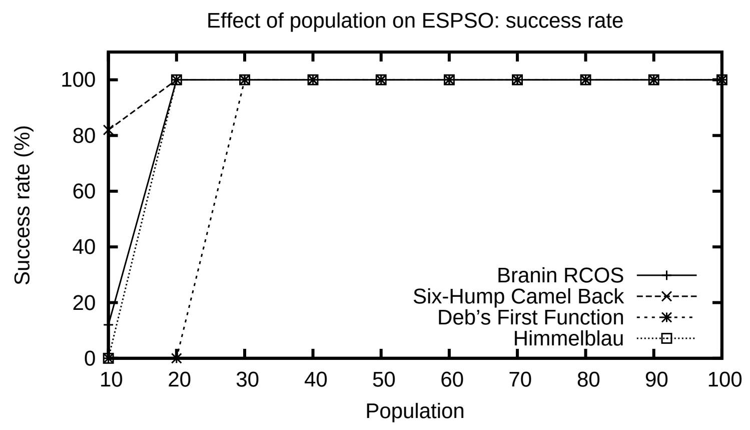


Figure 5.14: The number of successful ESPSO runs with different population sizes.

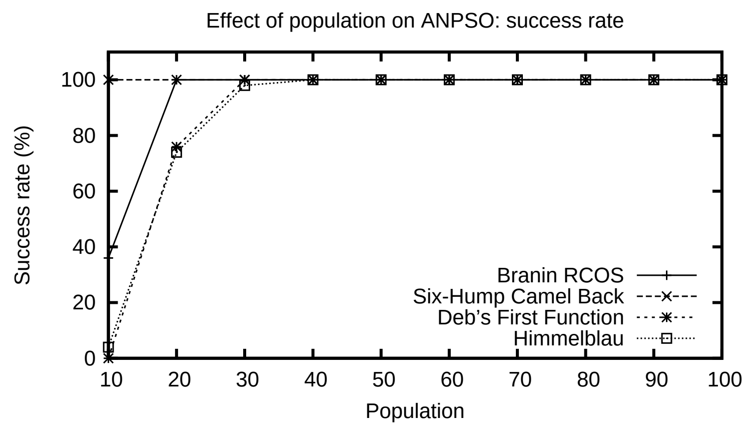


Figure 5.15: The number of successful ANPSO runs with different population sizes.

100% reliability (300 and 700 particles respectively), we can see the waste caused by the locally-trapped particles.

As the optimal number of particles differs for each test function we cannot offer a population size that would work for every environment. However it may be possible to make this property adaptive by adding new individuals when the number remaining in the main population becomes too low. This condition indicates there are too many optima for the number of particles and that it may be possible to locate more optima. To prevent a population explosion it would probably be necessary to "reclaim" particles from under-performing subpopulations before creating new individuals. This could be a focus of future research.

Robustness of r

Comparing Figures 5.20 and 5.21 we can see that ESPSO is a vast improvement over SPSO. On all of the functions except Inverted Shubert 2D, ESPSO had a 100% success rate. On Inverted Shubert 2D it required a larger value of r to avoid too many particles becoming trapped in local optima. SPSO was the opposite however; increasing r too far prevented it from locating all of the optima. If two optima are closer than r in the decision space, SPSO cannot differentiate them. As was discussed in the previous section, SPSO required a much larger population (700) to solve Inverted Shubert 2D with 100% reliability.

Figure 5.22 show the value of r has very little effect on ESPSO's performance. While there is benefit to setting it close to SPSO's optimal value, it is certainly not a requirement. In the graphs, an r multiplier of 10 corresponds to a radius larger than the search space on all of the problems except Inverted Shubert 2D. This shows it is possible to run ESPSO without

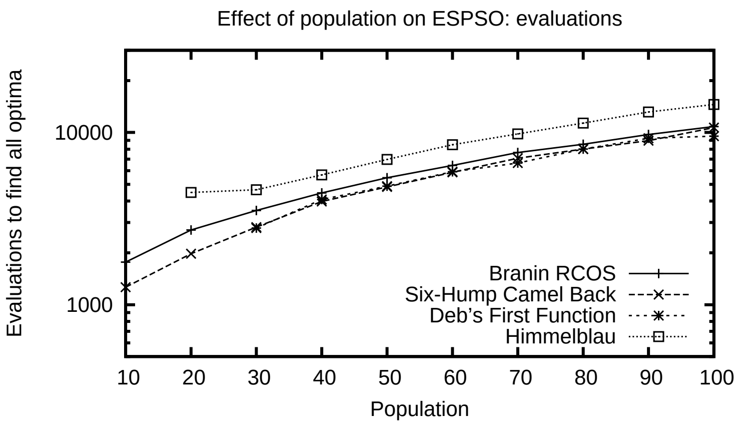


Figure 5.16: The number of evaluations required for ESPSO to locate all optima with different population sizes.

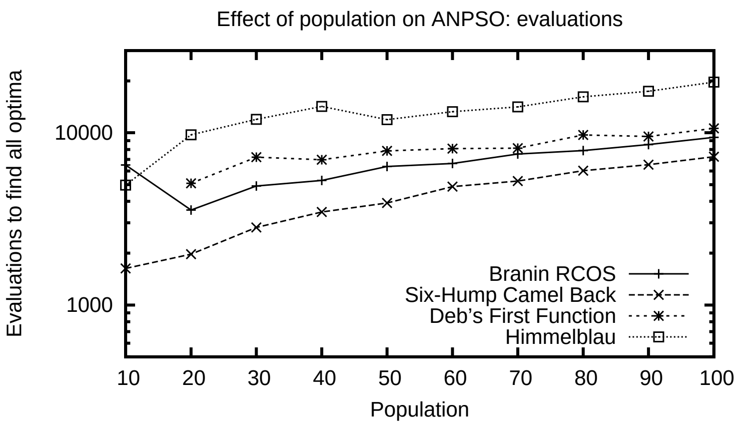


Figure 5.17: The number of evaluations required for ANPSO to locate all optima with different population sizes.

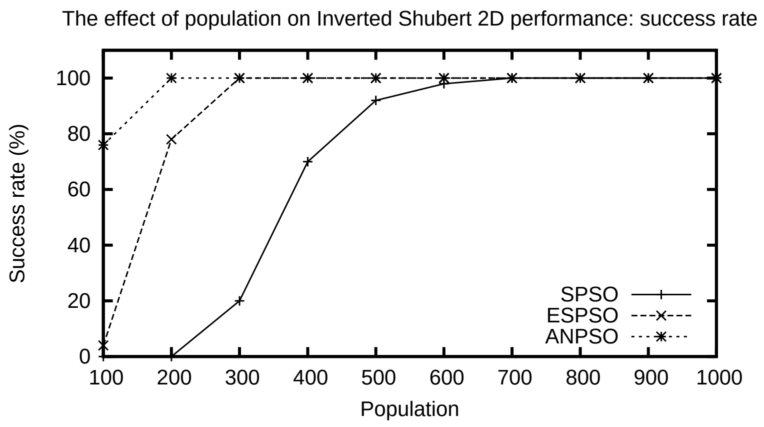


Figure 5.18: The number of successful runs with different population sizes for Inverted Shubert 2D.

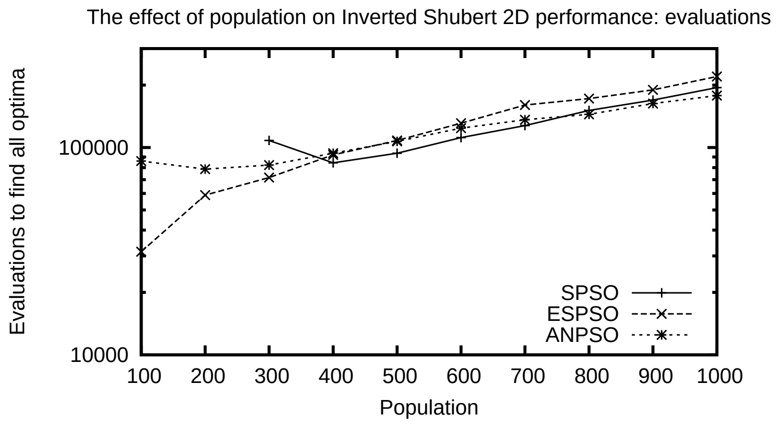


Figure 5.19: The number of evaluations required for different population sizes on Inverted Shubert 2D.

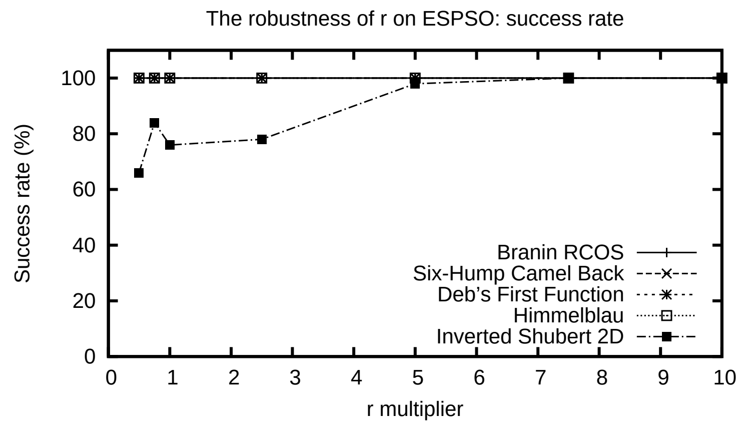


Figure 5.20: The number of successful ESPSO runs with different values of r. To aid comparison between test functions, r has been shown as a multiple of SPSO's optimal r for each test problem.

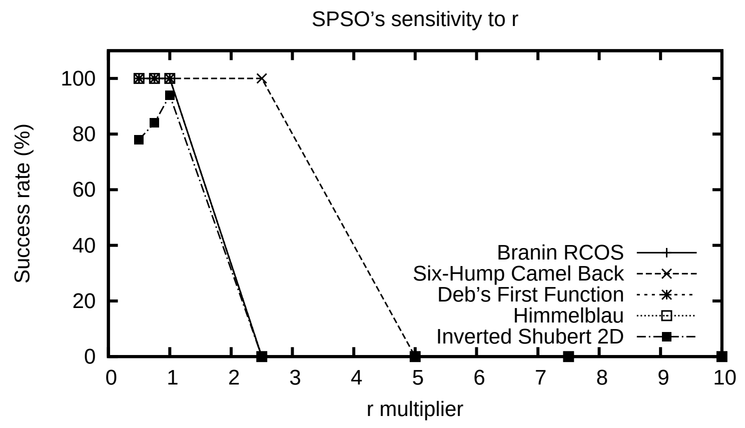


Figure 5.21: The number of successful SPSO runs with different values of r.

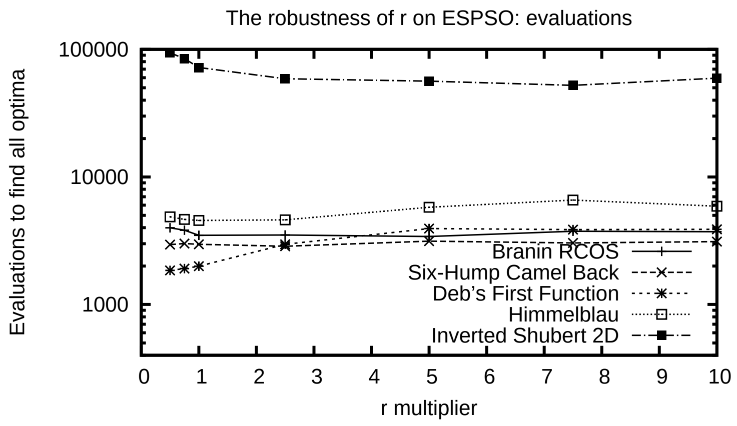


Figure 5.22: The number of evaluations required for ESPSO to locate all optima with different values of r.

using the r parameter at all on these functions. In this case it can be thought of as acting similarly to a sequential speciation algorithm as described in Section 2.7.7, although it is still inherently parallel. As r is still a problem-dependant parameter, even if only limited in effect, we are unable to recommend a value that will be optimal on a majority of test functions. If the distance between optima is unknown though, we recommend setting it to a large value as being the safest option. Larger values of r encourage more cooperation between particles, allowing individuals to explore the most promising areas.

Performance on low-dimensional functions

Table 5.1 compares the relative performance of the three algorithms. For ESPSO, r has been set to 2.5 times the optimal value for SPSO. It can be seen that on most functions, SPSO requires fewer evaluations to locate all of the optima. On Inverted Shubert 2D SPSO required more evaluations than the other two algorithms, however it was also using almost twice the population size to improve reliability.

Table 5.1: Number of evaluations required to find all global peaks (mean and standard error).

Function	Pop.	SPSO	ESPSO	ANPSO
Branin	30	2983.38 (±105.9)	3526.34 (±103.71)	4910.88 (±468.03)
Camel Back	30	2520.12 (±104.1)	2821.06 (±100.7)	2820.8 (±132.62)
Deb's First	30	1942.22 (±71.09)	2784.32 (±137.44)	7221.02 (±786.13)
Himmelblau	30	3677.06 (±80.87)	4651.12 (±196.92)	11954.96 (±1333.97)
Inv. Shubert 2D	300[a]	93858.7 (±1660.18)	71664.34 (±1739.9)	82217.08 (±1708.1)

[a]SPSO was tested with a population of 500 to improve reliability

Overall SPSO still gives the best performance, however this is only after substantial tuning. The r parameter has been carefully selected for each function after much trial and error. The largest difference is with Deb's First function, where ANPSO required approximately four times as many evaluations. This gives us a maximum of 3 trial runs of 1700 evaluations to find the ideal value of r, a formidable task. When we consider cases where the actual number of optima are unknown it becomes even harder; after our 3 trial runs we are still left with the question, "is r small enough?". If all our guesses are too large then it is likely we will miss some of the optima.

Performance on higher-dimensional functions

Figure 5.23 shows the success rates of ANPSO, ESPSO and SPSO on the Humps function for each number of dimensions. The performance of SPSO and ESPSO are similar, except for when there are only 5 decision variables. In this case the species radius parameter is slightly too high, meaning that neighbouring species are competing for particles. This further illustrates the importance of reducing the reliance on speciation parameters, which ESPSO and ANPSO are designed to do. While the extra dimensions did not reduce ESPSO's reliability, it did for ANPSO. Further research would be required to determine why this is the case and how the algorithm can be modified to better handle these environments.

The average number of evaluations required by each algorithm is shown in Figure 5.24. Overall ANPSO showed the worst scalability, again showing its weakness on this particular problem. The point plotted for ANPSO with 10 dimensions is not statistically significant as there were only 3 successful runs.

SPSO achieved its optimal result when there were seven dimensions. As the number of dimensions increases, so does the average distance between each optimum. With seven dimensions nearby optima were far enough apart that they were no longer competing against each other for particles. ESPSO also displayed the same trend to a lesser degree, as would be expected as its r parameter is more robust. The overhead in terms of evaluations of using ESPSO over SPSO was relatively constant for all the dimensions tested.

Overall

Overall our experiments show that when the effort and evaluations required to correctly tune r is considered, the performance of ESPSO and ANPSO is very competitive to SPSO. While SPSO generally required the fewest evaluations, this is only after tuning of its r parameter by a knowledgeable user. ANPSO is able to determine the most appropriate value as it runs, reducing the user's burden. ESPSO still uses r, although provided the value is large enough it will still perform well. If the user is at all unsure they can even set it to infinity with only a slight performance penalty. In the next part of this chapter, we will use SPSO, ESPSO and ANPSO to evaluate the performance metric proposed in Chapter 4.

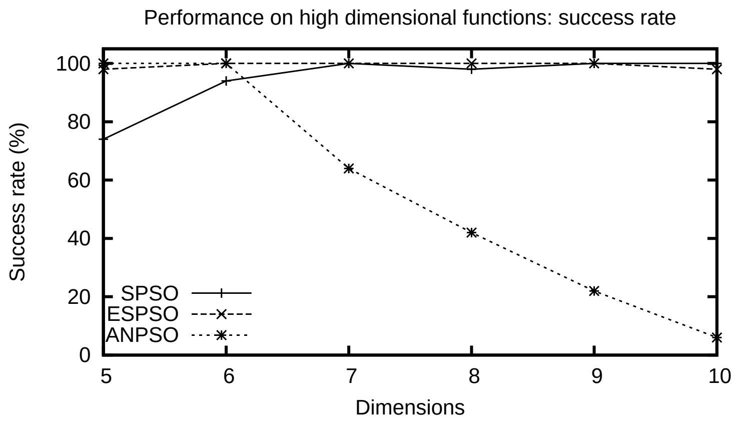


Figure 5.23: The success rates of SPSO and ESPSO were largely unaffected by dimensionality. The reliability of ANPSO decreases rapidly as the number of dimensions is increased.

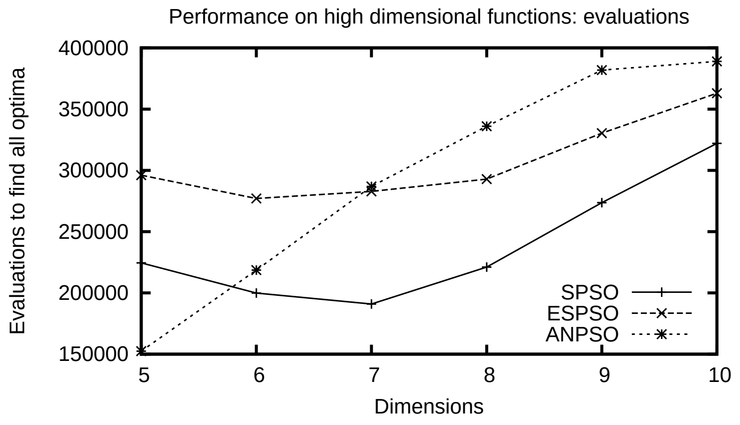


Figure 5.24: The number of evaluations scaled roughly linearly for all of the algorithms. SPSO's performance was poor below 7 dimensions because of the competition for particles by neighbouring species.

5.2 Informedness of performance metrics for dynamic environments

In Chapter 4 we proposed a new performance metric that is the complement of peak cover: BKPE. Since a metric is only useful if it can tell us something we did not already know, in this section we will be testing the informativeness of these two metrics. First we will determine whether they are reliable indicators of overall performance. We will then discuss how these two metrics can be used to indicate how a given algorithm can be improved.

5.2.1 Experimental setup

We are primarily interested in showing peak cover and BKPE's informativeness by correlating them with an existing accepted metric, offline error (see Section 2.3.4). To do that, we will be comparing the offline error of each run for a number of different algorithms to the distance from the point as it would be plotted on Figure 5.25, to the bottom right of the graph. The BKPE is normalised so that all values are in the range [0, 1]. The actual calculation for this is shown in Equation (5.4). We have called the resulting variable PB, standing for "**P**eak cover and **B**KPE".

$$PB = \sqrt{(1 - peak\ cover)^2 + \left(\frac{BKPE}{max(BKPE)}\right)^2} \tag{5.4}$$

where PB is a combined measure of the peak cover and BKPE performances and $max(BKPE)$ is the maximum BKPE of any of the runs.

In addition, we will also analyse the relative performance of four PSO variants and the effect of adding a local search to the algorithms. The PSO models we will use are standard

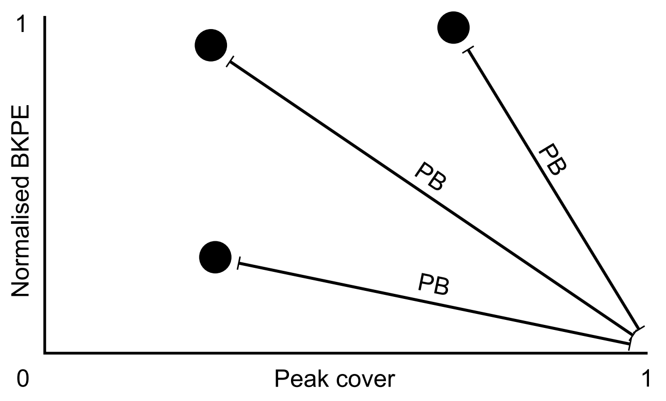


Figure 5.25: Algorithms that get a low BKPE and high peak cover will have a low offline error, thus offline error is related to the distance from the bottom right of the graph.

constriction PSO with a von Neumann neighbourhood model (see Section 2.4.9), SPSO, ESPSO and ANPSO. Each PSO is also tested in combination with GCPSO (see Section 2.7.9) to enhance convergence. All parameters have been set to the values recommended in Section 5.1.2. An r value of 30 has been used for SPSO and ESPSO. 50 runs with an initial population size of 100 were used for all PSO models and each run was stopped after 500000 evaluations. For SPSO, we limited each species to 6 particles – any excess particles were randomly distributed in the decision space. To allow better comparison with ESPSO and encourage more stable species, we used the particle's personal best as the species seed.

For our experiments we will be using the Moving Peaks dynamic test function generator, as described in Section 2.3.5. In order to function correctly, the PSO needs to be informed when the landscape changes. To detect these changes, at the end of each timestep we check the fitness of the top 5 species seeds. If any of the fitnesses differ from the recorded value, each particle's personal best location is set to their current location, as described in Section 2.5.2.

Unless stated otherwise, all Moving Peaks parameters have been set to the values for

Table 5.2: Moving Peaks Scenario 2 parameters

Parameter	Setting
Random seed	1
Dimensions	5
Peaks	10
Minimum peak height	30
Maximum peak height	70
Standard peak height	50
Minimum peak width	1
Maximum peak width	12
Standard peak width	0
Coordinate range	[0, 100]
Peak movement severity	1
Peak height severity	7
Peak width severity	1
Basis function	None
Movement correlation λ	0
Peak movement interval	5000 evaluations
Peak shape	Conic
Change stepsize	Constant

scenario 2, listed in Table 5.2. The parameters of most interest to us are the number of dimensions, the number of peaks and the movement severity. The number of dimensions and peaks are quite intuitive. The movement severity controls how far the peak moves each change. A severity of 1 means the peak moves one unit from its previous location. The movement correlation λ controls whether the movement is in the same direction each time or not. When this parameter is set to 0 the peaks will move in a random direction; there is no correlation between successive peak movements.

5.2.2 Results and analysis

This section will first discuss the correlation of the metrics with offline error and how it changes with the number of peaks. We will then analyse the differences in the performance of the tested PSO variants. Finally we will use the metrics to determine how the algorithms' performance can be improved.

Correlation with offline error

Figure 5.26 shows the Pearson correlation between PB and offline error when there are 10 peaks. The scattered points around 1.0 on the PB axis is caused by the von Neumann runs. Almost invariably this topology caused the PSO to converge on only one of the 10 peaks, giving a peak cover of 0.1. As the BKPE was small relative to the other algorithms, almost all of the von Neumann runs had a PB of around 1.0. In this case, the offline error was almost entirely determined by movements of the peak the algorithm converged on – it had very little chance to locate alternate peaks once it had converged. In our tests there was almost no correlation between offline error and PB for the von Neumann runs: 0.05 in the case of the 10-peak tests. To avoid skewing our results, we have excluded these runs when discussing overall correlations.

Figure 5.27 shows the correlation between BKPE, peak cover and offline error for the speciated algorithms as the number of peaks varies. A high correlation indicates that the given metric is a large determining factor in the offline error. To make the graph easier to read, we have plotted the correlation of $1 - \textit{peak cover}$ and made the peaks axis logarithmic. A coefficient of -1 or 1 indicates perfect correlation and 0 indicates none. A negative correlation indicates that as the first variable increases the second generally decreases.

When there are more than 10 peaks, the correlation of PB with offline error is higher than both peak cover and BKPE. When there are fewer than 50 peaks the correlation is very

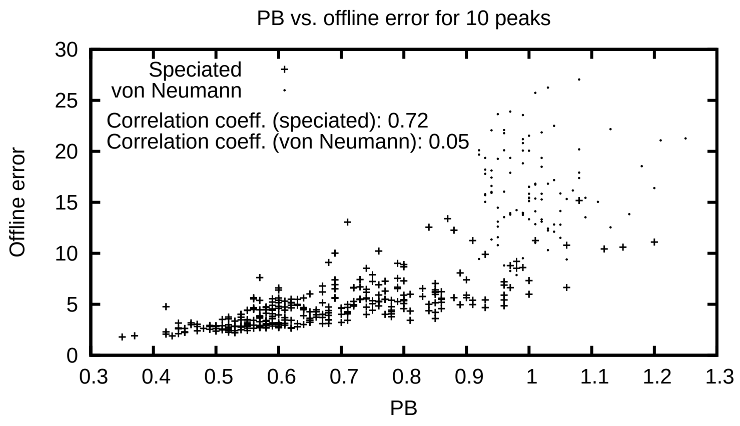


Figure 5.26: The correlation between PB and offline error. The scattered points are the runs using the von Neumann topology; these all had a small BKPE and a peak cover of 0.1. Across all the speciated runs, the correlation coefficient between offline error and PB was 0.72, showing these are strongly correlated.

strong, showing that these metrics are the main factors of offline error. Above 50 peaks the correlation declines; it becomes increasingly important to track the most promising peaks instead of every peak possible.

The peak cover is most important when there are only a few peaks. Since the number of peaks relative to the population size is low, it is possible to keep a significant number of individuals on each peak. This allows the algorithm to respond more quickly to peak movements and changes. The comparatively low BKPE correlation shows that while convergence speed does play a role here, the offline error is mainly determined by how many of the peaks were tracked.

As the number of peaks increases, tracking too many becomes a disadvantage. Since the algorithms only have a finite number of individuals available, there is a risk of spreading the particles too thin. Figure 5.28 shows that as the number of peaks increases, runs where the peak cover was high tended to also have a high BKPE. This indicates that maintaining

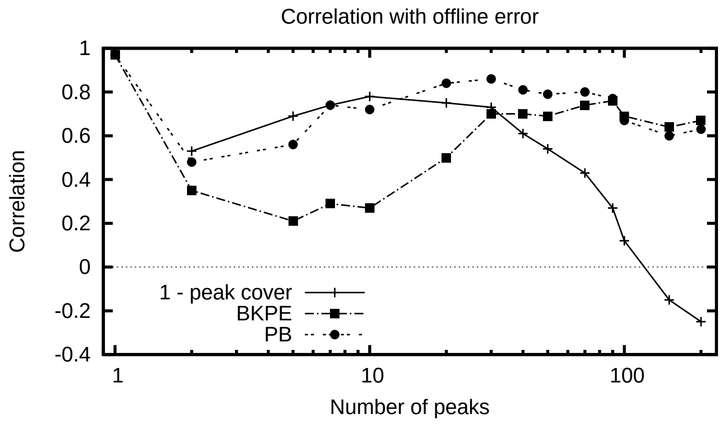


Figure 5.27: The correlation of each metric for different numbers of peaks.

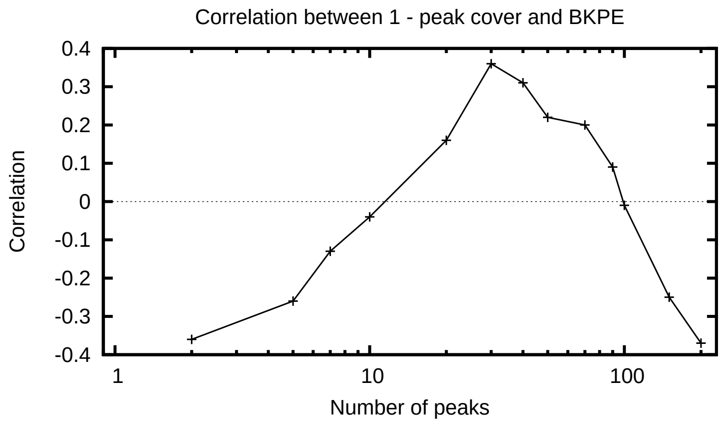


Figure 5.28: The correlation between 1 - peak cover and BKPE. When there are many peaks, maintaining a high peak cover results in a low convergence speed because there are fewer individuals on each peak.

individuals on too many peaks reduces the algorithm's ability to quickly converge, offsetting any diversity advantage. The correlation between offline error and BKPE indicates quick convergence is important, even if it is not on the correct peak. Although the algorithm may not be tracking the currently-optimal peak, it is likely to be tracking another with similar fitness. As long as the algorithm accurately knows the location of the alternative peaks, the error penalty should be relatively small.

Algorithm performance

Figures 5.29 and 5.30 show the performance of the algorithms on the 10 and 100 peak problems respectively. The most obvious difference between the two is that the peak cover is much lower with 100 peaks; the algorithms no longer have enough individuals to represent all of the peaks – they must choose the most promising ones.

To allow easy comparison, the symbol used for the GCPSO-enhanced algorithms is a filled version of the respective unenhanced version. As can be seen in both figures, GCPSO increased the convergence speed slightly for all of the algorithms.

The number of peaks did not change the algorithms' relative BKPE and peak cover rankings. Von Neumann has an extremely low BKPE because almost all individuals converged on the same peak. The extra particles gives it a far lower BKPE than the speciated algorithms, at the cost of only being able to track a single peak and thus a very low peak cover. It should be repeated that von Neumann's offline error performance was very poor as it was difficult for it to switch peaks.

ANPSO and SPSO showed very similar BKPE, although for the 10 peak problem SPSO

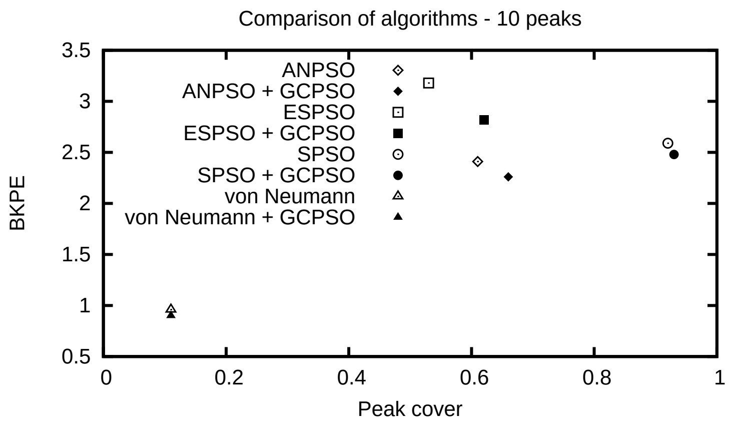


Figure 5.29: Comparative performance with 10 peaks. Bottom left: The non GCPSO von Neumann point is obscured by the GCPSO-enhanced result as the BKPE and peak cover are almost identical.

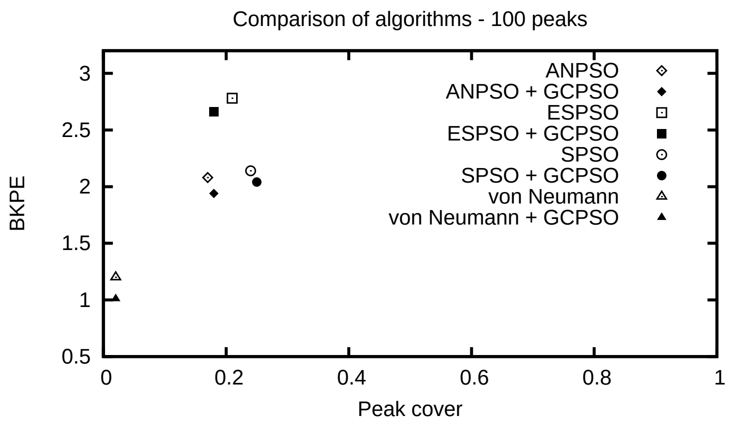


Figure 5.30: Comparative performance with 100 peaks.

Table 5.3: Algorithm performance with 10 peaks (mean and standard error).

Algorithm	**BKPE**	**Peak cover**	**Offline error**
ANPSO	2.41 (±0.07)	0.61 (±0.02)	5.28 (±0.22)
ANPSO + GCPSO	2.26 (±0.06)	0.66 (±0.01)	4.59 (±0.16)
ESPSO	3.18 (±0.10)	0.53 (±0.03)	7.39 (±0.37)
ESPSO + GCPSO	2.82 (±0.11)	0.62 (±0.02)	5.68 (±0.25)
SPSO	2.59 (±0.06)	0.92 (±0.01)	3.09 (±0.08)
SPSO + GCPSO	2.48 (±0.06)	0.93 (±0.01)	2.86 (±0.06)
von Neumann	0.96 (±0.05)	0.11 (±0.00)	16.63 (±0.55)
von Neumann + GCPSO	0.90 (±0.04)	0.11 (±0.00)	16.07 (±0.61)

had a significantly better peak cover. For the 100 peak problem both the BKPE and peak cover are almost identical (Figure 5.30), resulting in very similar offline errors.

ESPSO's comparatively poor BKPE and peak coverage can be expected as in general it is slower to locate optima than SPSO, as discussed in Section 5.1.2. It was designed to reduce the sensitivity to SPSO's radius parameter and thus require less tuning. Since the optimal radius value is already known and we are only interested in tracking the highest peak within each species area, the enhancements in ESPSO do not translate to improved performance as measured by these metrics.

Finally, in Table 5.3 we compare the offline error of the algorithms when tested on the 10 peak problem. SPSO combined with GCPSO was by far the best performing algorithm, while von Neumann was the worst. The correlation between offline error and the presented metrics is evidenced by these results.

Table 5.4: Correlation with offline error for 10 peaks.

	ANPSO	**ESPSO**	**SPSO**	**von Neumann**
1 - Peak cover	0.53	0.68	0.35	0.08
BKPE	0.26	0.01	0.71	0.03

Improving the algorithms

Each algorithm has its own strengths and weaknesses. Some algorithms such as SPSO are very good at maintaining diversity, others are able to converge very quickly. The overall effectiveness of an algorithm is still measured by offline error; as discussed above increasing peak coverage or decreasing BKPE may not necessarily be advantageous.

To find how an arbitrary algorithm can be improved, we determine the correlation of peak cover and BKPE with offline error. Table 5.4 lists these correlations for the tested algorithms. A high correlation between one of the metrics and offline error indicates the algorithm's final performance is dependant on that aspect. A low correlation suggests improving the algorithm in this area is not likely to impact on overall performance – the effort is better spent elsewhere.

It should be noted that the numbers in Table 5.4 do not indicate performance. Algorithms that have similar correlation for a particular metric may have wildly different performance in that area. For example, both ESPSO and von Neumann have almost no correlation between BKPE and offline error, however ESPSO's mean BKPE is more than 3 times as large as von Neumann's (Table 5.3).

The most striking feature of Table 5.4 is the very low correlations for the von Neumann runs. Since both metrics have a low correlation, we know the offline error must be determined by other factors. From observing individual runs we know in this case it is the movements and average height of the algorithm's chosen peak.

ANPSO showed a reasonable correlation for both metrics, indicating that improving either the diversity or convergence speed should improve offline error. Interestingly, SPSO and ESPSO were opposite each other in this regard: for SPSO increasing the peak coverage has a negligible effect, most likely because peak cover was already very high. ESPSO on the other hand benefited greatly from increased peak coverage. ANPSO has a similar peak cover but far less correlation with offline error, possibly indicating that ESPSO is becoming trapped on low value peaks. In any case, the high correlation shows that adding a diversity preservation mechanism to this algorithm would likely improve offline error.

The lack of correlation between BKPE and offline error for ESPSO shows that while the BKPE is very high, improving it is not likely to have much impact on the offline error score. By contrast SPSO had a strong correlation for BKPE – increasing convergence speed should improve the overall performance.

5.3 Summary

In this chapter we have tested the performance and reliability of our two new speciated algorithms, ESPSO and ANPSO. We were able to show that the parameters of both algorithms were robust, to the point where we could recommend values that should provide good performance on a range of optimisation problems. While overall both algorithms required more evaluations to find all the optima than SPSO, this does not take into account the extra time and evaluations required to tune SPSO's r parameter. Doing so for any given problem, especially one the user does not have much information about is very difficult. With these two algorithms we have been able to overcome SPSO's major limitation.

In the second part of the chapter we determined that both peak cover and BKPE are reliable indicators of overall performance as measured by offline error. These metrics specifically measure the exploration and exploitation performance of an algorithm respectively, giving a more detailed picture as to an algorithm's strengths and weaknesses.

By correlating an algorithm's performance as measured by these algorithms with offline error over a number of runs, we are able to determine which factor is most important for that particular algorithm. This allows the user to measure an algorithm's performance and determine the most effective way to improve it; if improving the convergence speed or diversity of an algorithm will not greatly improve its overall performance, it is usually not worth spending time researching in that area. In the next chapter we will use this information to greatly improve SPSO's performance on the Moving Peaks test problem.

Chapter 6

Improving local convergence

Local search methods, discussed in Section 2.4.2, are known for their extremely fast convergence, however they are also highly susceptible to becoming trapped in the first optimum they find. Many real world problems are far too complex to be solvable by these methods.

Evolutionary Algorithms (EAs) and Particle Swarm Optimisation (PSO) have also proved to be effective search strategies. They are able to converge on an optimum even when the catchment area occupies only a small portion of the search space. These algorithms are far less likely to become trapped in a sub-optimal peak than local search, especially when combined with a diversification measure.

Combining local search with an EA or PSO provides the best of both worlds. We gain the robustness of the population-based algorithms and the local search's convergence speed [Michalewicz and Fogel, 2000]. This hybrid approach is quite common, for example in [Vesterstrom et al., 2002], [Mahinthakumar and Sayeed, 2005] and [Yin et al., 2006]. One drawback with this approach is that using the local search requires extra fitness evaluations to be performed; when considered over the entire optimisation process, these evaluations can be very costly.

In this chapter we present a method that provides the advantages of a hybrid algorithm, without requiring any extra fitness evaluations. Rather than performing a local search, we use the existing information to compute an n-dimensional surface that best fits the peak[1]. We then attempt to calculate the highest point of this surface. Provided that the local features of the fitness landscape roughly match our surface, the local optimum should be very close to the computed highest point. This allows us to very quickly hone in on the actual maximum point – with each successive attempt we know more and more about the landscape, improving our estimation further.

This method differs from the meta-model techniques presented in Section 2.4.10 in that rather than using the computed model to decide which points do not need to be evaluated, we actively estimate the location of the optimum. We are essentially adding a heuristic to the base algorithm to increase the convergence speed.

6.1 Using regression to locate optima

Around each peak there is a catchment area or funnel, described in Section 3.3.1. Within this area, fitness generally improves as you get closer to the peak. If we can model the overall shape of the peak while ignoring the local features, we can calculate the highest point of that shape. Assuming that our model is reasonably accurate, the top of our shape should be close to the optimum.

[1]A paper on this technique was published by this author in 2007 [Bird and Li, 2007b].

The regression maintains a separate memory to the base algorithm, storing only the best locations and their fitnesses. If the base algorithm's memory was used, points would only remain known as long as there is an individual there. The regression needs to know the locations of the fittest points, regardless of the population's current state.

A minimum number of points is needed in order to calculate the regression – below this there will be more than one shape that fits the data. For example to uniquely define a 1 dimensional parabola, 3 points are required. Without the third point, there are an infinite number of polynomial functions that will fit the data. As the algorithm continues sampling the fitness landscape, the regression may keep some excess points to help reduce the effect of any local landscape features. The number of excess points e is a tunable, although robust, parameter.

By performing a linear least-squares regression on the known points and their fitnesses, we are able to estimate the peak's shape. From the regression we obtain a set of equations, one for each decision variable, that defines the shape that best fits our known points. Although more complex and flexible equations can be used if desired, for simplicity and efficiency we have used quadratic equations to represent the shape. This results in a set of simultaneous equations in the form of Equation (6.1) to be solved for $a_1, a_2, ..., a_{2n}$ and c, where n is the number of decision variables.

$$f(x_1, x_2, ..., x_n) = a_1x_1^2 + a_2x_1 + a_3x_2^2 + a_4x_2 + \ldots + a_{(2n-1)}x_n^2 + a_{(2n)}x_n + c \tag{6.1}$$

In matrix form, the simultaneous equations look like:

$$\mathbf{A} = \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_m \end{bmatrix} \qquad \mathbf{B} = \begin{bmatrix} x_{1,1}^2 & x_{1,1} & x_{1,2}^2 & x_{1,2} & \dots & x_{1,n}^2 & x_{1,n} & 1 \\ x_{2,1}^2 & x_{2,1} & x_{2,2}^2 & x_{2,2} & \dots & x_{2,n}^2 & x_{2,n} & 1 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ x_{m,1}^2 & x_{m,1} & x_{m,2}^2 & x_{m,2} & \dots & x_{m,n}^2 & x_{m,n} & 1 \end{bmatrix} \qquad \mathbf{C} = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_{2n} \\ c \end{bmatrix}$$

where there is an equation for each of the m known points.

To solve the simultaneous equations, we manipulate the matrices as in Equation (6.2):

$$\begin{aligned} \mathbf{A} &= \mathbf{BC} \\ \mathbf{B}^+\mathbf{A} &= \mathbf{B}^+\mathbf{BC} \\ \mathbf{B}^+\mathbf{A} &= \mathbf{C} \end{aligned} \tag{6.2}$$

$\mathbf{B}^+$ is the pseudoinverse of $\mathbf{B}$ [Ben-Israel and Greville, 1974]. If we only use the minimum number of points, $\mathbf{B}$ will be square and we can use the inverse $\mathbf{B}^{-1}$ instead. The coefficients that make our equation best match the known points are found by computing $\mathbf{C}$. We then find the turning point for the equation in each dimension $i = [1, n]$ by taking the partial derivative, as in Equation (6.3):

$$\frac{\partial}{\partial x_i} f(x_1, x_2, \dots, x_n) = 2x_i a_{(2i-1)} + a_{(2i)} \tag{6.3}$$

The turning point in dimension i is where $\frac{\partial}{\partial x_i} f(x_1, x_2, \ldots, x_n) = 0$. To find out whether it is a maximum or minimum point, we take the second derivative. When using quadratic equations, we can simply look at the sign of $a_{(2i-1)}$; a negative number indicates that it is a maximum. If this is a maximisation problem and one of the equations has only a minimum turning point, we abort the regression and wait for better data. Similarly, if it is a minimisation problem we abort if any of the equations has no minimum turning point.

The global maximum point of the shape will be at the location of the turning point in each decision variable. Even though we were able to compute a maximum, we still need to check that it is valid. If the points do not give a good representation of the peak, for example they are all on one side, the regression will not be accurate. If the computed point is outside the expected area, or even the entire decision space, it is discarded. We will try the regression again when we have more data.

To test the calculated position, we replace the least-fit individual with a new individual at the shape's highest point. This avoids using an extra evaluation, and it is unlikely that the individual's next movement would have contributed much to the search. If the regression was successful, the new point will be used to further refine the shape when it is next performed, hopefully improving the fitness still further. When using this technique with dynamic environments, we clear the memory whenever a peak movement is detected. This prevents the regression from being performed on stale data.

The main cost of this method is in performing the matrix inversion. Assuming the minimum number of points are used, this has a complexity of $O(n^3)$. As n depends only on the number of decision variables and complexity of the equations used, the cost is usually

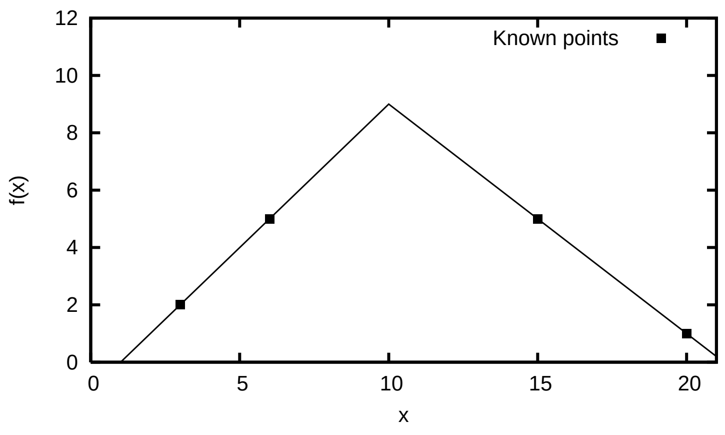


Figure 6.1: Trying to find the highest point of the peak. We currently know the fitnesses of 4 points: 3, 6, 15 and 20. The right side of the peak is less steep than the left.

quite low. The CPU cost can be further reduced by only performing the regression at certain intervals or only for the most promising peaks. In many environments, fitness evaluations are the most expensive aspect. The regression's minimal CPU overhead is usually far outweighed by the number of evaluations saved.

As an example we will try to solve a 1-dimensional triangular function, as shown in Figure 6.1. Currently we know the fitnesses of 4 points:

$$f(3) = 2$$

$$f(6) = 5$$

$$f(15) = 5$$

$$f(20) = 1$$

We place these values into **B** and **C**:

$$\begin{bmatrix} 2 \\ 5 \\ 5 \\ 1 \end{bmatrix} = \begin{bmatrix} 3^2 & 3 & 1 \\ 6^2 & 6 & 1 \\ 15^2 & 15 & 1 \\ 20^2 & 20 & 1 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ c \end{bmatrix}$$

Multiplying both sides by $\mathbf{B}^+$ gives:

$$\begin{bmatrix} 0.01 & -0.01 & -0.01 & 0.01 \\ -0.24 & 0.12 & 0.30 & -0.17 \\ 1.46 & 0.02 & -1.01 & 0.54 \end{bmatrix} \begin{bmatrix} 2 \\ 5 \\ 5 \\ 1 \end{bmatrix} = \begin{bmatrix} a_1 \\ a_2 \\ c \end{bmatrix}$$

$$\begin{bmatrix} -0.065 \\ 1.430 \\ -1.516 \end{bmatrix} = \begin{bmatrix} a_1 \\ a_2 \\ c \end{bmatrix}$$

This gives us the best-fitting quadratic curve, Equation (6.4).

$$f(x) = -0.065x^2 + 1.430x - 1.516 \tag{6.4}$$

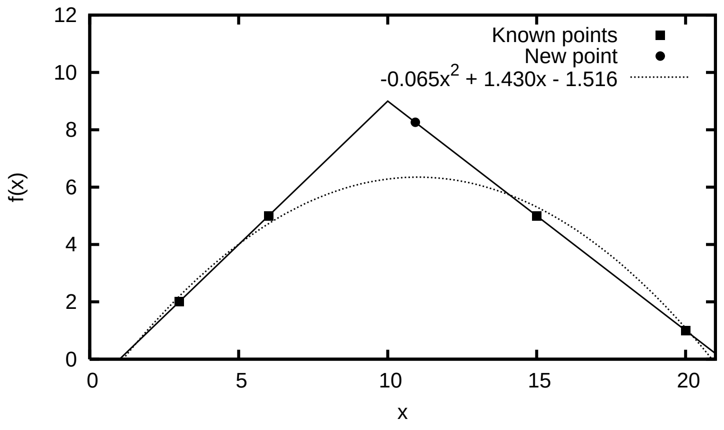


Figure 6.2: The regression curve has a maximum at $x = 10.926$, considerably closer to the peak than any of the previously known points.

To find the turning point, we differentiate it:

$$\frac{df(x)}{dx} = (-0.065)2x + 1.430 \tag{6.5}$$

Solving Equation (6.5) gives a turning point of $x = 10.926$. We know this point is the maximum because the x^2 coefficient is negative. The fitness at $x = 10.926$ is 8.2592. As can be seen from Figure 6.2 this is not the location of the actual peak, however it is considerably closer than any of the points known so far.

6.2 Advantages

Using a regression can improve the convergence speed quite considerably. Because the regression function is only a quadratic, it does not try to map the local features. This makes it very resistant to overfitting, that is where the algorithm becomes so tuned to the local features that it misses the overall trend, decreasing its accuracy.

This method combines well with speciation; since it makes the assumption it is only looking at a single peak, and speciated algorithms are designed to focus on each peak individually, the regression becomes less likely to be "confused" by points on other peaks. The regression provides the greatest relative benefit when the population is small, as is often the case within a species.

Unlike most other local search methods, using the regression does not require extra evaluations. Instead we replace the least-fit individual with the point computed by the regression. This individual is chosen as it is the least likely to contribute to the search during the next timestep. The only real cost of the regression is the CPU time and memory needed to compute it, however in most situations this is quite minimal.

6.3 Disadvantages

For most fitness landscapes, the regression needs to have a point on each side of the optimum for each dimension. Without this, it is trying to estimate a turning point that exists outside the area it has information about. Unless the overall shape of the peak's area is quadratic, it is likely the estimated peak will not accurately reflect the peak's true location. In each dimension there are two possibilities: the true optimum is either closer to, or further from, the population than the estimated optimum. If it is further, it is likely that even though the regression was not accurate, it is still in the correct direction and may induce the population to explore further in that direction. In cases where the regression overestimates the distance to the optimum, often the estimated location lies far outside the decision space. If this happens we discard the information and try again later.

6.4 Summary

Existing stochastic optimisation algorithms generally do not use the gradient information between multiple points to guide their search. Instead they attempt to converge on the best known solution and gradually climb the peak towards the optimum. By analysing the relative positions and fitnesses of previous candidate solutions, we can actively estimate the peak's true location. This technique is not limited to speciated algorithms or particle swarms, and can potentially be applied to any numeric optimisation algorithm.

The cost of the regression is minimal: only a small amount of CPU time and memory is required. The benefit is quite large; we have potentially saved many evaluations if the regression suggests a point close to the true optimum. For many fitness landscapes the benefits of using this technique far outweigh the costs, allowing algorithms to locate optima using far fewer evaluations than would otherwise be possible. In the next chapter we will discuss the experiments and results used to support this claim.

Chapter 7

The effectiveness of regression

In Chapter 5 we concluded that of all the PSO models tested, SPSO + GCPSO (described in Section 2.7.9) had the best performance on Moving Peaks. In Section 5.2.2 we were also able to show that if we can improve its convergence speed, we should be able to achieve a significant reduction in the offline error. To determine whether performing the regression is effective, we compared the performance of this combination with and without the regression heuristic. For simplicity and to better show the effect of the regression, we did not use the anticonvergence measure discussed in Section 3.3.1.

To determine whether performing the regression is effective, we will compare the performance of SPSO + GCPSO with and without the regression. For the rest of this chapter, we will use SPSO to mean SPSO + GCPSO, and rSPSO to mean SPSO + GCPSO + regression. The regression has been implemented so as to discard any calculated solutions that are outside the species boundary, as described in Section 6.1.

For the purposes of the regression, we consider each species to be an individual subpopulation with its own memory. This means that for every timestep, there is a regression run for each species.

7.1 Experimental setup

The regression will be tested on both static and dynamic test functions; we will describe our procedure below. For all of the tests, each species is limited to $P_{max} = 6$ particles; any excess particles are reinitialised elsewhere in the decision space. The success and failure thresholds for GCPSO have been set to the values recommended by van den Bergh and Engelbrecht [2002], that is $s_c = 15$ and $f_c = 5$. Unless otherwise stated, the regression stores a maximum of $e = 10$ excess points. Each experiment was performed 50 times and the results have been averaged.

7.1.1 Static Functions

To test general performance in a static environment, we have used the same test functions as in Chapter 5. As with the experiments in Chapter 5, a run is only considered successful if the algorithm locates all of the optima to within a fitness of $\epsilon = 0.00001$ within 2000 timesteps. For the functions other than Inverted Shubert 2D, a population size of 50 has been used. To reliably solve Inverted Shubert 2D, SPSO requires at least 500 particles. With the exception of Deb's First function, all of these are two dimensional functions.

7.1.2 Moving Peaks

As in Chapter 5, we used Moving Peaks to test the regression's performance in dynamic environments. To determine the robustness of the regression under different circumstances, we tested:

- The number of peaks between 1 and 200
- The severity of each peak movement, between 0 and 6
- The number of decision variables between 5 and 10

We also wanted to see whether the regression was still effective with other peak shapes. The standard conic shape used by Moving Peaks is produced by Equation (7.1):

$$f(x) = h - w\sqrt{\sum_{d=1}^{D} (x_d - p_d)^2} \tag{7.1}$$

Where w and h specify the width and height of the peak respectively, x_d is the location of the point in dimension d and p_d is the tip of the peak in dimension d.

In order to create functions of various shapes including asymmetric peaks, we have extended this equation in several ways:

- The relationship between height and distance from the peak's centre in dimension d is now controlled by α_d. $\alpha_d = 1$ will produce the standard conic shape used by the standard Moving Peaks scenarios. Using $\alpha_d > 1$ will produce a mound shape, with steepness increasing as the α_d gets bigger. Setting $\alpha_d \in (0, 1)$ produces a spike shape, as depicted in Figure 7.1.

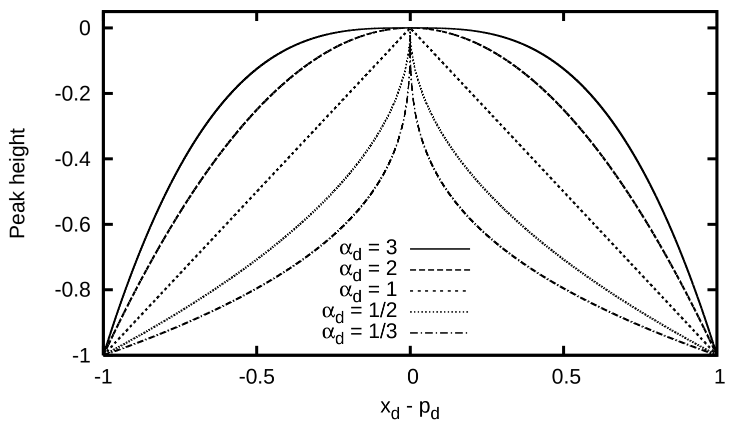


Figure 7.1: Five peaks have been overlaid, showing how α_d determines the peak's shape.

- To create asymmetric peaks, for each decision variable we divide the peak into 2 halves, left and right. The value of α_d used for the left and right halves are denoted α_{d0} and α_{d1} respectively. Figure 7.2 shows an asymmetric peak.

- Local optima have been added by superimposing a cosine wave over the peak. The amplitude and frequency of the wave are specified by β_d and γ_d respectively. All γ values used are in radians. Figure 7.3 shows a cosine wave superimposed on Figure 7.2. By adjusting β_d and γ_d we can specify the number and severity of local peaks in variable d.

The new peak function is defined in Equation (7.2). The standard conic peak shape can be achieved by setting $\alpha_d = 1, \beta_d = 0, \gamma_d = 0$ for all d.

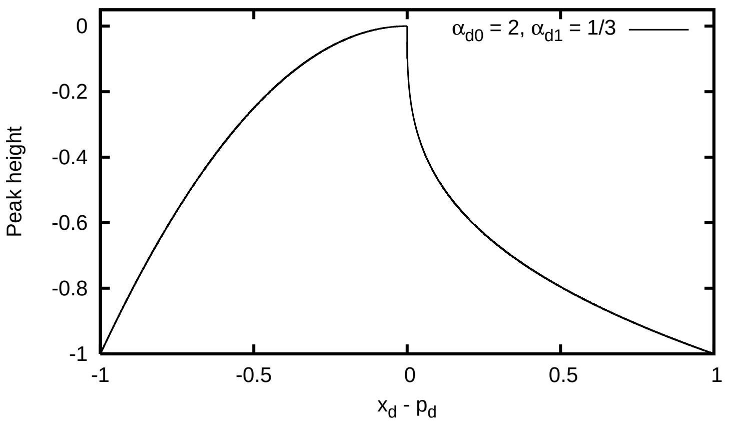


Figure 7.2: An example of an asymmetric peak. The values of α_d are 2 and $\frac{1}{3}$ for the left and right halves respectively.

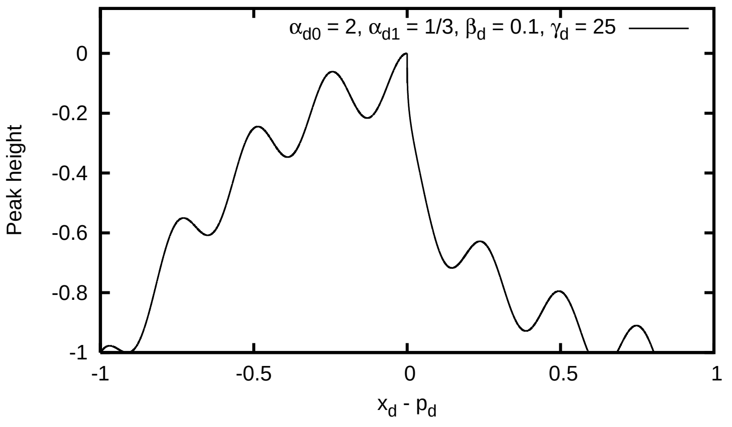


Figure 7.3: Superimposing the cosine wave over the peak in Figure 7.2

$$\alpha_{d0}, \alpha_{d1} \in (0, \infty), \qquad \beta_d, \gamma_d \in [0, \infty)$$

$$\alpha_d = \begin{cases} \alpha_{d0}, & \text{if } x_d < p_d \\ \alpha_{d1}, & \text{otherwise} \end{cases}$$

$$u_d = \beta_d \left(cos \left[\gamma_d \left(x_d - p_d \right) \right] - 1 \right)$$

$$f(x) = h - w \sqrt{\sum_{d=1}^{D} \left(|x_d - p_d|^{\alpha_d} + u_d \right)^2} \tag{7.2}$$

To determine the regression's performance on varying peak types, the following tests were carried out:

- Symmetric peaks where all α_{d0} and α_{d1} values are equal.

  $\alpha_d \in \left\{\frac{1}{3}, \frac{1}{2}, 1, 2, 3\right\}, \quad \beta = \gamma = 0.$

- Asymmetric peaks where all α_d values are randomly generated.

  $\alpha_{d0}, \alpha_{d1} \in [\frac{1}{3}, 3], \quad \beta = \gamma = 0.$

- Conic peaks with a superimposed wave:

  $\alpha_d = 1, \quad Max_\beta \in \{2, 4, 6, 8, 10\}, \quad Max_\gamma \in \{20, 40, 60, 80, 100\}.$

- Asymmetric peaks with a superimposed wave:

  $\alpha_{d0}, \alpha_{d1} \in [\frac{1}{3}, 3], \quad Max_\beta \in \{2, 4, 6, 8, 10\}, \quad Max_\gamma \in \{20, 40, 60, 80, 100\}.$

Max_β indicates that each peak's values for β_d are randomly chosen in the range $[0, Max_\beta]$. Max_γ indicates the same thing for γ.

We also tested the regression's sensitivity to e, both on the standard scenario 2 problem and with the modified peak function, using $\alpha_d \in [\frac{1}{3}, 3], \quad Max_\beta = 10, \quad Max_\gamma = 100$. The latter is designed to show whether using a larger e value improves performance on functions with many local optima.

Finally we compared our results to mQSO, one of the best performing algorithms on Moving Peaks scenario 2. We have compared our results to the $10\,(5 + 5^q)$ configuration that Blackwell showed to be optimal on this problem. All experiments were run with 100 particles and with SPSO's r parameter set to 30. These were the settings used in [Li et al., 2006] and Chapter 5.

7.2 Results

This section is divided into two main parts. We will first look at the regression's performance on static functions. Secondly we will analyse its behaviour on the Moving Peaks test suite.

7.2.1 Static Functions

Even with static environments, it is still very important to reduce the number of evaluations. Each evaluation costs CPU time, often well in excess of the time used by the optimiser itself. In this part we will report on the regression's performance on static problems, both in low dimensional and high dimensional environments. The third part of this section will investigate how the number of excess points affects performance.

Table 7.1: Regression performance on low dimensional static functions. Mean, standard error and improvement over SPSO are shown.

Function	rSPSO	SPSO	Improvement
Branin RCOS	6254.54 (±298.59)	9552.86 (±216.98)	35%
Six-Hump Camel Back	1489.63 (±53.90)	8934.58 (±304.97)	83%
Deb's First Function	5306.60 (±225.23)	7613.16 (±271.40)	30%
Himmelblau	4963.74 (±277.75)	11069.68 (±299.91)	55%
Inverted Shubert 2D	50511.46 (±794.94)	164360.00 (±2912.89)	69%

Low dimensional landscapes

Table 7.1 shows that using the regression dramatically increased performance on all of the low dimensional functions we tested. The largest improvement was on Branin RCOS, where the number of evaluations required was reduced by more than 80%. Even on Deb's First Function, the regression still reduced the number of evaluations by nearly a third. This is a significant improvement.

Inverted Shubert 2D (depicted in Figure 3.1) was by far the hardest 2 dimensional function we tested, normally requiring 160000 evaluations for SPSO to locate all of the optima. Adding the regression reduced this to just 50000. We suspect that one of the reasons the regression performed so well is that the peak tips resemble a parabola, allowing it to be accurately modelled.

High dimensional landscapes

The number of dimensions does not appear to affect the regression's effectiveness. Table 7.2 shows that using the regression reduced the number of evaluations by about 30%. This becomes especially significant as the number of dimensions increases; for the 20 dimensional

Table 7.2: Regression performance on the high dimensional Humps function.

Dimensions	rSPSO	SPSO	Improvement
5	191902.50 (±5353.82)	292472.57 (±12638.88)	34%
10	256795.42 (±2506.12)	356665.10 (±5146.87)	28%
15	277200.14 (±2038.94)	404640.00 (±1161.45)	31%
20	297978.00 (±1427.11)	462036.00 (±1216.41)	36%

function using the regression saved over 160000 evaluations.

Sensitivity to e

As Figure 7.4 shows, the best performance was obtained by setting e to between 10 and 30. Although it still beats SPSO, the regression performed relatively poorly when excess points were not kept. The Humps function's peaks are conic, making them difficult to model when only using the minimum number of points. The excess points help to define the surface better, improving the regression's guess.

Using too many points also decreased performance. The regression does not perform any weighting – it tries to match all of the points regardless of their fitness. By storing too much data, we allow the memory to become polluted with distant and poor quality points. Instead of just modelling the tip, the peak's overall shape is matched. We are then unable to accurately determine the tip's location as our model is not specific to that area.

We recommend that e be set to 10 for all problems. This value represents a good tradeoff; it provides excellent performance without creating too much CPU or memory overhead. We have also tested this on the Moving Peaks problem and found the same ideal value. This is discussed in the next section.

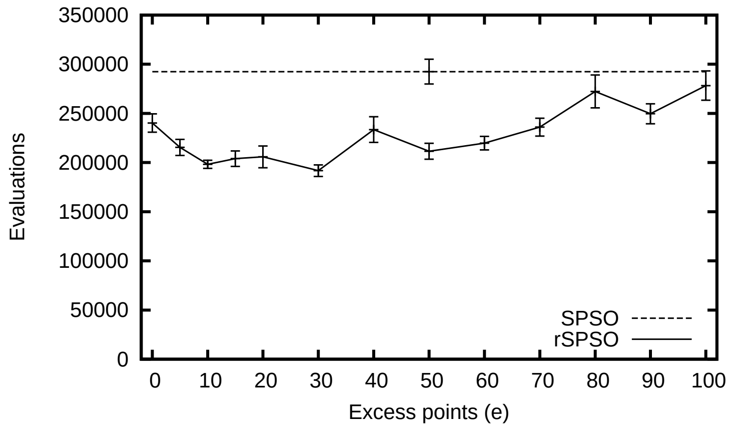


Figure 7.4: Number of evaluations needed for Humps 5D with different numbers of excess points. The vertical bars show one standard error.

7.2.2 Moving Peaks

Reducing the number of evaluations is critical when working with dynamic environments. If the environment is changing every 2 minutes, an algorithm that takes 3 minutes to find an adequate solution is useless. By adding the regression we are able to significantly reduce the number of evaluations needed; this section details the performance on Moving Peaks under different situations.

Increasing convergence speed

Figure 7.5 compares the current error over time for SPSO and rSPSO. Current error is the difference in fitness between the best known point and the global optimum. Offline error is calculated by averaging the current error over an entire run.

In scenario 2, the peaks move every 5000 evaluations, as can be seen by the upwards jumps in the graphs. The regression is inactive for at least the first 200 evaluations after a peak movement. As the population size is 100, this represents only two timesteps; improvements

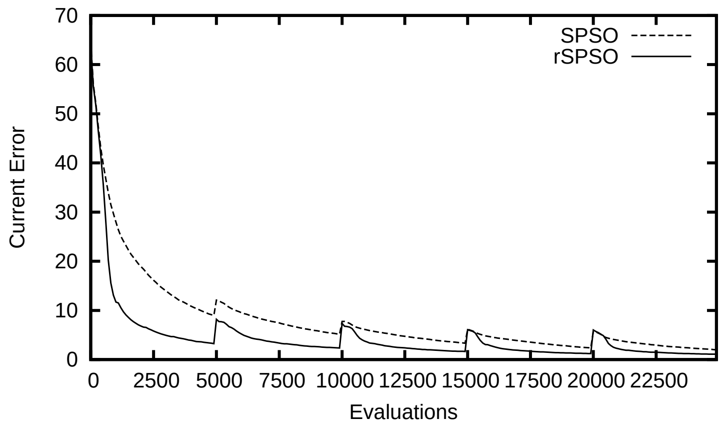


Figure 7.5: Current error over time on Moving Peaks scenario 2, showing the effect of adding the regression.

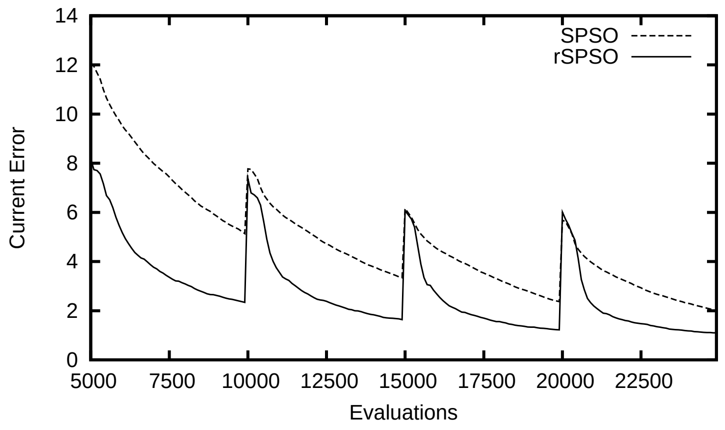


Figure 7.6: Current error over time after the first peak movement. This is the same data as shown in Figure 7.5.

here are mostly due to the fortunate placement and existing momentum of the particles. The regression cannot be used yet because there are insufficient points for it to be computed. For a 5 dimensional function, 11 points are needed. Since each species is limited to 6 particles, at least two timesteps are needed to collect the required data.

Figure 7.6 is the same as Figure 7.5, but without the first 5000 evaluations. This gives a better indication of the regression's performance helping to track the peaks. After a peak movement the PSO requires 100 evaluations to reevaluate the particles, one for each individual. Thus the indicated error immediately after a change is not representative of the individuals' true fitnesses.

At 300 evaluations after a peak movement the regression's effect becomes obvious. The curve for rSPSO drops quickly as the algorithm hones in on the optimum. After a landscape change, a normal PSO must wait for the particles to accelerate towards the new peak, then wait for them to converge again once the peak has been located. The combination of GCPSO and the regression reduces the time spent doing this: the regression moves the worst particle close to the peak while GCPSO's rules set the velocity to 0 and force it to explore the local area.

At 1200 evaluations after a peak movement rSPSO has already achieved the same error as SPSO does after 4900 evaluations. Even at this point, the curve has a fairly steep gradient; substantial improvements are being made each timestep. The graph plateaus around 3000 evaluations; the error at this point is very small. The curve does not converge to 0 because SPSO is usually unable to maintain species on all of the optima (see Section 5.2.2); the residual error is from the times that the algorithm is not tracking the highest peak.

Please note that to ensure a smooth curve for Figs. 7.5 and 7.6, we have performed 1000 runs for each of the algorithms. All other results presented in this thesis are based on experiments of 50 runs.

Number of peaks

Adding the regression reduced offline error by between 1 and 1.5, as shown in Figure 7.7. The settings of SPSO ($r = 30, P_{max} = 6$) are optimised for 10 peaks, which explains the sweet spot at that point. Below this, there are too many particles for the number of peaks. Since only 6 particles are allowed on each peak, when there are too few peaks most of the particles are continually reinitialised. This wastes evaluations, resulting in a larger offline error.

At the other end of the graph it becomes impossible for the algorithm to track all of the peaks. Having neither enough particles nor a small enough r value, the algorithm must rely on the particles to jump between peaks, hopefully to the globally optimal one. The increased error is caused by the times the algorithm is unable to discover the best peak.

Peak movement severity

The peak movement severity controls how far the peak moves each time. Larger severities increase the time needed to re-find the peak. The further the peaks move, the faster the particles will be travelling when they reach it. In a standard PSO they will then take longer to slow down and reconverge. Using the regression in combination with GCPSO helps this process; whenever the regression's guess is successful the worst particle will become

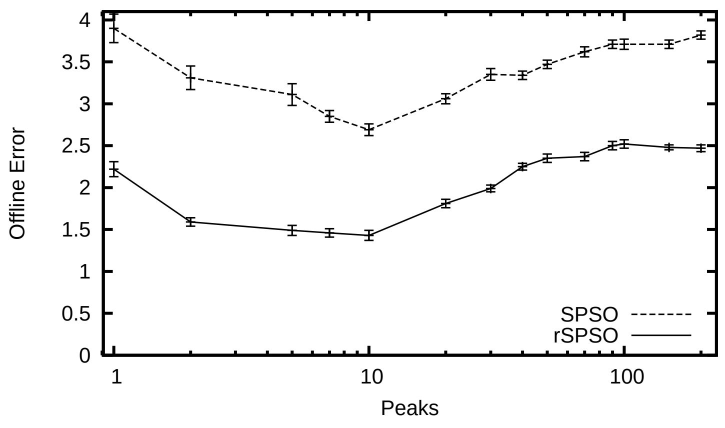


Figure 7.7: Adding the regression substantially reduces offline error.

the species seed, as it has been moved to a point better than any of its neighbours. Since the species seed follows the GCPSO movement rules, it immediately loses the momentum it previously had. As this happens to successive particles, the average velocity is quickly reduced, allowing the particles to reconverge.

Figure 7.8 shows how SPSO's error increases linearly with severity. rSPSO's error also increases, but at a much lower rate. At a severity of 0 the peaks do not move at all, they only change height. In this situation the error achieved depends almost entirely on being able to track all of the peaks. As the peaks are not moving, once the optima have been initially located the regression is not needed to track them, thus the performances of SPSO and rSPSO are very similar.

Dimensionality

As with the results shown in Table 7.2 for the Humps function, the performance improvement provided by the regression is fairly constant. The error increases linearly as the number of

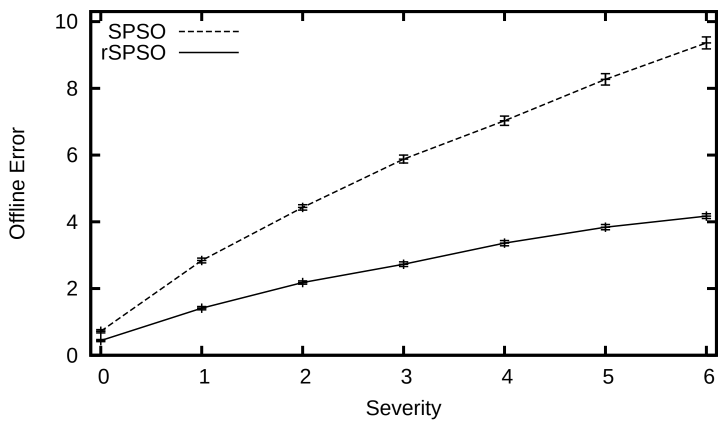


Figure 7.8: The benefit of the regression increases with severity.

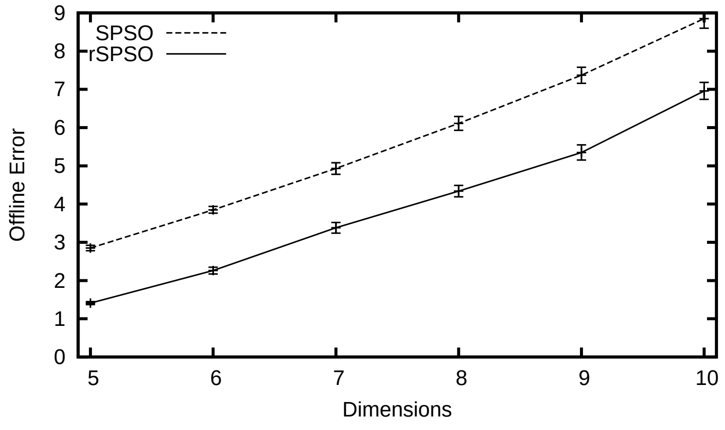


Figure 7.9: The regression reduced the offline error by around 1.5 for all of the dimension values tested.

dimensions increases, however the difference between rSPSO and SPSO remains about 1.5, as shown in Figure 7.9.

Peak shape

For this experiment, all of the α_d values have been set identically, making the shapes tested symmetrical. A one dimensional peak for each value of α_d is depicted in Figure 7.1.

The most obvious feature of Figure 7.10 is that outside the range $\alpha_d \in [1, 2]$ there is a

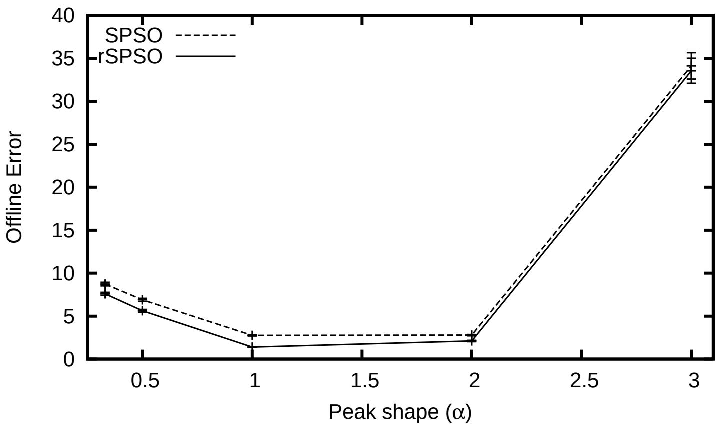


Figure 7.10: Offline error for different peak shapes.

large increase in offline error.

For $\alpha_d > 2$, the areas away from the optima are exceptionally steep. In this case, performance depends almost entirely on how quickly the algorithm can locate the general area of the tip, rather than its exact location. This is evidenced by the large variance at $\alpha_d = 3$. As the regression is only effective once the peak's general area has been found, we can see the relative performance improvement is weaker compared with the smaller α_d values.

When $\alpha_d < 1$, the penalty for being far away from the peak's tip is relatively small, however to achieve a small error it is extremely important to precisely locate the optimum. The difference between a point 0.1 units away from the peak and 0.2 units away can be substantial. Again, using the regression reduced the offline error by about 1. This is quite impressive as the actual peak shape is the opposite of the regression's model – the fitness landscape does not fit a parabola at all. This result suggests the parabolic model works well even on difficult peak shapes, and that more elaborate models are generally unnecessary.

Table 7.3: Regression performance on asymmetric peaks.

α	rSPSO	SPSO	Imp.
$[0.33, 3]$	1.83 (±0.08)	2.52 (±0.08)	27%
$[0.4, 2.5]$	1.64 (±0.07)	2.40 (±0.07)	32%
$[0.5, 2]$	1.66 (±0.06)	2.73 (±0.09)	39%
$[0.67, 1.5]$	1.51 (±0.05)	2.85 (±0.06)	47%
$[1, 1]$	1.40 (±0.05)	2.75 (±0.06)	49%

Further testing would be required to confirm this though.

Asymmetric peaks

In the real world, many fitness landscapes have asymmetrical peaks. The peak may be very steep on one or more sides, or be at the edge of the feasible region. For these experiments, we have used random peak shapes. The α_d values for each side of every peak in each dimension are chosen randomly within a specified range. For example, in a run with only two peaks, the following values may be chosen:

$$\alpha_{00} = 0.40, \alpha_{01} = 0.94, \alpha_{10} = 2.64, \alpha_{11} = 0.58$$

As Table 7.3 shows, the regression is most effective when the range of α_d is small. The regression works better on peaks that are roughly symmetrical as they more closely match the parabolic shape used. However even when highly asymmetrical peaks are created, the regression still achieved a 30% improvement over SPSO. This shows again that the regression's guesses are still accurate enough to aid the optimisation, even when it cannot closely model the peak.

Adding local optima

By comparing Figs. 7.11 and 7.12 we can see that the wave's amplitude had a much greater effect on performance than its frequency. The flatness of Figure 7.12 suggests that the swarm is able to jump from peak to peak with relative ease. The amplitude's effect is far greater because each new candidate solution is just as likely to be at the bottom of the wave as the top. On average each new point will be halfway down a wave, increasing the overall error incurred.

The local optima decrease the accuracy of the regression's model, reducing its effectiveness. Even so, it still managed to reduce the offline error by around 1 in all of the runs.

Asymmetric peaks with local optima

This is the most challenging landscape for the PSO; we are creating peaks that look similar to the one shown in Figure 7.3. In all of the experiments the regression reduced the offline error by between 1.5 and 2. As can be seen by comparing Figs. 7.13 and 7.14 with the results for the asymmetric peaks and local optima individually, the local optima are the primary cause of the large offline error – the asymmetric peaks are not a significant component. As would be expected, the results here are very similar to the results for the local optima tests.

Sensitivity to e

On scenario 2 the value of e does not greatly affect the regression's performance. Low, nonzero values provided slightly better results for the same reason as before, by allowing the regression to concentrate on the best information. In all cases the regression outperformed

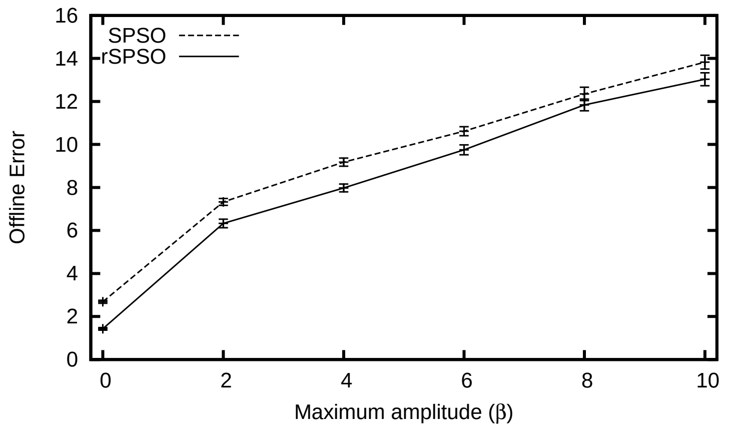


Figure 7.11: Performance on symmetric peaks for different values of Max_β. $\alpha_d = 1, \quad Max_\gamma = 100$

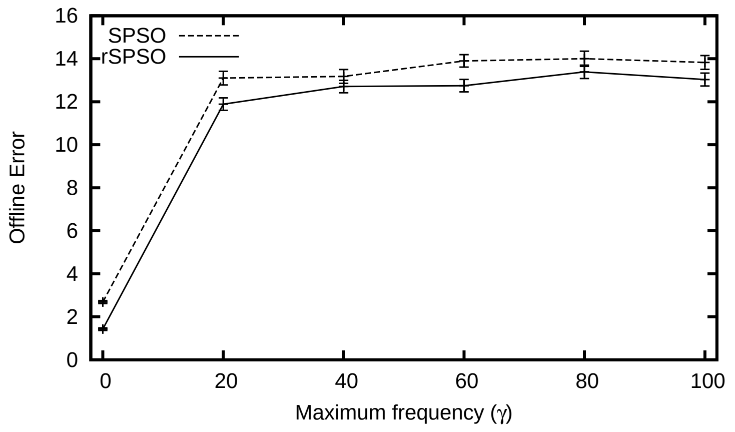


Figure 7.12: Performance on symmetric peaks for different values of Max_γ. $\alpha_d = 1, \quad Max_\beta = 10$

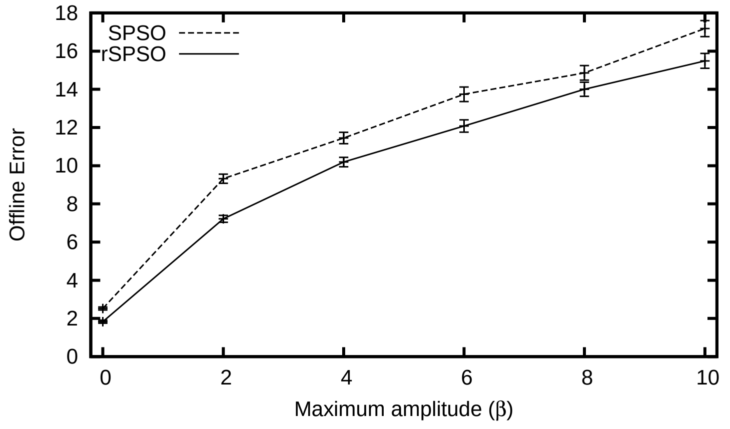


Figure 7.13: Performance on asymmetric peaks for different values of Max_β. $\alpha_d \in [0.33, 3], \quad Max_\gamma = 100$

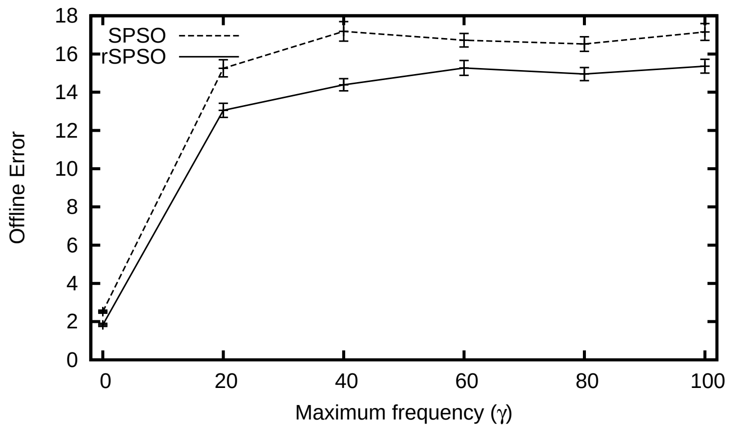


Figure 7.14: Performance on asymmetric peaks for different values of Max_γ. $\alpha_d \in [0.33, 3], \quad Max_\beta = 10$

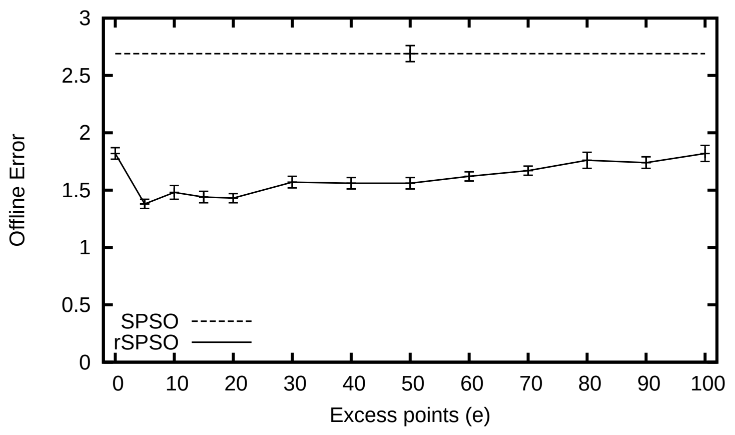


Figure 7.15: Offline error for Moving Peaks Scenario 2 for different values of e.

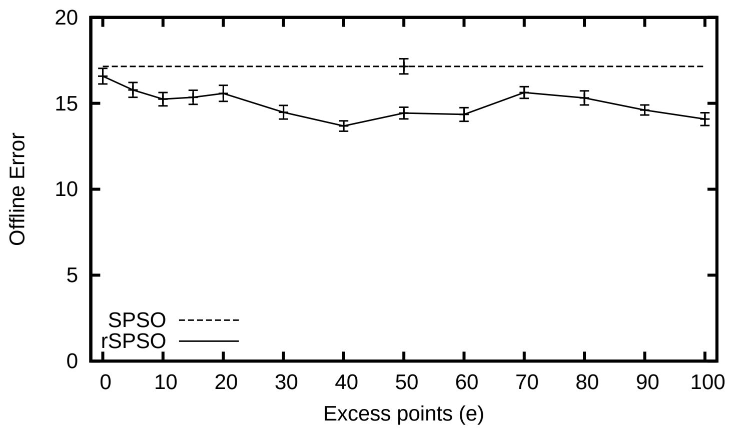


Figure 7.16: Offline error for asymmetric peaks with waves for different values of e.
$\alpha_d \in [0.33, 3], \quad Max_\beta = 10, \quad Max_\gamma = 100$

SPSO by a significant margin, as shown in Figure 7.15.

When optimising the most complex function, Moving Peaks with $\alpha_d \in [0.33, 3], \beta = 10, \gamma = 100$, the value of e played a larger role in performance (Figure 7.16). As would be expected, increasing e slightly helped the regression ignore the local optima. Values larger than 40 gave no extra advantage however, showing that even for difficult problems only a few excess points are needed.

Table 7.4: Comparing against mQSO: Severity

s	mQSO (AC)	rSPSO	SPSO
0	1.18 (±0.07)	0.44 (±0.03)	0.72 (±0.05)
1	1.75 (±0.06)	1.41 (±0.05)	2.84 (±0.07)
2	2.40 (±0.06)	2.18 (±0.05)	4.43 (±0.08)
3	3.00 (±0.06)	2.73 (±0.07)	5.88 (±0.12)
4	3.59 (±0.10)	3.36 (±0.08)	7.03 (±0.14)
5	4.24 (±0.10)	3.84 (±0.08)	8.27 (±0.17)
6	4.79 (±0.10)	4.17 (±0.07)	9.36 (±0.18)

Comparing to mQSO

In Table 7.4 we compare the performance against mQSO for differing movement severities. As can be seen, rSPSO exceeds mQSO's performance for all of the severities tested. This is even more impressive considering that for most of the experiments SPSO had far worse performance than mQSO.

Table 7.5 compares mQSO's performance against both SPSO and rSPSO for differing numbers of peaks. It can be seen that rSPSO is highly competitive with mQSO; offering better performance for all but the 100 and 200 peak runs.

The regression can be added to most numerical optimisation algorithms; it is highly likely that it could by used to improve mQSO's performance even further.

7.3 Summary

In this chapter we have tested the regression technique described in Chapter 6. Although we have chosen to test using SPSO, this technique can be applied to most numeric optimisation algorithms.

Table 7.5: Comparing against mQSO: Number of peaks

Peaks	mQSO	rSPSO	SPSO
1	5.07 (±0.17)	2.22 (±0.09)	3.90 (±0.17)
2	3.47 (±0.23)	1.59 (±0.05)	3.31 (±0.14)
5	1.81 (±0.07)	1.49 (±0.06)	3.11 (±0.13)
7	1.77 (±0.07)	1.46 (±0.05)	2.85 (±0.07)
10	1.80 (±0.06)	1.43 (±0.06)	2.69 (±0.07)
20	2.42 (±0.07)	1.81 (±0.05)	3.06 (±0.06)
30	2.48 (±0.07)	1.99 (±0.04)	3.35 (±0.07)
40	2.55 (±0.07)	2.25 (±0.04)	3.34 (±0.05)
50	2.50 (±0.06)	2.35 (±0.05)	3.47 (±0.05)
100	2.36 (±0.04)	2.52 (±0.05)	3.71 (±0.06)
200	2.26 (±0.03)	2.47 (±0.04)	3.82 (±0.05)

Our results show that adding the regression significantly improved SPSO's performance on a range of fitness landscapes. By using the existing population members it does not require any additional evaluations, only a modest amount of memory and CPU time depending on the number of decision variables and excess points.

As a future research direction it may be worthwhile exploring other deterministic techniques that could be combined with an EA or PSO. Currently much of the available information is thrown away as the population moves each generation. By retaining and analysing this data, as is done by Estimation of Distribution Algorithms [Larranaga and Lozano, 2002] for example, it is likely that further improvements in performance can be gained.

Chapter 8

Conclusion

In this thesis we discussed several issues of importance to the field of real-valued parameter optimisation. Many fitness landscapes are multimodal in nature. These environments are challenging for optimisation algorithms; it is highly desirable to locate multiple good solutions where they exist, however most algorithms that can differentiate multiple optima require the user to set a parameter in order to do so. We have proposed two new algorithms that remove this burden from the user. These algorithms are able to effectively locate all of the global optima on the problems we tested, without requiring tuning for each one. This makes them far more usable where tuning is not possible or practical, for example in many real-world problems.

One of the most difficult challenges is coping with changes in multimodal dynamic environments. Simple diversity preservation mechanisms may not provide adequate performance where there are multiple potential solutions that could lead to a future global optimum.

Speciation is a technique that allows the algorithm to locate and track multiple optima simultaneously. It does so by maintaining a small population on a number of optima. This has three major benefits:

- It provides the user with a choice of solutions.
- Population diversity is maintained; reducing the risk of premature convergence.
- The algorithm can make decisions for optima as well as individuals, for example by choosing to kill an under-performing species.

In addition, speciation provides a distinct advantage in environments where a current local optimum may become the future global optimum; algorithms that maintain populations on local peaks are much better positioned to react to changes.

This thesis introduced several techniques which can be used to improve performance on challenging static and dynamic multimodal environments. This chapter provides our conclusions about the methods presented, how they relate to the goals of our research and discusses possible directions for future research.

8.1 Addressing our research objectives

In Chapter 1 we outlined several research objectives to be addressed by this thesis. These related to the general question of how we can increase robustness and performance of speciated algorithms.

Performance has many aspects, not least of which is how many evaluations an algorithm needs to locate an environment's optima. We sought to improve performance in a number of ways, including designing algorithms that do not require the user to tune parameters and developing ways of profiling an existing algorithm's performance. Finally, we applied the knowledge we gained to vastly improve the convergence speed and overall peak-tracking ability of an existing speciated algorithm. In the subsections below we will detail how we have addressed the objectives presented in the first chapter.

8.1.1 Reducing the reliance on speciation parameters

In Chapter 3 we developed two algorithms that significantly alleviate the problem of setting user-specified speciation parameters. Most existing speciated algorithms require the user to specify a similarity threshold. This controls how close two individuals have to be in the decision space to be considered as representing the same peak. This parameter is difficult to set; the ideal value depends on the problem being optimised and the algorithm in question.

One of the proposed algorithms, ESPSO, modifies SPSO to allow it to differentiate nearby optima. It does this by isolating the particles that have converged within a certain area from the main population, allowing other particles to explore the area and locate any remaining peaks. In this way ESPSO removes the upper limit on r, SPSO's similarity threshold. ESPSO works effectively even if r is set to very large values, reducing its burden on the user. If he or she knows nothing about a new optimisation problem, they can set it to infinity or some other very large value and be confident that ESPSO will still perform well.

The other proposed algorithm, ANPSO, removes r from the user's control entirely. It instead uses population statistics to adaptively determine the value. ANPSO calculates r to be the average distance between each particle and its closest neighbour. This allows the algorithm to be completely invariant to the size of the decision space; large decision spaces will cause the particles to be futher apart than smaller ones, causing r to be larger as well.

Both ESPSO and ANPSO use two parameters s and P_{max}. s controls the trigger point of the algorithms, that is how many steps it waits before it forms a species or subpopulation. P_{max} determines the size of the species, preventing the algorithms from spending all their resources on a single optimum. In Chapter 5 we showed that these two parameters were robust across all the functions we tested against.

We were also able to show competitive results against SPSO, once the required tuning of r for this algorithm was taken into account. At worst, ANPSO took 4 times as many evaluations as SPSO to locate all of the optima, giving us a maximum of 3 trial runs for SPSO to refine r. It would be very difficult to determine the correct r value for a given problem within 3 tries; at the very least we would be left with the question, "did we try a low-enough value?". If all of the values we tried are too high it is likely we will not locate some of the optima.

8.1.2 Designing a metric to exclusively measure convergence speed

In Chapter 4 we presented a new metric for dynamic environments, BKPE. This metric exclusively measures the convergence speed of an algorithm by only considering the optimum it is concentrating on. By ignoring the other optima, especially the ones the algorithm has not discovered or has forgotten, we avoid measuring the algorithm's diversity characteristics.

By concentrating on only one aspect of an algorithm's performance rather than attempting to provide a general overview, we are able to obtain a better understanding of its strengths and weaknesses. When combined with the Branke's peak cover metric, we are able to compare both the exploration and exploitation aspects of an algorithm. An algorithm that is able to converge quickly while maintaining good population diversity is likely to perform well overall. This was borne out in Chapter 5 by correlating the combination of these metrics with offline error, an accepted metric that measures overall performance.

8.1.3 Performing a comparative study of several algorithms

Of a standard PSO using a von Neumann topology, SPSO, ESPSO and ANPSO, with and without GCPSO, we found that SPSO had the best overall performance on the Moving Peaks test suite. In general, adding GCPSO did not improve the convergence speed of the algorithms much. This suggests that the population of each species was usually large enough not to need GCPSO's enhancements.

One of the main things our data showed was that while von Neumann was able to locate a peak very quickly, its lack of diversity led to very poor performance. These experiments demonstrated the natural tendency of a PSO to converge on a single peak, and the consequences of doing so in a dynamic environment. Without any form of speciation, the entire population converged on a single point. The algorithm was unable to adapt when a different peak became the global optimum. This meant its performance was almost entirely determined by the height of its chosen peak throughout the run.

8.1.4 Using performance metrics to profile an algorithm

Having metrics that measure specific aspects of an algorithm's performance also allows us to see how best to improve it. By correlating the BKPE and peak cover of a set of runs with offline error, we can tell to what degree convergence speed and diversity affect overall performance for that particular algorithm.

We showed that SPSO had a high correlation between its offline error and BKPE, showing that the most effective way to increase overall performance is to increase its local convergence speed. Interestingly ESPSO was the opposite. There was almost no correlation between its convergence speed and overall performance, instead its effectiveness over a run was determined by how many of the peaks it was able to track.

8.1.5 Improving local convergence speed

From the overall performance of SPSO and the strong correlation between its BKPE and offline error scores, we could see that it would benefit the most from enhancing the local convergence speed. In Chapter 6 we developed a mechanism to achieve this. For each species we use linear regression to fit a surface to the best known points. We then calculate the surface's maximum point in each dimension to determine the estimated peak location. If the regression was successful the point should be within the decision space and we move the species' worst individual there. Otherwise we discard it and try again once we have better data.

Assuming the regression is successful, our estimated peak is often much closer to the true peak than any of our known points. As the regressions from successive timesteps increase the fitness of our candidate solutions, we can very quickly hone in on the true optimum.

For simplicity we have used a quadratic surface for our regression, although any derivable mathematical function can potentially be used. More complex functions could better map to the known data, at the cost of requiring more points to perform the regression. With the more flexible functions there is always a risk of overfitting. This is where our surface closely fits the actual points but does not follow their general trend, leading to inaccurate estimates of the true optimum.

We combined our regression technique with SPSO to create rSPSO, as suggested in the results shown in Chapter 5. By using the regression we saw a marked improvement in offline error. In particular the offline error was significantly lower than mQSO, currently the best performing PSO algorithm on Moving Peaks. This was shown in Chapter 7. As the regression technique is not tied to any particular algorithm, we expect that it will be able to provide benefit to many other methods as well.

8.1.6 Extending the Moving Peaks benchmark

To better test our regression heuristic we needed a more challenging peak shape. In Chapter 7 we extended the Moving Peaks benchmark suite to allow multimodal peaks of arbitrary shapes. Our extensions to Moving Peaks enabled us to show the heuristic was effective even when the fitness landscape did not match its assumptions.

These extensions also allow researchers more flexibility when testing and reporting their results. They are no longer limited to the conic peak shape; instead they can now create mounds and spikes of varying severity as well as combinations of the two. In addition, they can create many local optima by superimposing cosine waves of arbitrary amplitude and frequency. These extensions are backwards-compatible, meaning that the original conic peak shapes can be easily achieved by setting the parameters to specified values.

Hopefully the more flexible peak shapes will spur more research into optimising dynamic environments where the peaks do not monotonically increase towards the optimum. The resulting algorithms are more likely to be effective on real-world problems, which are rarely as simple as the landscapes generated by the original Moving Peaks benchmark.

8.2 Future directions

There are many future research directions from this thesis. Generally these are to improve the techniques presented herein, although we also have suggestions for further studies that are not a direct extension of the methods discussed in the preceding chapters.

8.2.1 Speciation algorithms

While both ESPSO and ANPSO perform very well, there are still a few ways the algorithms could be improved. On the Moving Peaks test suite we found that ESPSO's creation of subswarms tended to cause a population explosion. Many excess particles were being created which waste evaluations. It is likely that if we can reduce this problem we will see better performance as measured by offline error.

ANPSO's main difficulty is that over time r tends to zero. As the particles converge into species, the average distance between each particle and its nearest neighbour becomes very small. This makes it difficult for particles to join or form species. By the time a species has formed many other particles may have been attracted into the area. This may cause the algorithm to concentrate too many particles in a small area, to the detriment of other peaks that may have otherwise been discovered.

It is possible that both algorithms could be improved by dynamically allocating particles to species, depending on their performance. If we can reduce a species we know has already converged to a single particle, we free the remaining particles to search elsewhere instead of repeatedly sampling in a very small area. Preliminary tests showed that moving particles between species tended to disrupt the swarm, degrading performance. Reducing this disruption would allow us to allocate our available particles to the areas they are most needed.

8.2.2 Performance metrics

It may be possible to modify both the peak cover and BKPE metrics to better measure the diversity and convergence properties of an algorithm. By weighting each peak by the number of individuals on it, we can analyse how an algorithm is spending its resources. This may encourage the development of algorithms that dynamically allocate individuals to the most promising peaks. By maintaining only a minimal population on suboptimal peaks, an algorithm can increase its convergence speed on good peaks while still being able to react quickly to a low peak becoming the global optimum. This relates back to the potential research direction mentioned above.

8.2.3 Regression

Since the regression method is applicable to most numerical optimisation techniques, it is likely it will be of benefit to other algorithms too. For the Moving Peaks benchmark suite, a very promising candidate to apply regression to is mQSO. As the only performance enhancement shared by mQSO and rSPSO is speciation, it is highly likely that combining regression with mQSO, possibly to create mrQSO, would reduce the offline error to levels unachievable by either algorithm individually.

Another interesting area of research is to find in what other ways deterministic heuristics and methods can be combined with EC to improve performance. Even with functions where steepest ascent methods cannot be used, there is still a lot of information that is discarded by existing population-based optimisation algorithms. By keeping and analysing this information, it is likely further performance gains can be found.

8.3 Summary

This thesis presented several novel techniques for improving the performance of speciated algorithms. We developed and tested two algorithms that reduce the parameter-setting burden on the user while still offering competitive performance. We then showed that by using metrics that measure only specific aspects of an algorithm's performance, we can profile it to suggest the most effective way of improving its performance. Using this information, we were able to significantly improve the performance of an existing algorithm by using a regression technique. Finally we presented our conclusions and provided suggestions for future research.

Appendix A

Static test functions

Branin RCOS: 3 global optima

$f(x,y) = (y - \frac{5x^2}{4\pi^2} + \frac{5x}{\pi} - 6)^2 + 10(1 - \frac{1}{8\pi})cos(x) + 10$

$-5 \leq x \leq 10$; $0 \leq y \leq 15$

Ideal r for SPSO: 4

Six-Hump Camel Back: 2 global and 4 local optima

$f(x,y) = -4[(4 - 2.1x^2 + \frac{x^4}{3})x^2 + xy + (-4 + 4y^2)y^2]$

$-1.9 \leq x \leq 1.9$; $-1.1 \leq y \leq 1.1$

Ideal r for SPSO: 0.5

Deb's First function: 5 equally spaced global optima

$f(x) = sin^6(5\pi x)$

$0 \leq x \leq 1$

Ideal r for SPSO: 0.15

Himmelblau: 4 global optima

$f(x, y) = 200 - (x^2 + y - 11)^2 - (x + y^2 - 7)^2$

$-6 \leq x, y \leq 6$

Ideal r for SPSO: 3

Inverted Shubert: 3^n separated clusters of n global optima with many local optima

$f(x_1, x_2, ..., x_n) = -\prod_{i=1}^{n} \sum_{j=1}^{5} j \cos[(j + 1)x_i + j]$

$-10 \leq x_i \leq 10$

Ideal r for SPSO: 0.715

Bibliography

D. Ayvaz, H. Topcuoglu, and F. Gurgen. Hybrid techniques for dynamic optimization problems. *Lecture Notes in Computer Science*, 4263:95–104, 2006.

T. Bäck. On the behavior of evolutionary algorithms in dynamic fitness landscapes. In *Proceedings of the Congress on Evolutionary Computation*, pages 446–451, Anchorage, AK, May 1998. IEEE Press.

T. Bäck. *Evolutionary Algorithms in Theory and Practise: Evolution Strategies, Evolutionary Programming, Genetic Algorithms.* Oxford University Press, New York, NY, 1996.

D. Beasley, D. Bull, and R. Martin. A sequential niche technique for multimodal function optimization. *Evolutionary Computation*, 1(2):101–125, 1993.

A. Ben-Israel and T. Greville. *Generalized Inverses: Theory and Applications.* Wiley-Interscience, New York, NY, 1974.

M. Bessaou, A. Pétrowski, and P. Siarry. Island model cooperating with speciation for multimodal optimization. In *International Conference on Parallel Problem Solving from Nature*, pages 437–446, Paris, France, Sept 2000. Springer-Verlag.

S. Bird and X. Li. Adaptively choosing niching parameters in a PSO. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 3–10, Seattle, WA, 2006a. ACM Press.

S. Bird and X. Li. Informative performance metrics for dynamic optimisation problems. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 18–25, London, England, 2007a. ACM Press.

S. Bird and X. Li. Enhancing the robustness of a speciation-based PSO. In *Proceedings of the Congress on Evolutionary Computation*, pages 843–850, Vancouver, Canada, 2006b. IEEE Press.

S. Bird and X. Li. Using regression to improve local convergence. In *Proceedings of the Congress on Evolutionary Computation*, pages 1555–1562, Singapore, 2007b. IEEE Press.

T. Blackwell and J. Branke. Multiswarms, exclusion, and anti-convergence in dynamic environments. *Evolutionary Computation, IEEE Transactions on*, 10(4):459–472, 2006.

T. Blackwell and J. Branke. Multi-swarm optimization in dynamic environments. *Applications of Evolutionary Computing*, pages 489–500, 2004.

F. Branin. Widely convergent method for finding multiple solutions of simultaneous nonlinear equations. *IBM Journal of Research and Development*, 16(5):504–522, 1972.

J. Branke. Memory enhanced evolutionary algorithms for changing optimization problems. In *Proceedings of the Congress on Evolutionary Computation*, volume 3, pages 1875–1882, Washington, DC, July 1999. IEEE Press.

J. Branke. *Evolutionary Optimization in Dynamic Environments.* Kluwer Academic Publishers, Norwell, MA, 2002.

J. Branke and H. Schmeck. Designing evolutionary algorithms for dynamic optimization problems. *Advances in Evolutionary Computing: Theory and Applications*, pages 239–262, 2003.

R. Brits, A. Engelbrecht, and F. van den Bergh. A niching particle swarm optimizer. In *Proceedings of the Conference on Simulated Evolution and Learning*, volume 2, pages 692–696, Singapore, Nov 2002.

R. Brits, A. Engelbrecht, and F. van den Bergh. Scalability of Niche PSO. In *Proceedings of the Swarm Intelligence Symposium*, pages 228–234, Indianapolis, IN, Apr 2003. IEEE Press.

E. Cantú-Paz. A survey of parallel genetic algorithms. *Calculateurs Parallèles, Réseaux et Systèmes Répartis*, 10(2):141–171, 1998.

A. Carlisle and G. Dozier. Adapting particle swarm optimization to dynamic environments. In *International Conference on Artificial Intelligence*, volume 1, pages 429–434, Las Vegas, NV, 2000.

A. Carlisle and G. Dozler. Tracking changing extrema with adaptive particle swarm optimizer. In *Proceedings of the World Automation Congress*, volume 13, pages 265–270, Orlando, FL, 2002.

M. Clerc. The swarm and the queen: towards a deterministic and adaptive particle swarm optimization. In *Proceedings of the Congress on Evolutionary Computation*, volume 3, pages 1951–1957, Washington, DC, July 1999. IEEE Press.

M. Clerc and J. Kennedy. The particle swarm - explosion, stability, and convergence in a multidimensional complex space. *Evolutionary Computation, IEEE Transactions on*, 6: 58–73, Feb 2002.

H. Cobb. An investigation into the use of hypermutation as an adaptive operator in genetic algorithms having continuous, time-dependent nonstationary environments. *NRL Memorandum Report*, 6760:523–529, 1990.

D. Corne, M. Dorigo, F. Glover, D. Dasgupta, P. Moscato, R. Poli, and K. Price. *New ideas in optimization.* McGraw-Hill Ltd., Maidenhead, UK, England, 1999.

T. Cover and J. Thomas. *Elements of Information Theory.* Wiley-Interscience, New York, NY, 2006.

K. De Jong. *An Analysis of the Behavior of a Class of Genetic Adaptive Systems.* PhD thesis, University of Michigan, 1975.

K. Deb and D. Goldberg. An investigation of niche and species formation in genetic function optimization. In *Proceedings of the International Conference on Genetic Algorithms*, pages 42–50, San Francisco, CA, 1989. Morgan Kaufmann.

K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *Evolutionary Computation, IEEE Transactions on*, 6(2):182–197, 2002.

M. Dorigo and G. D. Caro. Ant colony optimization: a new meta-heuristic. In *Proceedings of the Congress on Evolutionary Computation*, volume 2, pages 1470–1477, Washington, DC, July 1999. IEEE Press.

R. Eberhart and Y. Shi. Comparing inertia weights and constriction factors in particle swarm optimization. In *Proceedings of the Congress on Evolutionary Computation*, volume 1, pages 84–88, La Jolla, CA, July 2000. IEEE Press.

R. Eberhart and Y. Shi. Tracking and optimizing dynamic systems with particle swarms. In *Proceedings of the Congress on Evolutionary Computation*, volume 1, pages 94–100, Seoul, South Korea, 2001. IEEE Press.

A. Eiben, R. Hinterding, and Z. Michalewicz. Parameter control in evolutionary algorithms. *Evolutionary Computation, IEEE Transactions on*, 3(2):124–141, 1999.

A. Engelbrecht and L. van Loggerenberg. Enhancing the NichePSO. In *Proceedings of the Congress on Evolutionary Computation*, pages 2297–2302, Singapore, Sept 2007.

D. Fogel. An introduction to simulated evolutionary optimization. *Neural Networks, IEEE Transactions on*, 5(1):3–14, 1994.

S. Geman, D. Geman, K. Abend, T. Harley, and L. Kanal. Stochastic relaxation, gibbs distributions and the bayesian restoration of images. *Journal of Applied Statistics*, 20(5): 25–62, 1993.

D. Goldberg and J. Richardson. Genetic algorithms with sharing for multimodal function optimization. In *Proceedings of the International Conference on Genetic Algorithms and their application*, pages 41–49, Cambridge, MA, 1987. Lawrence Erlbaum Associates, Inc.

D. Goldberg, K. Deb, and J. Horn. Massive Multimodality, Deception, and Genetic Algorithms. In *Parallel Problem Solving from Nature*, volume 2, pages 37–46, Brussels, Belgium, Sept 1992. Springer-Verlag.

B. Gottfried. *Introduction to Optimization Theory.* Prentice-Hall, Englewood Cliffs, NJ, 1973.

J. Grefenstette. Evolvability in dynamic fitness landscapes: A genetic algorithm approach. In *Proceedings of the Congress on Evolutionary Computation*, volume 3, pages 2031–2038, Washington, DC, July 1999. IEEE Press.

J. Grefenstette. Genetic algorithms for changing environments. In *Parallel Problem Solving from Nature*, volume 2, pages 137–144, Brussels, Belgium, Sept 1992. Springer-Verlag.

W. Hillis. Co-evolving parasites improve simulated evolution as an optimization procedure. *Artificial Life II, SFI Studies in the Sciences of Complexity*, 10:313–324, 1991.

J. Holland. *Adaptation in natural and artificial systems.* MIT Press, 1992.

X. Hu and R. Eberhart. Adaptive particle swarm optimization: detection and response to dynamic systems. In *Proceedings of the Congress on Evolutionary Computation*, volume 2, pages 1666–1670, Honolulu, HI, May 2002. IEEE Press.

L. Ingber. Simulated annealing: Practise versus theory. *Mathematical and Computer Modelling*, 18(11):29–57, 1993.

S. Janson and M. Middendorf. A hierarchical particle swarm optimizer for dynamic optimization problems. In *European Workshop on Evolutionary Algorithms in Stochastic and Dynamic Environments*, pages 513–524, Coimbra, Portugal, 2004. Springer.

Y. Jin. A comprehensive survey of fitness approximation in evolutionary computation. *Soft Computing-A Fusion of Foundations, Methodologies and Applications*, 9(1):3–12, 2005.

Y. Jin and J. Branke. Evolutionary optimization in uncertain environments-a survey. *Evolutionary Computation, IEEE Transactions on*, 9(3):303–317, 2005.

J. Kennedy. Stereotyping: improving particle swarm performance with cluster analysis. In *Proceedings of the Congress on Evolutionary Computation*, volume 2, pages 1507–1512, La Jolla, CA, July 2000. IEEE Press.

J. Kennedy and R. Eberhart. *Swarm intelligence*. Morgan Kaufmann, San Francisco, CA, 2001.

J. Kennedy and R. Eberhart. Particle swarm optimization. In *Proceedings of the Conference on Neural Networks*, volume 4, pages 1942–1948, Perth, Australia, Nov 1995. IEEE Press.

J. Kennedy and R. Mendes. Population structure and particle swarm performance. In *Proceedings of the Congress on Evolutionary Computation*, pages 1671–1676, Honolulu, HI, May 2002. IEEE Press.

H. Kim and S. Cho. An efficient genetic algorithm with less fitness evaluation by clustering. In *Proceedings of the Congress on Evolutionary Computation*, volume 2, pages 887–894, Seoul, South Korea, May 2001. IEEE Press.

S. Kirkpatrick, C. Gelatt, and M. Vecchi. Optimization by simulated annealing. *Science*, 220, 4598:671–680, May 1983.

J. Koza. *Genetic programming.* MIT Press Cambridge, Massachusetts, USA, 1992.

W. Langdon and R. Poli. Evolving problems to learn about particle swarm and other optimisers. In *Proceedings of the Congress on Evolutionary Computation*, volume 1, pages 81–88, Munich, Germany, July 2005. IEEE Press.

P. Larranaga and J. Lozano, editors. *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation.* Kluwer Academic Publishers, Norwell, MA, 2002.

P. Leopold, M. Montal, and J. Onuchic. Protein folding funnels: A kinetic approach to the sequence-structure relationship. *Proceedings of the National Academy of Sciences*, 89(18): 8721–8725, 1992.

J. Li, M. Balazs, G. Parks, and P. Clarkson. A species conserving genetic algorithm for multimodal function optimization. *Evolutionary Computation*, 10(3):207–234, 2002.

X. Li. A multimodal particle swarm optimizer based on fitness euclidean-distance ratio. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 78–85, London, England, 2007. ACM Press.

X. Li. Adaptively choosing neighbourhood bests using species in a particle swarm optimizer for multimodal function optimization. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 105–116, Seattle, WA, June 2004. Springer.

X. Li. Efficient differential evolution using speciation for multimodal function optimization. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 873–880, Washington, DC, June 2005.

X. Li and K. Dam. Comparing particle swarms for tracking extrema in dynamic environments. In *Proceedings of the Congress on Evolutionary Computation*, volume 3, pages 1772–1779, Canberra, Australia, Dec 2003. IEEE Press.

X. Li, J. Branke, and T. Blackwell. Particle swarm with speciation and adaptation in a dynamic environment. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 51–58, Seattle, WA, 2006. ACM Press.

K.-H. Liang, X. Yao, and C. Newton. Evolutionary search of approximated n-dimensional landscapes. *International Journal of Knowledge-Based Intelligent Engineering Systems*, 4 (3):172–183, July 2000.

S. Louis and J. Johnson. Solving similar problems using genetic algorithms and case-based memory. In *Proceedings of the International Conference on Genetic Algorithms*, pages 283–290, San Mateo, CA, 1997. Morgan Kaufmann.

S. Mahfoud. *Niching Methods for Genetic Algorithms*. PhD thesis, University of Illinois at Urbana-Champaign, 1995a.

S. Mahfoud. A comparison of parallel and sequential niching methods. In *Conference on Genetic Algorithms*, volume 136, pages 143–150, San Francisco, CA, 1995b. Morgan Kaufmann.

S. Mahfoud. Crowding and preselection revisited. *Parallel problem solving from nature, 2*, pages 27–36, 1992.

G. Mahinthakumar and M. Sayeed. Hybrid genetic algorithm - local search methods for solving groundwater source identification inverse problems. *Journal of Water Resources Planning and Management*, 131(1):45–57, 2005.

Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs.* Springer-Verlag, New York, 1996.

Z. Michalewicz and D. Fogel. *How to Solve It: Modern Heuristics.* Springer-Verlag, Berlin, Germany, 2000.

N. Mori, H. Kita, and Y. Nishikawa. Adaptation to changing environments by means of the memory based thermodynamical genetic algorithm. *Transactions of the Institute of Systems, Control and Information Engineers*, 14(1):33–41, 2001.

R. Morrison. *Designing Evolutionary Algorithms for Dynamic Environments.* Springer-Verlag, Berlin, Germany, 2004.

R. Morrison. Performance measurement in dynamic environments. In *GECCO Workshop on Evolutionary Algorithms for Dynamic Optimization Problems*, pages 5–8, Chicago, IL, July 2003. ACM Press.

I. Parmee, M. Johnson, and S. Burt. Techniques to aid global search in engineering design. In *Proceedings of the International Conference on Industrial and Engineering Applications* *of Artificial Intelligence and Expert Systems*, pages 377–385, Austin, TX, 1994. Gordon and Breach Science Publishers, Inc.

D. Parrott and X. Li. Locating and tracking multiple dynamic optima by a particle swarm model using speciation. *Evolutionary Computation, IEEE Transactions on*, 10(4):440–458, 2006.

A. Passaro and A. Starita. Clustering particles for multimodal function optimization. In *ECAI Workshop on Evolutionary Computation*, Riva del Garda, Italy, Aug 2006.

E. Peer, F. van den Bergh, and A. Engelbrecht. Using neighbourhoods with the guaranteed convergence PSO. In *Proceedings of the IEEE Swarm Intelligence Symposium*, pages 235–242, Indianapolis, IN, Apr 2003. IEEE Press.

A. Pétrowski. A clearing procedure as a niching method for genetic algorithms. In *Proceedings of the Congress on Evolutionary Computation*, pages 798–803, Nagoya, Japan, May 1996.

N. Raman and F. Talbot. The job shop tardiness problem: A decomposition approach. *European Journal of Operational Research*, 2:187–199, Sept 1993.

R. Regis and C. Shoemaker. Local function approximation in evolutionary algorithms for the optimization of costly functions. *Evolutionary Computation, IEEE Transactions on*, 8 (5):490–505, 2004.

R. Roy and I. Parmee. Adaptive restricted tournament selection for the identification of multiple sub-optima in a multi-modal function. *Lecture Notes in Computer Science, Evolutionary Computing*, pages 187–205, 1996.

B. Sareni and L. Krahenbuhl. Fitness sharing and niching methods revisited. *Evolutionary Computation, IEEE Transactions on*, 2(3):97–106, 1998.

Y. Shi and R. Eberhart. Fuzzy adaptive particle swarm optimization. In *Proceedings of the Congress on Evolutionary Computation*, volume 1, pages 101–106, Seoul, South Korea, 2001. IEEE Press.

Y. Shi and R. Eberhart. A modified particle swarm optimizer. In *Proceedings of the Congress on Computational Intelligence*, pages 69–73, Anchorage, AK, May 1998a. IEEE Press.

Y. Shi and R. Eberhart. Parameter selection in particle swarm optimization. *Evolutionary Programming*, 7:611–616, 1998b.

G. Singh and K. Deb. Comparison of multi-modal optimization algorithms based on evolutionary algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1305–1312, Seattle, WA, 2006. ACM Press.

R. Storn and K. Price. Differential evolution-a simple and efficient adaptive scheme for global optimization over continuous spaces. *Journal of Global Optimization*, 11(4):341–359, 1997.

A. Sutton, D. Whitley, M. Lunacek, and A. Howe. PSO and multi-funnel landscapes: how cooperation might limit exploration. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 75–82, Seattle, WA, July 2006. ACM Press.

R. Tanese. Parallel genetic algorithms for a hypercube. In *Proceedings of the Second International Conference on Genetic Algorithms on Genetic algorithms and their application*, pages 177–183, Mahwah, NJ, USA, 1987. Lawrence Erlbaum Associates, Inc.

R. Ursem. Multinational evolutionary algorithms. In *Proceedings of the Congress on Evolutionary Computation*, volume 3, pages 1633–1640, Washington, DC, July 1999. IEEE Press.

R. Ursem. Multinational GAs: Multimodal optimization techniques in dynamic environments. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 19–26, Las Vegas, NV, July 2000. ACM Press.

F. van den Bergh and A. Engelbrecht. A new locally convergent particle swarm optimiser. In *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, volume 3, pages 96–101, Hammamet, Tunisia, Oct 2002. IEEE Press.

K. Veeramachaneni, T. Peram, C. Mohan, and L. Osadciw. Optimization using particle swarms with near neighbor interactions. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 110–121, Chicago, IL, July 2003. Springer.

J. Vesterstrom, J. Riget, and T. Krink. Division of labor in particle swarm optimisation. In *Proceedings of the Congress on Evolutionary Computation*, volume 2, pages 1570–1575, Honolulu, HI, 2002. IEEE Press.

H. Wang, D. Wang, and S. Yang. Triggered memory-based swarm optimization in dynamic environments. *Lecture Notes in Computer Science*, 4448:637–646, 2007.

D. Wilde. *Optimum seeking methods.* Prentice-Hall, Englewood Cliffs, NJ, 1964.

D. Wolpert and W. Macready. No free lunch theorems for optimization. *Evolutionary Computation, IEEE Transactions on*, 1:67–82, April 1997.

X. Yao. An empirical study of genetic operators in genetic algorithms. *Microprocessing and Microprogramming*, 38(1-5):707–714, 1993.

P. Yin, S. Yu, P. Wang, and Y. Wang. A hybrid particle swarm optimization algorithm for optimal task assignment in distributed systems. *Computer Standards & Interfaces*, 28(4): 441–450, 2006.

X. Zhu and W. Wilhelm. Scheduling and lot sizing with sequence-dependent setup: A literature review. *IIE Transactions*, 38(18):987–1007, 2006.