INVESTIGATING ADAPTIVE MIGRATION SCHEMES FOR DISTRIBUTED EVOLUTIONARY ALGORITHMS

BY

MUHANNAD HIJAZE

Submitted for the Degree of

Doctor of Philosophy

on completion of research in the

Department of Computer Science

School of Mathematical and Computer Sciences

Heriot-Watt University

September 14

The copyright in this thesis is owned by the author. Any quotation from the thesis or use of any of the information contained in it must acknowledge this thesis as the source of the quotation or information.
Declaration

I hereby declare that the work presented in this thesis was carried out by myself at Heriot-Watt University, School of Mathematical and Computer Science, except where due acknowledgement is made, and not been submitted for any other degree.

_________________________________________
Muhannad Hijaze (Candidate)

_________________________________________
Professor David Wolfe Corne (Supervisor)

_________________________________________
Date
ACKNOWLEDGEMENTS

Professor David Wolfe Corne has been my academic advisor since when I began my PhD study in Computer Science at Heriot Watt University. His encouragement and advice over the years have been invaluable.

The Departments of Computer Science and Mathematics have my gratitude for the education and support provided. Thanks to the department’s Administration and IT support staff for the convenience they provided, especially for allow me to occupy numerous facilities to finish my work of this thesis.

Honourable thanks to my parents, brothers and sister in my beloved home country Syria and to my small family in Scotland (Farah, Jana and Taim). They continuously provide me great love, unlimited support and encouragement to achieve my target and success my dream. They always believe in me even if sometimes I doubted myself.

Last but not least, I would like to express my sincere gratitude and deep appreciation to my home country (Syria) who is facing great troubles and uprising events. Wishing this to be end very soon and become in peace and safety.


ABSTRACT

Evolutionary Algorithms (EAs) are a set of optimization techniques that have become highly popular in recent decades. One of the main reasons for this success is that they provide a general purpose mechanism for solving a wide range of problems. Several approaches have been proposed, each of them having different search characteristics. Many efforts have been made to make EAs search faster to reach a good solution in good time. One of the most promising alternatives is to employ parallel implementation. The parallel nature of evolutionary algorithms has been recognized for long time, and many have used distributed EAs to reduce the required time to get reasonable solutions for complex problems. Evolutionary algorithms work with populations of independent solution which make it naturally be easy to distribute the computational load among several processes. Distributed Evolutionary Algorithms (dEAs) become increasingly interesting and important for three main reasons: (i) a well-designed dEA can outperform a ‘standard' EA in terms of reliability, solution quality, and speed; (ii) they can (of course) be implemented on parallel hardware and platforms, and hence combine efficient utilization of parallel resources with very fast and reliable optimization; (iii) parallel hardware resources are increasingly common. A dEA operates as separate evolving populations with occasional interaction between them via ‘migration’. A specific dEA is characterized by the topology and nature of these interactions. Although the field is sizeable, there is still relatively little exploration of the performance of alternative topologies and interaction mechanisms.

This thesis introduces and evaluates some novel dEA architectures and algorithms to solve high dimensional problems. The performances of these algorithms are compared with standard genetic algorithms and other well-known contemporary evolution algorithms such as GA-MPC. Some significant results have emerged from our investigations and test on these developed algorithms. In particular, we found that a range of simple ideas for the distributed architecture and for the approach to migration can lead to robust improvements in performance. The resulting algorithms are very promising and compete very well against a state of the art dEA both in well-defined real-parameter function optimization problems and practical evaluation-expensive problems.
# TABLE OF CONTENTS

Acknowledgements..............................................................................................................i
Publications..........................................................................................................................ii
Abstract..................................................................................................................................iii
List of Tables ..........................................................................................................................vii
List of Figures ........................................................................................................................xii

1. Chapter I  Introduction ........................................................................................................1
   1.1 Overview.........................................................................................................................1
   1.2 Basic Definitions...............................................................................................................2
   1.3 Metaheuristics ................................................................................................................5
   1.4 The Distributed Genetic Algorithms .................................................................................6
   1.5 Motivation .......................................................................................................................7
   1.6 Contributions ..................................................................................................................8
   1.7 Document Outline ..........................................................................................................9
   1.8 Conclusions ....................................................................................................................10

2. Chapter II  Literature Review ............................................................................................11
   2.1 Evolutionary Algorithms Overview:................................................................................11
      2.1.1 The Simple Genetic Algorithm (SGA)........................................................................13
   2.2 Parameters and Operators ..............................................................................................16
      2.2.1 Population Size........................................................................................................16
      2.2.2 Selection Operators ................................................................................................17
      2.2.3 Mutation Operator ..................................................................................................19
      2.2.4 Crossover Operator ................................................................................................20
   2.3 General Difficulties .........................................................................................................21
   2.4 Distributed Evolutionary Algorithms (dEAs) .................................................................22
      2.4.1 Master-Slave Parallelization ..................................................................................25
2.4.2 Fine-Grained Parallelization

2.4.3 Multiple-Deme Coarse-grained Parallelization

2.4.4 Hierarchical Parallelization

2.5 Master-Slave Model Advantages and Limitations

2.6 Conclusion

3. Chapter III Parallel Models And Topologies

3.1 Overview

3.2 The Parallel Models

3.2.1 Model I

3.2.2 Model II

3.2.3 Model III

3.3 Master and Client Processes

3.3.1 Master Side Process

3.3.2 Client’s Side Process

3.4 Benchmark Test Functions

3.5 Experiments

3.5.1 Experimental Results

3.6 Conclusion

4. Chapter IV Investigating Adaptive Migration Strategies

4.1 Overview

4.2 The dEA with Adaptive Migration Schemes dEA-AMS

4.3 Experiments

4.3.1 Benchmark Test Functions

4.3.2 The Distributed GA Parameters

4.3.3 The Experimental Results

4.4 Conclusion
5. Chapter V  Investigating Diversity-Based Guidance In Distributed Genetic Algorithms

5.1 A Brief Overview of Diversity-Guided Control in EAs

5.2 Diversity-Guided Control in Distributed Evolutionary Algorithm

5.2.1 Monitoring Diversity and Injection of DACs

5.2.2 An artificial chromosome generation mechanism

5.2.3 Integrating DAC-Injection Based Diversity Control In The dEA

5.2.4 Population Diversity Measures

5.3 The Algorithm Design $DGdGA$

5.4 Experiments

5.4.1 Benchmark Test Functions

5.4.2 The Distributed GA Parameters

5.4.3 The Experimental Results

5.5 Conclusion

6. Chapter VI  Validation of Algorithms on Real-World Problems

6.1 The Real-World Test Problems IEEE-CEC’11

6.2 The Test Results

6.3 Conclusion

7. Chapter VII  Conclusion

7.1 Summary

7.2 Future Work

A. Appendix A Genetic Algorithms: Further Details

B. Appendix B

Bibliography
LIST OF TABLES

Table 2-1: neo-Darwinian vs. Evolutionary Computation.............................................12
Table 2-2: Evolutionary Algorithms’ Categories.............................................................13
Table 3-1: dGA parameters and attributes ......................................................................51
Table 3-2: Mean execution time (ms) to optimize benchmark functions for SGA and dGAT1, dGAT2, dGAT3 over a cluster of 16 clients .................................................................53
Table 3-3: The Speedup SN values of dGAT1, dGAT2 and dGAT3 over SGA of 30D benchmark problems ........................................................................................................55
Table 3-4: The dGAT1, dGAT2 and dGAT3 Parallel Efficiency PE values .......................57
Table 3-5: The dGAT1, dGAT2 and dGAT3 Serial Fraction SF values .............................57
Table 3-6: The 16 clients paired 1-tailed T-Test p-values for the comparison between SGA execution times and dGAT1, dGAT2, dGAT3 .................................................................60
Table 3-7: The 16 clients paired 1-tailed T-Test p-values for the comparison between the execution times of dGAT1, dGAT2 and dGAT3.................................................................60
Table 3-8: The dGAT1, dGAT2 and dGAT3 mean execution times (ms) for 8 clients ...62
Table 3-9: The Speedup SM values of dGAT1, dGAT2 and dGAT3 with 8 clients .........64
Table 3-10: The Serial Fraction SFM of dGAT1, dGAT2 and dGAT3 with 8 clients ......65
Table 3-11: The speedup SNM of dGAT1, dGAT2 and dGAT3 .........................................66
Table 3-12: The 8 clients paired 1-tailed T-Test p-values for the comparison between the execution times of dGAT1, dGAT2 and dGAT3.................................................................67
Table 4-1: dGA parameters and attributes .................................................................78
Table 4-2: The AdGAT1, AdGAT2 and AdGAT3 success rates out of 20 to optimize the 30D benchmark functions with adaption regime 100(300). ........................................79
Table 4-3: The dGAT1, dGAT2 and dGAT3 success rates out of 20 to optimize the 30D benchmark functions ........................................................................................................80
Table 4-4: The Speedup values of the adapted models AdGAT1, AdGAT2 and AdGAT3 of adaptive regime 100(300) over the dGAT1, dGAT2 and dGAT3 of 30D benchmark problems .......................................................................................80
Table 4-5: The Serial Fraction values of the adapted models $AdGAT1$, $AdGAT2$ and $AdGAT3$ of adaptive regime 100(300) over the $dGAT1$, $dGAT2$ and $dGAT3$ of the 30D benchmark problems.

Table 4-6: The $AdGAT1$, $AdGAT2$ and $AdGAT3$ success rates out of 20 to optimize benchmark functions with 50D adaption regime 100(300).

Table 4-7: The $dGAT1$, $dGAT2$ and $dGAT3$ success rates out of 20 to optimize the 50D benchmark functions.

Table 4-8: The Speedup values of the adapted models $AdGAT1$, $AdGAT2$ and $AdGAT3$ of adaptive regime 100(300) over the $dGAT1$, $dGAT2$ and $dGAT3$ with 50D benchmark functions.

Table 4-9: The Serial Fraction values of the adapted models $AdGAT1$, $AdGAT2$ and $AdGAT3$ of adaptive regime 100(300) over the $dGAT1$, $dGAT2$ and $dGAT3$ of the 50D benchmark problems.

Table 4-10: The 50D benchmarks paired 1-tailed T-Test p-values for the comparison between the execution times of adaptive models $AdGAT1$, $AdGAT2$ and $AdGAT3$.

Table 4-11: The $AdGAT1$, $AdGAT2$ and $AdGAT3$ success rates out of 20 to optimize the 100D benchmark functions with adaption regime 100(300).

Table 4-12: The $dGAT1$, $dGAT2$ and $dGAT3$ success rates out of 20 to optimize the 100D benchmark functions.

Table 4-13: The Speedup values of the adapted models $AdGAT1$, $AdGAT2$ and $AdGAT3$ of adaptive regime 100(300) over the $dGAT1$, $dGAT2$ and $dGAT3$ with 100D benchmark functions.

Table 4-14: The Serial Fraction values of the adapted models $AdGAT1$, $AdGAT2$ and $AdGAT3$ of adaptive regime 100(300) over the $dGAT1$, $dGAT2$ and $dGAT3$ of the 100D benchmark problems.

Table 4-15: The 100D benchmarks paired 1-tailed T-Test p-values for the comparison between the execution times of adaptive models $AdGAT1$, $AdGAT2$ and $AdGAT3$.

Table 5-1: dGA parameters and attributes.

Table 5-2: The $DGdGAT1$, $DGdGAT2$ and $DGdGAT3$ success rates out of 240 to optimize the 30D x 12 benchmark test functions using diversity threshold $dmin \in 0.1, 0.4$ & $dmax \in 0.5, 0.8$. 
Table 5-3: The Speedup values of the diversity guided models The $DGdGAT1$, $DGdGAT2$ and $DGdGAT3$ against the fixed models $dGAT1$, $dGAT2$ and $dGAT3$ of 30D benchmark problems. ................................................................. 115

Table 5-4: The Serial Fraction values of the diversity guided models The $DGdGAT1$, $DGdGAT2$ and $DGdGAT3$ against the fixed models $dGAT1$, $dGAT2$ and $dGAT3$ of the 30D benchmark problems. ........................................................................ 116

Table 5-5: The mean Speedup values of the adapted models $AdGAT1$, $AdGAT2$ and $AdGAT3$ of the 30D benchmark problems with adaptive scheme 100,300. .................. 117

Table 5-6: The mean Serial Fraction values of the adapted models $AdGAT1$, $AdGAT2$ and $AdGAT3$ of the 30D benchmark problems with adaptive scheme 100,300........... 118

Table 5-7: The AdGAT1, AdGAT2 and AdGAT3 success rates out of 220 runs to optimize 50D benchmark functions using diversity threshold $d_{\text{min}} \in 0.1$, $0.4$ & $d_{\text{max}} \in 0.5, 0.8$. .............................................................................................................. 122

Table 5-8: The Speedup values of the diversity guided models The $DGdGAT1$, $DGdGAT2$ and $DGdGAT3$ against the fixed models $dGAT1$, $dGAT2$ and $dGAT3$ of 50D benchmark problems. ........................................................................ 124

Table 5-9: The Serial Fraction values of the diversity guided models The $DGdGAT1$, $DGdGAT2$ and $DGdGAT3$ against the fixed models $dGAT1$, $dGAT2$ and $dGAT3$ of the 50D benchmark problems. ........................................................................ 125

Table 5-10: The mean Speedup values of the adapted models $AdGAT1$, $AdGAT2$ and $AdGAT3$ of the 50D benchmark problems with adaptive scheme 100,300. .................. 126

Table 5-11: The mean Serial Fraction values of the adapted models $AdGAT1$, $AdGAT2$ and $AdGAT3$ of the 50D benchmark problems with adaptive scheme 100,300........... 126

Table 5-12: The 50D benchmarks paired 1-tailed T-Test p-values for the comparison between the execution times of diversity guided models $DGdGAT1$, $DGdGAT2$ and $DGdGAT3$ using the diversity threshold of $(d_{\text{min}}, d_{\text{max}}) = (0.2, 0.8)$ ............... 129

Table 5-13: The $DGdGAT1$, $DGdGAT2$ and $DGdGAT3$ mean success rates out of 220 to optimize the 100D benchmark functions with diversity threshold $d_{\text{min}} \in 0.1$, $0.4$ & $d_{\text{max}} \in 0.5, 0.8$. .............................................................................................................. 131
Table 5-14: The Speedup values of the diversity guided models The $DGdGAT1$, $DGdGAT2$ and $DGdGAT3$ against the fixed models $dGAT1$, $dGAT2$ and $dGAT3$ of 100D benchmark problems. ........................................................................................................ 133

Table 5-15: The Serial Fraction values of the diversity guided models The $DGdGAT1$, $DGdGAT2$ and $DGdGAT3$ against the fixed models $dGAT1$, $dGAT2$ and $dGAT3$ of the 100D benchmark problems. ........................................................................................................ 136

Table 5-16: The mean Speedup values of the adapted models $AdGAT1$, $AdGAT2$ and $AdGAT3$ of the 100D benchmark problems with adaptive scheme 100,300. .................. 137

Table 5-17: The mean Serial Fraction values of the adapted models $AdGAT1$, $AdGAT2$ and $AdGAT3$ of the 100D benchmark problems with adaptive scheme 100,300. ........... 137

Table 5-18: The 100D benchmarks paired 1-tailed T-Test p-values for the comparison between the execution times of diversity guided models $DGdGAT1$, $DGdGAT2$ and $DGdGAT3$ using the diversity threshold of $(d_{min}, d_{max}) = (0.2, 0.8)$ ............... 139

Table 6-1: The mean fitness value for 22 test functions optimized by the studied models and GA-MPC ............................................................................................................................................. 144

Table 6-2: Some speedup values for comparisons between $AdGAT1,2,3$ vs. $dGAT1,2,3$ , $AdGAT1,2,3$ vs. $DGdGAT1,2,3$ and $DGdGAT1,2,3$ vs. $dGAT1,2,3$ .............................................. 146

Table A-1: Interval Properties ........................................................................................................................................ 172

Table A-2: Families of Fuzzy Connectives ........................................................................................................................................ 173

Table A-3: Set of crossover operators ........................................................................................................................................ 174

Table B-1: Benchmark test function with normal version ........................................................................................................ 190

Table B-2: Benchmark test function with shifted version ........................................................................................................ 191

Table B-3: The 16 clients’ average execution (ms) time of 20 runs to optimise the normal version of 30D benchmark test problems ........................................................................................................ 193

Table B-4: The 16 clients’ average execution time (ms) of 20 runs to optimise the shifted version of 30D benchmark test problems ........................................................................................................ 193

Table B-5: The 8 clients’ average execution time (ms) of 20 runs to optimise the normal version of 30D benchmark test problems ........................................................................................................ 194

Table B-6: The 8 clients’ average execution time (ms) of 20 runs to optimise the shifted version of 30D benchmark test problems ........................................................................................................ 194
Table B-7: The 16 clients’ average execution time (ms) of 20 runs to optimise the normal version of 50D benchmark test problems ................................................................. 195

Table B-8: The 16 clients’ average execution time (ms) of 20 runs to optimise the shifted version of 50D benchmark test problems ................................................................. 195

Table B-9: The 16 clients’ average execution time (ms) of 20 runs to optimise the normal version of 100D benchmark test problems ................................................................. 196

Table B-10: The 16 clients’ average execution time (ms) of 20 runs to optimise the shifted version of 100D benchmark test problems ................................................................. 196

Table B-11: The Summary of IEEE-CEC’11 [69] benchmark problems .................... 197

Table B-12: Model T1 Speedup values of CEC 2011 benchmark test problems ........ 200

Table B-13: Model T2 Speedup values of CEC 2011 benchmark test problems ........ 201

Table B-14: Model T3 Speedup values of CEC 2011 benchmark test problems ........ 202
LIST OF FIGURES

Figure 2-1: Parallel Evolutionary Algorithms dEAs  a) master-slave dEAs; b) fine-grained dEAs; c) multiple-population dEAs .................................................................24

Figure 2-2: Hierarchical Parallel GA:  a) multi-deme + fine-grained, b) multi-deme + master-slave, c) multi-deme + multi-deme .........................................................33

Figure 2-3: The [237] TriBA’s structures $k \in [0,3]$ .................................................34

Figure 3-1: Model I, Master-Slave Multiple-Deme dGA, all clients connected to master process ........................................................................................................41

Figure 3-2: Model II, Master-Slave Multiple-Deme dGA, half clients connected to master process .......................................................................................................42

Figure 3-3: Model III, Master-Slave Multiple-Deme dGA, 2×3D Cubes based on Alba et al. [8] model .................................................................................................43

Figure 3-4: Benchmark test function from [157]; a) sphere, b) Rosenbrock, c) Rastrigin, d) Schwefel, e) and f) Griewangk, g) and h) Ackley ..................................................50

Figure 3-5 Benchmark mean execution time (ms) for $dGAT1$, $dGAT2$ and $dGAT3$ ......54

Figure 3-6: The T1, T2, and T3 mean speedup values for normal and shifted fitness functions ..................................................................................................................56

Figure 3-7: The $dGAT1$, $dGAT2$ and $dGAT3$ Serial Fraction .......................................59

Figure 3-8: The $dGAT1$, $dGAT2$ and $dGAT3$ Parallel Efficiency .................................59

Figure 3-9: The three models with 8 clients, (a) model T1, (b) model T2, (c) model T3 the master process does not appear in model T3......................................................63

Figure 3-10: The $dGAT1$, $dGAT2$ and $dGAT3$ Serial Fraction $SF_{M}$ with 8 clients .......66

Figure 4-1: The $AdGAT1$, $AdGAT2$ and $AdGAT3$ Speedup values of the (100) adaptive regime of the 30D benchmark test problems ..................................................82

Figure 4-2: The $AdGAT1$, $AdGAT2$ and $AdGAT3$ Speedup values of the (300) adaptive regime of the 30D benchmark test problems .....................................................82

Figure 4-3: The $AdGAT1$, $AdGAT2$ and $AdGAT3$ Serial Fraction values of the (100) adaptive regime of the 30D benchmark test problems ..............................................83
Figure 4-4: The $AdGAT_1$, $AdGAT_2$ and $AdGAT_3$ Serial Fraction values of the (300) adaptive regime of the 30D benchmark test problems.................................................................83

Figure 4-5: The $AdGAT_1$, $AdGAT_2$ and $AdGAT_3$ Speedup values of the (100) adaptive regime of the 50D benchmark test problems. .................................................................87

Figure 4-6: The $AdGAT_1$, $AdGAT_2$ and $AdGAT_3$ Speedup values of the (300) adaptive regime of the 50D benchmark test problems. .................................................................88

Figure 4-7: The $AdGAT_1$, $AdGAT_2$ and $AdGAT_3$ Serial Fraction values of the (100) adaptive regime of the 50D benchmark test problems.................................................................88

Figure 4-8: The $AdGAT_1$, $AdGAT_2$ and $AdGAT_3$ Serial Fraction values of the (300) adaptive regime of the 50D benchmark test problems.................................................................89

Figure 4-9: The $AdGAT_1$, $AdGAT_2$ and $AdGAT_3$ Speedup values of the (100) adaptive regime of the 100D benchmark test problems. .................................................................94

Figure 4-10: The $AdGAT_1$, $AdGAT_2$ and $AdGAT_3$ Speedup values of the (300) adaptive regime of the 100D benchmark test problems. .................................................................95

Figure 4-11: The $AdGAT_1$, $AdGAT_2$ and $AdGAT_3$ Serial Fraction values of the (100) adaptive regime of the 100D benchmark test problems.................................................................95

Figure 4-12: The $AdGAT_1$, $AdGAT_2$ and $AdGAT_3$ Serial Fraction values of the (300) adaptive regime of the 100D benchmark test problems.................................................................96

Figure 5-1: The $DGdGAT_1$, $DGdGAT_2$ and $DGdGAT_3$ success rates out of 240 to optimize the 30D x 12 benchmark test functions using diversity threshold $d_{min} \in 0.1,$ $0.4$ & $d_{max} \in 0.5, 0.8$. .................................................................114

Figure 5-2: The $DGdGAT_1$, $DGdGAT_2$ and $DGdGAT_3$ Speedup values of the 30D benchmark test problems........................................................................................................118

Figure 5-3: The $DGdGAT_1$, $DGdGAT_2$ and $DGdGAT_3$ serial fraction values of the 30D benchmark test problems........................................................................................................119

Figure 5-4: Comparison between the mean Speedup values of the Adaptive (100 & 300) and Diversity Guided ($d_{min} = 0.2$ & $d_{max} = 0.7, 0.8$) models of the 30D benchmark test problems.................................................................120

Figure 5-5: Comparison between the mean Serial Fraction values of the Adaptive (100 & 300) and the Diversity Guided ($d_{min} = 0.2$ & $d_{max} = 0.7, 0.8$) models of the 30D benchmark test problems.................................................................120
Figure 5-6: The $DGdGAT_1$, $DGdGAT_2$ and $DGdGAT_3$ success rates out of 220 runs to optimize the 50D x 12 benchmark test functions using diversity threshold $d_{min} \in 0.1, 0.4 \& d_{max} \in 0.5, 0.8$. ..........................................................123

Figure 5-7: The $DGdGAT_1$, $DGdGAT_2$ and $DGdGAT_3$ Speedup values of the 50D benchmark test problems..........................................................127

Figure 5-8: The $DGdGAT_1$, $DGdGAT_2$ and $DGdGAT_3$ Serial Fraction values of the 50D benchmark test problems..........................................................127

Figure 5-9: Comparison between the mean Speedup values of the Adaptive (100 & 300) and Diversity Guided ($d_{min} = 0.2 \& d_{max} = 0.7, 0.8$) models of the 50D benchmark test problems..........................................................128

Figure 5-10: Comparison between the mean Serial Fraction values of the Adaptive (100 & 300) and the Diversity Guided ($d_{min} = 0.2 \& d_{max} = 0.7, 0.8$) models of the 50D benchmark test problems..........................................................128

Figure 5-11: The $DGdGAT_1$, $DGdGAT_2$ and $DGdGAT_3$ success rates out of 220 runs to optimize the 100D x 12 benchmark test functions using diversity threshold $d_{min} \in 0.1, 0.4 \& d_{max} \in 0.5, 0.8$. ..........................................................132

Figure 5-12: The $DGdGAT_1$, $DGdGAT_2$ and $DGdGAT_3$ Speedup values of the 100D benchmark test problems..........................................................134

Figure 5-13: The $DGdGAT_1$, $DGdGAT_2$ and $DGdGAT_3$ Serial Fraction values of the 100D benchmark test problems..........................................................135

Figure 5-14: Comparison between the mean Speedup values of the Adaptive (100 & 300) and Diversity Guided ($d_{min} = 0.2 \& d_{max} = 0.7, 0.8$) models of the 100D benchmark test problems..........................................................137

Figure 5-15: Comparison between the mean Serial Fraction values of the Adaptive (100 & 300) and the Diversity Guided ($d_{min} = 0.2 \& d_{max} = 0.7, 0.8$) models of the 100D benchmark test problems..........................................................138

Figure 6-1: Convergence plots for problem P1, a) $GA-MPC$ and $dGAT_{1,2,3}$ b) $DGdGAT_{1,2,3}$ c) $AdGAT_{1,2,3}$..........................................................147

Figure A-1 : Morlet Wavelet..........................................................163

Figure A-2: One Point Crossover..........................................................165

Figure A-3: Two-Point Crossover; a) Normal representing, b) Ring representing ..... 166
Figure A-4: M-Point Crossover, m=4 .................................................. 167
Figure A-5: Uniform Crossover with bit-mask ....................................... 168
Figure A-6: Action interval for a gene .................................................. 171
Figure A-7: Genes generated using F, S, M and L .................................. 172
Figure A-8: Poli et al. OneMix fitness function with low pass filter, l=100, a=0.6 ..... 180
Figure A-9: The [203] Valley fitness function ..................................... 187
Evolutionary Algorithms (EA) have been popular ever since the idea of using evolution in computer science, which even dates back to 1948 to Alan Turing [66]. Throughout the years many developments have taken place in the field of evolutionary computing and it is still one of the most studied topics.

1.1 Overview

Genetic Algorithms (GAs) are search methods utilizing principles of natural selection and genetics. GAs have been applied successfully to a huge number of problems in different areas in business, engineering, and sciences [93]. In a wide range of practical applications, GAs can successfully locate good solutions in an acceptable amount of time. However, in some particular cases, GAs may require very huge amounts (maybe hundreds of thousands) of expensive function evaluation. Depending on the cost of each evaluation, this process may takes days, months or even years to search and locate an acceptable solution. GAs are loosely based on some biological processes that can be seen in nature, like natural selection [49] or genetic inheritance [142] of parental good traits. Part of the evolution is determined by the natural selection of different individuals competing for resources in the environment. Therefore, some individuals are better than others. Those that are better are more likely to survive, learn, and propagate their genetic material.

Sexual reproduction allows some shuffling of chromosomes, producing offspring that contains a combination of information from each parent. This is known as the recombination operation, which is often referred to as crossover because of the way that biologists have observed strands of chromosomes crossing over during the exchange. Recombination happens in an environment where the selection of mates for reproduction is largely a function of the fitness of individuals, i.e., how good each individual is at competing in its environment. As in the biological case, individuals can occasionally mutate. Mutation is an important source of diversity for GAs. In a GA, a large amount of diversity is usually introduced at the start of the algorithm by randomizing the genes in the population. The importance of mutation, which introduces
further diversity while the algorithm is running, is a matter of debate. Some refer to it as a background operator, simply replacing some of the original.

Over years, there have been multiple efforts to make GAs faster and reliable. One of the most promising alternatives is to use parallel implementations. The parallel nature of genetic algorithms has been noticed for long time, and many researchers have employed the parallel GAs to reduce the processing time required to reach suitable solutions to such complex problems. In fact, GAs are very easy to efficiently implement on parallel machines. However, in contrast of their operational simplicity, parallel GAs are complex non-linear algorithms that are controlled by many parameters that affect their searching performance (efficiency and solution quality). Tuning these parameters correctly is critical to successfully locate good results reliably and rapidly.

Practically, the construction of distributed Genetic Algorithms (dGAs) involves different choices such as using a single or multiple populations. However, in this sense, the population size must be determined carefully, and for multiple populations, it must be decided how many to be used. In addition, the populations may remain isolated or they may communicate by exchanging individuals or some other information. Communications between sub-populations might involve extra cost and additional decisions on the pattern of communications, the number of individuals to be exchanged and on the frequency of communications.

1.2 Basic Definitions

It is a good idea to begin this section with some important definitions that will explain some basic notations that will be used in the context of the following chapters. Initially, we give a formal definition of optimization. Assuming minimization (without any loss in generality), we can define any optimization problem as follows:

\textbf{Optimization:} An optimization problem is formalized by a pair \((S,F)\), where \(S \neq \emptyset\) represents the solution space -or search space- of the problem, while \(f\) is a quality criterion known as the objective function, defined as:

\[ f : S \to \mathbb{R} \]
Thus, solving an optimization problem consists in finding a set of decision variables values such that the represented solutions by these values $i^* \in S$ satisfy the following inequality:

$$f(i^*) \leq f(i), \quad \forall i \in S \tag{1-2}$$

Assuming maximization or minimization does not restrict the generality of the results, we can establish equivalence between the maximization and minimization problems as:

$$\max\{f(i)| i \in S \} \equiv \min\{-f(i)| i \in S\} \tag{1-3}$$

A definition of proximity or distance between different solutions of the search space is necessary for solving an optimization problem. Two solutions are close to each other if they belong to the same neighbourhood in the search space. We define the neighbourhood of a solution as:

**Neighbourhood:** Assume $(S, F)$ is an optimization problem; a neighbourhood structure in $S$ can be defined as:

$$N : S \rightarrow S \tag{1-4}$$

such that, for each solution $i \in S$ a set $S_i \subseteq S$ is defined. It also holds that if $i$ is in the neighbourhood of $j$, then $j$ is also in the neighbourhood of $i$; $i : j \in S_i$ iff $i \in S_j$.
In general, in a complex optimization problem, the objective function often presents an optimal solution that is an optimum only in its neighbourhood, but which is not optimal if we consider the whole search space. Therefore, a global search method can be easily trapped in an optimal value inside a neighbourhood, thus giving rise to the concept of local optimum:

**Local optimum:** Assume \((S, F)\) is an optimization problem, and \(S_i \subseteq S\) the neighborhood of a solution \(i^' \in S_i\), \(i^'\) is a local optimum if the next inequality is satisfied (assuming minimization):

\[
f(i^') \leq f(i), \quad \forall i \in S_i
\]

When processing real life optimization problems, one usually has to deal with constraints. Thus, the area of possible solutions \(S\) is restricted to those that fulfil all the constraints. Consequently, the next definition is needed:

**Optimization with constraints:** Given an optimization problem \((S, F)\), we define

\[
M = \{ i \in S \mid g_k(i) \geq 0, \forall k \in [1, ..., q]\}
\]

as the region of feasible solutions of the objective function \(f : S \to \mathbb{R}\). The functions \(g_k : S \to \mathbb{R}\) are called constraints, and these constraints are named differently according to the value taken by \(i \in S\) as follow:

- **satisfied** : \(g_k(i) \geq 0\)
- **active** : \(g_k(i) = 0\)
- **inactive** : \(g_k(i) > 0\)
- **violated** : \(g_k(i) < 0\)

While the *unconstrained optimization* is the term used when no constraints exist, i.e. \(\text{iff } M = S\).
1.3 Metaheuristics

In the research literature one can find many different schemes of algorithmic methods in order to solve an optimization problem. Optimization can be done by either exact or approximate methods. The exact algorithms guarantee to find the optimal solution for all the existing (finite set of) instances. Generally, since exact methods typically need exponential computation times when facing large instances of complex problems, NP-hard problems cannot be realistically tackled. Therefore, the use of approximate techniques is a rising topic in the last decades. These methods involve a risk to lose the guarantee of finding the global optimal solution (often, but not always) in order to find good solutions in a significantly shorter time compared to exact methods. For the last twenty years a new kind of approximate technique has been emerging, consisting basically in combining simple ad-hoc heuristic methods (approximate algorithms with stochastic guided components) in higher level search strategies in order to explore and to exploit the search space efficiently and effectively. These methods are commonly known as metaheuristics. In [30] the reader can find some metaheuristic definitions given by different authors, but in general one can state that metaheuristics are high level strategies having a given structure that plans the application of a set of operations (variation operators) to explore high dimensional and complex search spaces.

Metaheuristics can be classified in many different ways. In [30] a classification is given according to different properties which characterize them. One of these classifications relies on the number of solutions: population based (work with a set of solutions) and trajectory based (work with a single solution). With a single initial solution, and at each step of the search the current solution is replaced by another (often the best) solution found in its explored neighbourhood. Frequently, such a metaheuristic allows to find a local optimal solution, and so they are called exploitation-oriented methods. On the other hand, population based methods make use of a population of solutions. The initial population is enhanced through a natural (or purely mathematical) evolution process. At each generation of the process, the whole population or a part of the population is replaced by newly generated individuals (usually the best ones). Population based methods are often called exploration-oriented methods. Among the best well-known metaheuristics some examples are evolutionary algorithms (EAs) [22], simulated annealing (SA) [143], tabu search (TS) [88], variable neighbourhood search (VNS) [172], and ant colony optimization (ACO) [63].
1.4 The Distributed Genetic Algorithms

Most genetic algorithms use a single population of individuals and apply operators on them as a whole. In contrast, a tradition also exists in using structured GAs (where the population is decentralized somehow), especially in relation to their parallel implementation. The use of parallel multiple populations is based on the idea that the isolation of populations allows to keep a higher genetic differentiation [260]. In many cases [10], these algorithms using decentralized populations provide a better sampling of the search space and improve both the numerical behaviour and the execution time of an equivalent panmictic algorithm. Among the many types of structured GAs, distributed algorithms are one of the most popular optimization tools.

A distributed GA (dGA) [9, 156] is defined as a method that involves more than one individual copy (usually serial) of a genetic algorithm. This structure requires for each copy, additional opportunities for interaction that support exchange of information among the several copies. The design and implementation of dGAs are affected by two main things, the nature of the individual GAs (parameters/attributes) and the type of communication that is established among them.

In general, the distributed GAs parameters are found either by systematic experiments or just by chance [35]. These approaches usually cause a waste of computing resources or inadequate search quality, which sometimes leads practitioners to dismiss the distributed GAs as being impractical or unreliable. The research in this thesis aims in part to address the problem of fine tuning some of the dGA parameters, and suggest general rules to control and adapt parameters wisely. The consequences are effective dGAs that consistently reach high quality solutions to hard problems.

The research in this thesis is largely comprised of algorithm design and experimentation, to develop and test different parallel dGA models by observing the effects of different parameters and design choices on the algorithms' performance. This study does not develop exact models of every facet of distributed genetic algorithms, as the exact and complete models would probably be too difficult and expensive to be implemented and used, and therefore would have little practical significance. Instead, the intention of this research is to present and discuss some simple approximate models that are easy to calibrate to the particular problems and to the hardware used in the implementation.
In fact, in the short term, faster implementations that are made possible only by hardware enhancements will be adequate to satisfactorily solve many current problems; however, in the long run, the increasing complexity of some problems will exceed any hardware superiority. Since the parallel models introduced in this thesis are almost certainly applicable despite future developments in hardware, we will be able to design algorithms that can exploit those developments without having to experiment blindly with all possible settings of the parameters.

1.5 Motivation

When real-world problems are faced using EAs, times-to-solution usually tends to take from days to even months (check for instance [243]). As a consequence, researchers have tried over the years to reduce the time-to-solution constraints. There are several reasons for such large time-to-solution requirements. The main ones are:

- **The evaluation phase**: In EAs, one of the most expensive tasks of the algorithm in terms of execution time is the evaluation phase.
- **Large number of generations**: Sometimes, the number of generations will be large, requiring days or weeks to obtain a good solution.
- **Large population sizes**: With thousands of individuals. It is obvious that the larger the population the larger the time needed to evaluate the whole population.

Therefore, when facing real-world problems a mixture of the above reasons could lead to large times-to-solution (days, weeks or even months). One of the main employed methods to reduce the required time to obtain a solution is using a parallel version of the algorithm and running it on a parallel or a distributed system. Parallel and distributed systems are becoming widely used nowadays. These systems provide huge computing capacities for running complex problems that require too much time to be solved on serial machines. Currently, these systems tend to have a large number of microprocessors, which enables access to greatly increased computing power.

One approach is to use a cluster of desktop machines (or a desktop grid) as a physical platform for parallel algorithms. This platform employs commodity hardware for harnessing the computing power of desktop computers. These systems are becoming widely adopted as desktop computers nowadays have high-end hardware like dual cores microprocessors or large RAM memories and are widely employed all around the
world. For these reasons, different researchers are using these systems to research and run their experiments on physics, astrophysics, chemistry, mathematics, etc. Nevertheless, the reported use of dEAs in this context is currently very limited.

1.6 Contributions

This thesis makes the following contributions:

1. We introduce, compare and evaluate some simple dGAs and interaction schemes with a serial GA using a well-studied benchmark fitness functions to observe how these models behave when optimising different type of fitness functions across with a pre-defined value of GA parameters. In addition, exam the performance of these models with different number of subpopulations to see how the relationship represents the effect of the number of demes on the successful performance of the algorithm. The main finding is that a simple and little-explored basic topology for dEAs is shown to have robust and usually better performance than other topologies.

2. We test and analyse the behaviour of new parallel models with a new adaptive migration scheme. We estimate the overall quality and performance of the algorithms when adapting the individuals’ migration among the sub-populations, in which the frequency of migration events adapts dynamically in response to the current balance between exploitation and exploration. We also focus on high dimensional versions of a selection of hard function optimization problems.

3. We introduce a new method - 'diversity-guided control' - which adapts migration in a dEA according to the diversity of the demes. Again, by testing on a set of difficult problems, we show that this technique is a simple and effective addition to the field of dEAs. This action would be designed to lead to a boost in population diversity to enhance the performance of proposed models.

4. Finally, the quality and promise of the new dEA techniques examined in this thesis are further validated on a set of real-world optimization problems (from the IEEE CEC 2011 competition).
1.7 Document Outline

This thesis is broadly organised into introductory chapters and research chapters. The introductory Chapter II provides the background knowledge to put the research Chapters (III-VI) into context. Each research chapter also includes additional background knowledge specific to that chapter. The remainder of the thesis is set out as follows:

Chapter II provides a brief introduction to basic Genetic Algorithms. It defines some terms (parameters and operations) that will be used in the rest of the thesis; in addition, it explains how the simple GA works, and the main difficulties in term of balancing between exploration and exploitation of the search space. In addition, Chapter II describes parallel genetic algorithms, and parallelism’s levels and difficulties. It addresses some types of problems where genetic algorithms are applied for, different types of distributed genetic algorithms like global single-population master-slave dEAs, single-population fine-grained dEAs, multiple-population coarse-grained dEAs and hierarchical parallelization. Finally, it concludes some advantages and limitations of the master-slave model.

Chapter III presents the basic studied parallel models (dGAT1, dGAT2, and dGAT3) employing master-slave multiple-deme parallelization with a set of sixteen subpopulations. In addition, it gives some details about the master and client processes along with general parameters and attributes used in each model. Then it details the benchmark test functions that been used to test these models and algorithms. Also, it presents experiments on these models with sixteen and eight subpopulations.

Chapter IV introduces and explores an adaptive migration scheme to enhance the performance of the proposed models (dGAT1, dGAT2 and dGAT3) which have been previously mentioned in Chapter III. Firstly, it explores using the migration rate to control the search, and investigates how the migration/interaction mechanism can help to enhance the performance of the proposed models. After that, it introduces the new algorithms (AdGAT1, AdGAT2 and AdGAT3) that been applied on the same topologies. In conclusion, it demonstrates the experiments, results and discussion.

Chapter V demonstrates the usage of a diversity guided scheme to enhance performance. Initially, it gives a brief overview about how the population diversity can affect genetic algorithms, and how maintaining diversity can help to enhance performance. It also presents some studies that have been done with an intention to
maintain population diversity that allows rapid convergence and to avoid premature convergence.

Chapter VI tests and validates the studied models and algorithms that were investigated in Chapter III, IV, V (\(dGAT_{1,2,3}\), \(AdGAT_{1,2,3}\) and \(DGdGAT_{1,2,3}\)) using a range of real-world numerical optimization problems and compares the reached results with the results of the IEEE-CEC 2011 competition. The test benchmarks consist of 22 different test functions, detailed in technical report [50]. These test functions have different sizes ranging between single and 216 dimensions. The wide range of problems dimensions and complexity gives a good chance to examine our model more deeply to test each model with different types of real problems. The total number of competitive algorithms was 14 different algorithms and the winner of the IEEE-CEC 2011 competition was the algorithm called \(GA-MPC\) [69]. Finally, the Chapter VII ends with a brief summary and recommendations for future work.

1.8 Conclusions

In this chapter, we briefly introduced genetic algorithms, which are iterative techniques operating on a set of individuals composing a population; each of these individuals represents a potential solution to the problem. This population of individuals evolves due to the application of a set of operators inspired by biological processes of nature, such as natural selection and genetic inheritance. As a result, the individuals of the population are improved during the evolution. GAs are really useful tools for solving complex problems, as they work fast in large (and complex) search spaces thanks to their ability to process multiple solutions simultaneously. Hence, GAs can follow different search paths simultaneously, that would be in turn explored in parallel.

Additionally, it is possible to improve the numerical behaviour of the algorithm by structuring the population. In this study we focus on the distributed genetic algorithms as it was briefly introduced in this chapter. For parallel algorithms, this kind of coarse-grained model of search is exceptionally good and represents a rich set of research lines. We will see in the next chapters how the panmictic models allow for master-slave parallel execution.
CHAPTER II

LITERATURE REVIEW

“The Darwinian process of continued interplay of a random and a selective process is not intermediate between pure chance and pure determinism, but in its consequences’ qualitatively utterly different from either”.

Sewall Wright [259]

2.1 Evolutionary Algorithms Overview:

The Evolutionary Algorithm (EA) involves a set of methods and algorithms that simulate the general evolutionary methodology of Darwinian evolution which serves mainly like a heuristic guide to solve an optimization problem. These types of algorithms mainly manage a population of solution’s members using an objective (fitness) function to assess them. This will help in judging and representing the quality of solution. The search process to locate a solution is made through consecutive generations, evaluating the fitness of members and generating new offspring that depend on the individuals of recent generations, employing a set of genetic operators. The essential common methods and the relationship to neo-Darwinian evolution are illustrated in Table 2-1 as it is currently known; the field is classified into four main groups that are shown in Table 2-2.

The explicit details for each group may differ in important ways and are not the same in general. Each element of the EA family has a wide number of variants described in the literature and as such, the below generalization should be accepted with care. Even this kind of classification is not firm, mainly between the GA and EA as there are binary representations of the latter and real-valued genetics for the former. See DeJong 2002 for a unified background on EAs [53]. A major algorithm- the estimation of distribution (EDA)- of Pelikan and co-authors [188] is not fitting into this compartmentalization. The EDA rejects the idea of using a population of solutions and
uses a probability distribution instead, sampling from it then subjecting it to manipulation via the evolutionary process.

An alternate separation of these families may be classified into those that either utilizes a static representation or executable one. Evolutionary Programming and its successor Genetic Programming (GP), generally uses execution to evaluate candidate solutions that directly represent computer code, and by its nature this representation is variable in length. The EA as well as GP traditionally use a fixed length binary or numeric representation that must be evaluated by an external function.

Table 2-1: neo-Darwinian vs. Evolutionary Computation

<table>
<thead>
<tr>
<th>Neo-Darwinian Evolution</th>
<th>Evolutionary Computation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Organism</td>
<td>Candidate Solution</td>
</tr>
<tr>
<td>Population of organisms</td>
<td>Population of candidates.</td>
</tr>
<tr>
<td>Environment</td>
<td>Problem</td>
</tr>
<tr>
<td>Environmental fitness</td>
<td>Solution quality</td>
</tr>
<tr>
<td>Survival of the fittest</td>
<td>Differential selection</td>
</tr>
<tr>
<td>Genetic inheritance</td>
<td>Genetic inheritance</td>
</tr>
</tbody>
</table>

The research literature presents several particular sub-fields with variants of the mentioned EA families, sometimes based on particular applications. Some prominent examples include evolvable hardware [2] and artificial life [82]. For more details on Genetic Programming see [146, 192, 196] and see [66, 77, 78] for introductions to Evolutionary Programming. Also there exist many algorithms related to this field that do not fit into the above categorizations. Dorigo's research [62] on ant colony optimization (ACO) imitates the pheromone and foraging behaviour of insects. ACO uses a kind of population that collectively builds a shared probabilistic model of a solution instead of each member representing a specific solution.
Table 2-2: Evolutionary Algorithms' Categories

<table>
<thead>
<tr>
<th>Evolutionary Strategies</th>
<th>Established and published by Ingo Rechenberg in the 1960's. It used to search for real-valued systems optimization.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Genetic Algorithms</td>
<td>Established in the 1960's by John Holland. The mainly employed to search for optimization of bit-string and real-valued representations’ systems.</td>
</tr>
<tr>
<td>Evolutionary Programming</td>
<td>Set up in the 1960's by Lawrence Fogel and mainly applied to find evaluation of finite state machines.</td>
</tr>
<tr>
<td>Genetic Programming</td>
<td>Published and employed by John Koza in the 1990's over functional computer languages such as (Prolog and Lisp) to develop and enhance the parsing trees.</td>
</tr>
</tbody>
</table>

Another technique (like ACO, inspired by 'swarm intelligence') is called particle swarm optimization (PSO) [140]. In PSO, candidate solutions are called particles; each particle jeeps a solution memory, and moves about the search space, transmitting information about new discovered solutions to neighbouring particles. The study of Cutello and Nicosia on artificial immune systems (AIS) [47] describes another collection of algorithms which are sometimes grouped with EAs. The main objective of AIS algorithms is to mimic attributes considered in biological immune systems to solve problems. The EA also can be classified as a part of the wider family of stochastic local search (SLS) algorithms. They are usually applied to both decision problems and optimization problems. For more detail on SLS see the excellent text by Hoos and Stützle [124].

2.1.1 The Simple Genetic Algorithm (SGA)

Genetic Algorithms (GA) as proposed by Holland [122] can be defined as a set of populations utilizing a randomized exploration methodology. These are increasingly being used in a large number of practical problems. Such algorithms generally use populations (also called pools, or collections) to keep and maintain a collection of chromosomes (individuals); these individuals can be probable solutions to the current problem being processed by the algorithm. Canonical version of Genetic Algorithm
works on a binary bit string of length L which corresponds to the problem encoding with three operators which are selection, mutation and crossover. New points (bit strings) are continually produced from the search space, are then evaluated, and may be embedded into the new population.

This population is used to provide GA algorithms with a way of defining a non-uniform probability distribution function (p.d.f.) guiding the generation of new points in the search space. The general mission of the p.d.f. is to reflect all possible interactions between population points, coming from the “recombination” from two (or more) points of the population (parents). This is in contrasts to the globally uniform distribution of blind random search (BRS), or the locally uniform distribution which utilized by a lot of other stochastic algorithms such as simulated annealing (SA) and various hill-climbing algorithms (HC). By supplying a fitness (objective) function, the GA can connect a fitness value to every point in the problem’s search space. Thus, a unique fitness value will be connected to every point in the problem’s search space, and as a result, the search space can beneficially be imagined as a “fitness landscape”. The simple Genetic Algorithm can be generally formulated as:

\[
GA = (P^0, I, \delta^0, \mu, \lambda, l, F, G, U)
\]  

(2-1)

Where:

\[
P^0 = (a_1^0, \ldots, a_n^0) \in I^\mu
\]  

(2-2)

\[
I = \{0, 1\}^l
\]  

(2-3)

\[
\delta^0 \subseteq \mathcal{R}
\]  

(2-4)

\[
\mu \in \mathbb{N}
\]  

(2-5)

\[
\lambda \in \mathbb{N}
\]  

(2-6)

\[
l \in \mathbb{N}
\]  

(2-7)

\[
F : I \rightarrow \mathcal{R}^+
\]  

(2-8)

\[
G : I^\mu \rightarrow I^\lambda
\]  

(2-9)

\[
U : I^\mu \times I^\lambda \rightarrow I^\mu
\]  

(2-10)
The Searching process for a problem’s solution using genetic techniques can be seen as a iterated process of two tasks. The first task is “Generating” a new group of nominee points which they called offspring. This is done in a probabilistic way consistent with the p.d.f. defined by the action of the chosen recombination operators (mutation and crossover) on the original population, which are called parents. The second task is “Updating” the population, this usually done by evaluating each new point using the fitness function, followed by applying a selection strategy to the union of the offspring and the parents. The key objective of the genetic algorithms is to explore the binary search space to optimize the fitness function to be either minimized or maximized

\[
f(x) : \{0,1\}^n \rightarrow \mathbb{R}^+ \tag{2-11}\]

Genetic algorithms are commonly applied to complex problems that are poorly understood, which contain high level of interdependencies, or otherwise are excessively complicated to be optimized with standard methods. Once the problem has been successfully encoded into a pseudo-Boolean function, the GA process can be established to generate and evaluate candidate solutions through a simulation of the evolutionary process.

The following algorithm gives a general overview of how a GA process works:

a. Generate initial population randomly of \( \mu \) candidates of length \( l \).
b. Evaluate each individual’s fitness using provided fitness function \( f(x) \).
c. From current population, select pairs of individuals as parent using \( P_r (a_i^I) \).
d. For each selected pair do:
e. Utilize recombination probability \( P_r \) to affect operator to the parents to produce an offspring.
f. Use probability \( P_m \) to mutate offspring.
g. Add new offspring to the update population.
h. Swap the old population with the new one.
i. Until a termination condition, return to step b.
2.2 Parameters and Operators

The next sections are summarising the most important operators of evolution as considered by EA researchers.

2.2.1 Population Size

Population size can be seen as a critical primary choice. Simple intuition shows that population size should be selected with regard to the chromosome length, since in a small population with large chromosomes; many of the genes in the chromosomes will only have one value represented in the population. A large population size might result in squandering processing time with many unnecessary function evaluations. However a small population size can lead to ineffective exploration of the search space.

The first attempt to analyse this key question was by Goldberg and his co-authors through a series of papers [91, 94, 95, 111]. Initially, population size, as they concluded, should expand as an exponential function with regard to chromosome length. While another empirical works of Grefenstette [102] and Alander [6] had showed acceptance of linear relationship. Reeves [199, 201] provided a remarkable result to determine the minimal necessary population size of GA application, using a simple idea based on a theoretical background. The precept that been adopted in this case was to make all possible loci in the search space accessible through the original population by applying crossover operator only. If the population size is large enough to achieve this in a random initial population, it would be theoretically possible to discover any point in the search space without depending on mutation. The conclusion of Reeves' work suggested an \( O(\log l) \) formula to calculate the minimum population size to meet these criteria.

Mühlenbein & Schlierkamp-Voosen [175, 176] and Cvetković & Mühlenbein [48] through their empirical work on monotonic fitness functions (like the One-Max problem) derived a similar formula \( O(\sqrt{T\log l}) \) as a recommendation for the computation of GA population size. Hansen and his co-workers [110] provided an important theoretical study of an adaptive offspring population size in \((1, \lambda)\)-Evolution Strategies with respect to the local progress. Herdy [116] worked on maintaining offspring population size in a discrete search space. He claimed that Hansen et al's [110] approach is fraught with difficulties when run on discrete search spaces. His idea used a different adaptation rule which trying to keep the number of offspring that outperform their parent roughly equal to 1.
Another work by Gao [83] presented quantitative characterization and theoretical analysis results of a linear lower bound on effective population size. He conclude that a large enough population size $>> l$ is not a good choice in general. In contrast to that, another assumption is widely believed by De Jong [53] and Reeves [202] saying that the allele diversity maintained through a 'large enough' population size has many intuitive benefits. Though both ideas suppose that allele diversity is beneficial to have and maintain.

2.2.2 Selection Operators

The EA selection operators can be categorised into two major classes, survival selection and reproduction selection. These operators rely on the evolutionary theories of biology, and it’s very useful for the general discussion of context to state which operator is being handled.

Selection for Survival

The Survival selection can be described as a mechanism to choose the population individuals of the next generation from current population of parents and offspring. The traditional selection method supports the idea that the current population should be completely replaced by the new population. De Jong named this survival methodology as the generation gap [52]. He also introduced other concepts in survival selection such as élitism, population overlaps and steady-state GA.

The élitist approach grants survival to the best fitness individualy, keeping it and after that replacing the remaining ($\mu - 1$) the population individuals with new offspring. While population overlaps was designed to replace only a fraction $G$ of the population at each new generation, Bulmer [32] & Crow and Kimura [46] stated that this selection scheme is frequently used by breeders and in population genetics. Finally, the steady-state survival strategy, Mitchell [169] refers to it as “steady-state selection”, completely merges the current population individuals with new offspring and then selects the best $\mu$ as the new population for the next generation.

The reader can easily notice that this sets up a range of parental survival schemes. The generational (canonical) GA is designed with a zero level of elitism with no parental survival. While the steady-state GA expects full parental survival unless a child overcomes some individuals of the current population.
Selection for Reproduction

Fitness-proportionate or “roulette-wheel” selection as introduced by David Goldberg [93] is the most popular reproductive selection strategy. The idea that stands behind this selection scheme comes from the action of a rotating roulette wheel in a casino. The circle divisions of the roulette wheel are assigned proportionate to the fitness level of each individual $f_i$. This could be managed by dividing the fitness of a selected individual by the total fitness of all individuals, thus normalizing them to 1:

$$ P_i = \frac{f_i}{\sum_j^\mu f_i} \quad (2-12) $$

The scaled selection or “sigma selection” scheme as illustrated by Baker [24] is generally considered as the roulette-wheel method integrated with a strategy to normalize the weights along with fitness shape of the population. The basic idea is to keep ‘selection pressure’ (the tendency of highly-fit chromosomes to get reproduction opportunities) constant during the running of the EA. It would be good idea to allow algorithm to select less fit individuals at the early steps and then to increase the selection pressure to choose highly-fit individuals as GA is approaching to the end. Thus, at the beginning the GA has a wide search space to select individuals, but after some generations this search space is shrunk by increasing the pressure to select better individuals. This selection strategy helps to eliminate early domination of highly-fit members and to increase the level of late selection pressure.

The “ranking selection” was first introduced by Grefenstette & Baker [104] and Whitley [256] to avoid the critical disadvantages of proportionate selection schema, the probability to choose is based upon a simple ranking of the population by fitness value. Ranking strategy assign a rank $R_i$ to population individuals after sorting them according to their fitness values, where the rank $\mu$ is granted to the best one and the rank 1 is to the worst individual. The selection probability guided by ranking strategy is achieved using linear or exponential schemes. Once more, the main objective is to avoid domination of breeding opportunities by highly-fit individuals [169]. For example, the
probability of selection via the linear ranking strategy [24] for each individual according to its rank can be assigned using the following probability function:

\[
P_i = \frac{1}{\mu} \left( \eta_{\text{min}} + (\eta_{\text{max}} - \eta_{\text{min}}) \frac{R_i - 1}{\mu - 1} \right); \quad i \in \{1, \ldots, \mu\}
\]  

(2-13)

where \( \frac{\eta_{\text{min}}}{\mu} \) is the probability of selection of the worst population member and \( \frac{\eta_{\text{max}}}{\mu} \) is the probability of selection of the best individual in population.

### 2.2.3 Mutation Operator

The mutation operator is sometime considered by scholars as the most basic operator in the EA. It is derived from the original mutation noted in biological genetics caused by natural changes in chromosomes and different transcription effects observed in early evolutionary studies. The genetic algorithms mutation was considered as a background operator with small importance [123]. The mutation operator main objective is to enhance the diversity of the population individuals and to support the fine-tuning capacities of the algorithm.

This mutation operator is usually applied with a low probability, there have been number of studies to find the optimum mutation probability level in GAs, but unfortunately, their results as well as the recommended values vary, leaving practitioners in the dark. The most popular mutation technique for bit strings is to use a bit-flip probability, which is used against each bit independently. Therefore the actual number of bits changed in an individual is not fixed.

DeJong [52] suggested \( P_m = 0.1\% \), but the meta-level GA that was employed by Grefenstette [102] pointed to \( P_m = 1\% \), while Schaffer et al. [212] came up with \( P_m \in [0.5\%, 1\%] \). Bremermann through his work [31] derived the global optimum mutation probability as \( P_m = 1 - \left( \frac{m}{l} \right)^{\frac{1}{l-m}} \) where \( m \) represents the number of bits that are correct out of \( l \) individual size. Bäck [20] shows clearly that the rate of optimization of a GA is mainly affected by the mutation probability. In addition to that, he pointed out that the mutation rate should not be constant, as typically implemented in GAs.
Mühlenbein [174] stated a mutation rate formula $P_m$ which is generally considered as "optimal" for a static mutation probability, and which depends only on the individual's length $P_m = \frac{1}{l}$. Smith and Fogarty [224] have studied and compared this rate with other static mutation values, and founded that it outperformed other values. Bäck [19] also found this formula to be a good value of mutation probability.

However, many researchers have studied some type of adaptive mutation. As an example of that, Fogarty [75] investigated many different mutation rates at different loci, he employed deterministic control schemes decreasing over time and over the loci, and then concluded that the mutation rate should decrease according to the number of generations. Another example is Reeves study [200], that changed the probability of mutation according to the diversity in the population. Whenever the population diversity became too small, the mutation probability will be increased to a pre-defined high value but was allowed to reduce steadily to a minimum level at which it stayed until population convergence was again noted.

2.2.4 Crossover Operator

The Crossover mechanism can be considered as the most complex process of all EA operators [203]. Sometimes this operator is mentioned in the literature as the recombination operator. The crossover mechanism is inspired by the genetic recombination operator that is active during meiosis in sexual reproduction [169]. However, the crossover processes elated to biological organisms are quite different from the EA crossover operator.

Usually, crossover is done with some probability which is called the crossover rate. In case of not applying the operator to the selected parents, they will be copied to the next round's population without any amendments. This will help in preserving the good individuals from one generation to another. The crossover rate can aggressively change the direction of the search. High crossover rates can produce and generate more children along with the possibility of losing many good chromosomes in the population. On the contrary, low crossover rate leads to maintaining good individuals and transferring them to the next generation, resulting in additional exploitation. DeJong [52] proposed crossover rates that might lead to good performance; he suggested 0.6 as a reasonable rate to use in general.
Many different types of crossover have been investigated for EAs, including One-Point Crossover, Two-Point Crossover, M-Point Crossover, Uniform Crossover and Adaptive Crossover [72, 122, 158, 169, 228, 229, 238, 240]. Please see Appendix A for more details.

2.3 General Difficulties

In reality, Evolutionary Algorithms EAs generally experience some critical difficulties. The EAs is usually defined as a kind of algorithm that endeavours to impose a form of balancing between exploration and exploitation of the search space. When an EA is found to be exploiting a specific area of the search space, with exploration having no effect, this is a common defect which is called premature (early) convergence. The disadvantage of premature convergence is related to the suppositions that the search has stopped prematurely, and that there are better solutions available in the search space, but the current search is not able to improve on the best solution found so far.

Another difficulty of algorithm is may not adapt to the unique conditions of the search space. For example, if the algorithm is climbing a hill towards a local optimum, then the EA will success to choose a suitable step-size to climb the hill; this step-size can be represented by mutation rate for each real number in the real-values representation. However, each part of search space may require different best step-size. Thus, the set parameters of the EA determining step-size may be ill suited to the fitness function at hand [203].

The non-elitist EAs may also fall into the trap of the lack of persistence; this means the reached solutions in such a generation may be lost in next generations. While the steady-state and elitist approaches have the ability to deal with this problem at first instance, but there is no optimal strategy treats this matter in general. The lack of persistence is not common difficulty for fitness functions. The fitness landscape possibly modular and the eventual loss of a particular genetic attribute may not lead to fitness trap. However, this type of action may prevent the process in the future from merging that attribute with others. Examples of this instance can be found in [128], where a non-elitist EA can perform better than an elitist one.

Another critique of EA performance is the apparent incapability to deeply deal with the nature of the search space or even appropriately employ human knowledge of a problem. Nevertheless, as this defect has reasonable truth, there are several researches
and practices which blend EA algorithm design with human knowledge and judgement [68, 187]. In fact, EAs has undergone several specifications and changes which have been applied by adding extra operators or even by tuning features of the main operators. These specifications can practically be helpful, but unfortunately the majority of these results are still far away from generalizing. Stephens [232] criticised an EA conference which had explosion of papers with titles like "A New Genetic Algorithm for the Multi-Resource Travelling Gravedigger Problem with Variable Confin Size." Given the propensity of copy-cat usages of algorithms these specializations may actually be harmful when applied blindly.

2.4 Distributed Evolutionary Algorithms (dEAs)

EAs can be successfully applied to multiple types of problem and find a reasonable solution in reasonable time, but unfortunately for some complex problems they may take ages to get acceptable solution. In fact, many efforts have been made to make EAs search faster to reach a good solution in good time. One of the most promising alternatives is to employ parallel implementation. The parallel nature of evolutionary algorithms has been recognized for long time, and many have used distributed EAs to reduce the required time to get reasonable solutions for complex problems. Evolutionary algorithms work with populations of independent solutions which make it naturally be easy to distribute the computational load among several processors. In fact, an EA is “embarrassingly parallel” which make it very easy to be efficiently implemented on parallel computers. However, despite their operational simplicity, distributed EAs are complex non-linear algorithms that are controlled by many parameters that affect their efficiency and quality of their search. Setting these parameters correctly is crucial to obtain good solutions quickly and reliably.

In particular, the design of distributed EAs involves choices such as using single or multiple populations [38]. In both cases, the size of population must be decided carefully, and for multiple population it has to decided how many to be used. In addition, the populations may remain isolated or they may communicate by exchanging individuals or some other information among them. Number of existed techniques has the capability to actively utilize huge parallel computer architectures, while other methods are keen to use multi-computers technique with less number and extra powerful processing components. Erick Cantú-Paz [38] defined three main types of distributed EAs:
Global single-population master-slave dEAs.

Single-population fine-grained dEAs.

Multiple-population coarse-grained dEAs.

In **master-slave dEAs** (Figure 2-1a) there is a single population, however, the whole evaluation process is distributed through multi processors. This type of dEA is sometime known as a globally distributed EA where the selection and crossover operators consider the whole population.

The **Fine-grained dEAs** (Figure 2-1b) are designed for huge and powerful parallel architectures. Only one spatially-structured population is used in this type of dEA, and the selection and mating process are restricted to a small number of individuals, however, the overlapping and sharing between neighbourhoods allowing some interaction through other individuals. The key point is to allocate a single individual for each processing element in this structure.

The **Coarse-grained (multiple-deme) dEAs** (Figure 2-1c) are more interesting. It consists of more than one subpopulation or (demes), which occasionally communicates individuals among them. This type of communication is known as individuals’ migration and, it is controlled by several parameters. Coarse-grained EAs are very common, but since the effect of migration are still not fully understood makes this class of distributed EAs is the most difficult to be predicted. Multiple-deme dEAs introduce fundamental changes in the operation of the EA and has a different behaviour than simple EAs. Multiple- deme distributed EAs are sometimes called “distributed” EAs, because they are usually implemented on distributed memory MIMD computers. Since the computation to communication ratio is usually high, they are occasionally called coarse-grained EAs.

Lastly, multiple-deme EAs resemble the “island model” in Population Genetics which considers relatively isolated demes [115], so the distributed EAs are also known as “island” distributed EAs. Since the size of the demes is smaller than the population used by a serial EA, it will be expected that the distributed EA converges faster. Therefore, while it is true that smaller demes converge faster, it is also true that the quality of the solution might be poorer.
Figure 2-1: Parallel Evolutionary Algorithms dEAs
a) master-slave dEAs; b) fine-grained dEAs; c) multiple-population dEAs
2.4.1 Master-Slave Parallelization

Only one single population is used in this type of dEA and the application of genetic operators and fitness evaluation are processed in parallel. Any individual in this single population has a chance to compete and mate with any other, this lead to consider selection and mating are global. Usually, the Global distributed EAs architectures are designed to be implemented as master-slave applications; this design allows the master terminal to manage and track the population while the slaves evaluate the individuals. The fitness evaluation is the easiest and most popular parallelized operation, that because the fitness of an individual can be done independently from other individuals in the population, and there is no necessary communications among them during this process. The parallel evaluation is occurred by making each available processor calculate the fitness for a single part of the population, thus the expected communication in this phase is only done when each slave receives a portion of population form the master to be evaluated and also when the slave returns the fitness values.

Two different types of master-slave parallelization can be implemented synchronous and asynchronous dEAs. The synchronous form is done when algorithm does not proceed into next generation until it gets all the fitness values for the entire population, the only one advantage for this type of EA is the processing speed while all other properties are exactly similar to the simple EA. However, the other available implementation of master-slave EA is an asynchronous version; this type of algorithm does not work in the same way of simple EA, and does not affected with any delay from any slow processors. The flexibility of the global parallelization technique does not recommend any specifications about computer architecture, thus, it can practically be implemented on different types of parallel machines (shared and distributed memory). On shared-memory parallel machine, each processor can actively access (read and write) the algorithm’s population as it is stored in the main (shared) memory.

However, the population on a distributed-memory multiprocessor can be stored in the “master” processor. This processor has the responsibility of sending and selected individuals to the other processors “slaves”. The slaves then evaluate and apply the genetic operators on these individuals and return them back to master processor to form the population of the next generation. In [60] Desell et al. examined different strategies for computing EAs using simulated computing environments. The researchers tried to simulate and represent computing clusters or supercomputers by using a simple homogeneous environment with various communication latencies. They aimed, by
using this environment, to compare the scalability of sequential or parallel EAs to asynchronous EAs and examine the effect of heterogeneity on asynchronous algorithms. The experimental results clarified that asynchronous EAs can scale to hundreds of thousands of computing hosts while being highly resilient to heterogeneous and faulty computing environments, which is sometime not available for traditional distributed EAs that require synchronization. Also, they showed that the asynchronous EAs required a less fitness evaluation rate to reach the solution when the environment’s heterogeneity increased. A similar work is presented in [84], the authors implemented EAs on distributed environment using cloud services. This method helps in avoiding the acquisition of expensive resources. But also it raises another problem of select between different services at different levels and actual scalability obtained by distributed evolutionary algorithm. To overcome this problem, they used a pool-based evolutionary algorithm and they concluded that this technique allows to use the cloud service more efficiently and dynamically than a static or synchronous service.

In [74], Fogarty and Huang tried to develop some rules to maintain a pole balancing method to reduce the processing time. The researchers utilized a network of parallel microprocessors “transputers” which are designed specifically to do parallel computations. Each node “transputer” in this network can connect to only four other nodes and communication between the connected nodes is controlled by retransmitting tokens. This topology and communication strategy causes an overloading in communications; this made Fogarty and his co-researcher used different topologies to connect the transputers in an attempt to overcome this problem. Their results showed that tuning the network topologies and configurations not enough to make an important difference. They successfully gained acceptable speed-ups over the serial algorithm, but also showed that the communication overhead can critically obstruct further possible improvements in algorithm’s speed.

Another work was done by Abramson and Abela [3], they implemented their algorithms using a shared-memory computer of 16 processors to solve time table problem, but their work obtained small speed-ups. They attributed the reason of some parts of serial code on the critical path of the program which prevent a good speed-up in the parallel method. Another related work was done by Abramson et al. [4]. The authors used distributed-memory architecture with 128 processors to do their experiments. The practical results showed significant speed-ups for up to 16 processors on the two computers; however the speed-ups were significantly decreased as more processors are
joined the system because of increasing in communications. Hauser and Männer [114] employed three different types of parallel architecture; they only success to achieve good speed-ups on the architecture of the lowest communications overhead.

Another possible parallel application of EAs is that can be done by using parallelisation over the genetic operators (selection, crossover and mutation). The crossover and mutation operators can be parallelized using the same way of distributing the population over multiple processors. However, as these operators are so simple, the expected efficiency of this work will be limited as the required communications to do this job will require additional time over doing the same work on the same processor. The parallelization of selection will also be obstructed by communication overhead, this because many types of selection methods require additional information about the entire population, which also require extra communications between processors to achieve this work. Salto et al. [209] analysed the impact in the performance of a parallel algorithm when it is executed over a set of heterogeneous computing architectures, where each of these computers running at different clock speeds. They concluded that the solution quality and efficiency that achieved by heterogeneous computing environment is similar to accuracy and efficiency of homogeneous one, even using a lower number of fitness evaluations.

We can conclude that master-slave distributed EAs is a very efficient method and it can be easily implemented over different parallel architectures. Also, it has the advantage of not altering the search behaviour of the traditional EA, so we could still apply directly all the theory available for simple EAs.

2.4.2 Fine-Grained Parallelization

This type of dEAs deals with a single population, and it consists of a special structure which control and reduce the interaction between population elements. In this structure, the interaction is only allowed between the neighbours. However, as the neighbourhoods exchanging good solutions, this make the individuals have a chance to interact with entire population.

Merelo-Guervos et al. [166] parallelized the evolutionary algorithm “SofEA” in conjunction of using a special database “CouchDB” to store population elements. The authors explained how this formula can create a framework for distributed scalable asynchronies systems. This system puts evaluation and reproduction operations in
different computing nodes. They explored the client configuration space and concluded that there are certain configurations can be done to double or even triple the algorithm’s speed-up over the sequential version. In [34] Cantor and Gomez used the fine-grained EA over 2-dimensional cellular automaton surface (grid) and made each individual on this grid has two different state (active, inactive). Every step of the evolutionary process, the individual state is changed along with the cellular automaton state rules and only active individuals have a chance to be selected for evolution process by applying one of the genetic operators and considering just their active neighbours when more than one individual is required. The researchers in [34] aimed to achieve optimal or near optimal solution in a proper time by maintaining the population diversity and by applying a massive extinction operation in a random fashion to prevent undesirable growth in the population size.

In [135] a fine-grained distributed EA model was proposed using the evolvable agent (EvAg) model over a massive parallel Peer-to-Peer hardware. In this study, two main responsibilities were assigned to each EvAg; evolves a single individual with the mate selected locally within a neighbourhood, and to dynamically structuring such a neighbourhood by means of the protocol newscast. This technique makes the approach suitable for a P2P execution in which every EvAg can be potentially placed in a different peer. The results showed that the algorithmic performance of the (EvAg) model is competitive with respect to a canonical approach. Furthermore, times to solution are significantly shrunk at the fine-grained parallelization of individuals in the population and outperformed the sequential approach in several orders of magnitude.

In [162], researchers proposed a fine-grained parallel GA and distributed the population over a 2-dimensional grid. The genetic operators (selection and mating) were restricted within a neighbourhood. The authors noticed that the performance of the algorithm changed according to the size of neighbourhood, and they noted that the bigger neighbourhood is the less algorithm performance. They concluded that if the size of the neighbourhood is near to population size then the parallel GA will act like a single panmictic population. In [210], Sarma and his co-researcher applied the selection methods for sequential EAs over local neighbourhood fine-grained models, and analysed how the neighbourhood size and shape can affect the selection mechanism. Moreover, the authors presented a quantitative model to analyse this relationship. The experimental results showed that the critical parameter is ratio of the radius of the
neighbourhood to the radius of the underlying grid. This parameter directly controls the selection intensity over the global population. Also, the results indicated that the growth rate of the best solution across the whole population is shown to be an inverse exponential function of this ratio.

In fine-grained dEA, the traditional way to store population elements is in a 2-D grid; this will be very efficient in massively parallel architectures as the processors will be directly connected to these elements. However, most super parallel computers utilize a global router, which is responsible to control and supervise the communication among the processors in the grid. In [219], Schwehm implemented a fine-grained parallel genetic algorithm on the MasPar MP-1 computer, which is a massively parallel mesh architecture, in intension to investigate the impact of changing the topology of the environment of the individuals on the performance of a genetic algorithm. These networks were emulated using a global router and 1024 processing elements. The authors used a graph partitioning problem to test different population structures of this parallel environment. He tested this problem over five different topologies, each of which has different number of dimensions and processing elements. He concluded that, the algorithm with the two dimensional topology (torus) converged faster than the other algorithms as the mating partner is found in the local neighbourhood, but there is no mention of the resulting quality.

In [148], Li et al. used Graphic Processing Units (GPU) to optimize some fine-grained multi-objective EAs (MOEAs) along with some new parallel crossover computation schemes. These schemes are based on ordinal and sequential representations. The researchers in this study used a binary representation in MOEAs, and noticed that the fitness evaluation can be transformed a parallel matrix multiplication which can be easily and more efficiently implemented on GPU. The empirical results showed that the use of GPU-based parallel fine-grained EAs could achieve up to 14x times speed-up than that of CPU-based. The authors in [180] used a simple parallel genetic algorithm over MapReduce parallel architecture and aimed to find the optimal conformation of a protein folding problems using the two dimensional square hydrophobic-hydrophilic (HP) model. They found that the main problem of such architecture is the memory management which could arise for large sets of data and could result in overflow. One suggestion to overcome this problem was to move the
contents which are least recently used into files with the help of using a hash table as a tracker of all movements.

Some studies compared between fine-grained and coarse-grained EAs [25, 41, 98] using a wide range of optimization functions. These comparisons showed that the coarse-grained EAs sometime outperform the fine-grained EAs, while sometimes else the situation is just inversed. This can be explained that the comparisons between these two kinds cannot be done in absolute terms, instead, we must focus on the main objective either to minimize (e.g., processing time) or maximize (e.g., the solution quality) and to carry out the comparison on a specific criteria. Gordon et al. [99] analysed and compared three different types of parallel architectures (global population, master-slave and massively parallel models). He theoretically showed that the critical path length of each model is independent of population size, i.e. the increasing of population size in the ideal environment will not increase the processing time of parallel models. Also, he mentioned that the critical path of fine grained EAs is shorter than that of a coarse-grained master-slave EAs. This means that with more processors available in massively parallel EAs would decrees the processing time to finish their execution, regardless of the population size. However, it is important to note that his study did not have any further considerations like the memory requirements or communications bandwidth. In a connected research, Gordon [97] tried to quantify spatial locality of memory references in several genetic algorithms. He utilized two different approaches to measure the locality, the first one is the percentage of remote references (for parallel computers with a few processors), while the second one is traffic per link (for massively parallel computers). He concluded that the ability to measure locality may ultimately allow us to compute trade-offs more exactly and it could be useful to determine the most adequate computing platform for each model.

2.4.3 Multiple-Deme Coarse-grained Parallelization

The important characteristics of multiple-deme Coarse-grained dEAs are the use of a few relatively large subpopulations and migration. The early work of multiple populations with parallel EA could be assigned to Grosso’s dissertation [105]. The author simulated the interaction of some parallel sub-populations. The whole population was divided into five subpopulations (demes), where each one exchanged their solution elements with all the other demes with a fixed migration rate. In a similar work, Pettey et al. [191] made their algorithm communicates the best individual found in each deme
and send it to all others in every generation. The authors intended to keep a good mixing of good individuals among the subpopulations.

Jakobović et al [129] designed two different asynchronous models of parallel evolutionary algorithms and compare their behaviours with explicitly synchronous generational global parallel EA (SGenGPEA). The authors aimed to reduce of idle time in worker processes with respect to the synchronous version, considering that some of the evaluations performed by the workers may certainly be wasted. The experimental result showed that the overall speedup is comparable to that of the synchronous algorithms. However, these results associated with the problem’s type as the speed-up changed when the problem architecture changed, which sometime makes the synchronised model overcome the asynchronous one.

2.4.4 Hierarchical Parallelization

Some researchers have tried to merge between two different methods of parallel GAs to produce hierarchical PGAs. Some of these new hybrid methods increase the complexity level to the already complicated prospect of parallel GAs, while other hybrids algorithms could be able to reduce the complexity as one of their components.

When two methods of parallelized GAs are combined they form a hierarchy. The most of the hybrid parallel GAs use the multiple-population algorithms on the upper level while some other hybrids have a fine-grained GA at the lower level (see Figure 2-2a). Gruau [106] presented a “mixed” parallel GA, he used 2-dimensional grid as a core for each island and the connection between all islands were formed as a 2D torus. The migrations between these islands occurred at a fixed rate of iterations and the best individuals from each island were transmitted to a neural network architecture and training patterns. A ring topology (Asparagos96) was introduced by Gorges-Schleuter [100] which is considered as an updated vision of the original asparagos [101] with a ladder structure. The new topology has a longer diameter and allows a better differentiation of the individuals. The new topology consisted of several demes where each of this deme was structured as rings. The migration between subpopulations occurs only when a deme converges, by receiving the best individual from another deme. After all the subpopulations converge, the algorithm takes the best and second best solution member of each subpopulation to form the population for the last run.
Lin et al. [149] employed a multi-demes GA with spatially structured subpopulations with a ring topology. To connect the subpopulation, the authors made ring topology very sparse with a long diameter, and each deme was constructed as a torus. The researchers in [149] compared their hybrid algorithm against different topologies with different number of demes (fine-grained GA, a ring topologies uses 5 demes with variable population sizes, and another ring topology with a constant population size but with different number of demes, and simple GA). To test their work they used a job shop scheduling problem as a test function. The reached results showed that the simple GA could not reach solution in reasonable time like the other algorithms, and adding extra demes to the multiple-population GAs enhance the algorithm’s performance better than increasing the size of the whole population. However, the hybrid topology found the best solutions overall.

Hierarchical parallel GA could also be formed by constructing a master-slave method on each deme of multi-population (see Figure 2-2b). In this type of hierarchical PGA, migration occurs between demes and the individuals' evaluation is done in parallel. This approach does not introduce new analytic problems, and can be helpful in solving some complex problems which need a considerable time to be optimized. The authors in [28] gave an instance of this type of hierarchical method, and showed that it can reach the optimized solution of the same quality of a master-slave PGA or a multi-population GA in less time. Also, the authors in [237] presented another example of master-slave parallel genetic algorithm (PGA) used a triplet based architecture (TriBA) topological structure to enhance the computation speed and decrease the communication costs using multi-core processors; every node in ground-grade is connected with other three nodes around. A single node is connected with three communication links, like a triplet. They used $k$ as a number of levels (sub-populations), thus the number of nodes will be calculated as $3^k$. See Figure 2-3.
Figure 2-2: Hierarchical Parallel GA;  
a) multi-deme + fine-grained,  
b) multi-deme + master-slave, c) multi-deme + multi-deme
The authors in [205] presented a hybrid genetic algorithm combining two shared and distributed memory algorithm based on the island model using Open-mp and MPI libraries. The authors aimed to increase the performance speed-up by using of two different parallelization strategies, the Global Parallelization and the Coarse-Grain techniques which can run on high-performance cluster computers. They studied the effect of migration on the performance and reliability of the algorithm and used different migration rates for this purpose. They found that the optimization time per subpopulation increases as the migration becomes more frequent, i.e. there is an increase in the communication time.

Another approach of hybridizing PGAs is to employ multi-population GAs at both the upper and the lower levels of algorithm (see Figure 2-2c). The main idea of this approach is to make high migration rate on the lower level with a dense topology, while it uses a low migration rate on the high level [40]. The complexity of this method would be equal to a multiple-population PGA if the lower level subpopulations considered as a single deme. The execution time to solve such a problem can be significantly less with using hierarchical methods than solving the same problem with any their components alone. For example, if we considered that the optimal speed-up of a multiple-deme PGA is $Sp_{md}$, and the optimal speed-up of a master-slave PGA is $Sp_{ms}$, then the overall speed-up of a hierarchical PGA that combines these two methods would be $Sp_{ms} \times Sp_{md}$.

![Figure 2-3: The [237] TriBA’s structures $k \in [0,3]$](image)
2.5 Master-Slave Model Advantages and Limitations

An important observation on master-slave GAs is that as more processors are used, the time to evaluate the fitness of the population decreases. But at the same time, the cost of sending the individuals to the slaves increases. This trade-off between diminishing computation times and increasing communication times entails that there is an optimal number of slaves that minimizes the total execution time. A recent study [36] concluded that the optimal number can be calculated as:

\[ S = \sqrt{\frac{n T_f}{T_c}} \]  

(2-14)

Where \( n \) represents the population size, \( T_f \) is the time it takes to do a single function evaluation, and \( T_c \) is the communications time. The optimal speed-up is 0.5S.

A natural starting point in the investigation of multiple-population GAs is the size of the demes, because the size of the population is probably the parameter that affects most the solution quality of the GA [90, 96, 112]. In [39], the authors developed two bounding different coarse-grained parallel GA topologies. The first one was completely isolated demes while the second one was fully connected demes. The migration rate in the second topology was set to the highest possible level. They aimed by using these two different topologies to predict the convergence quality of a simple GA and to show how the probability that the parallel GA finds a solution of the minimum desired quality increases as more demes are used. They tested their topologies using two test problems; the one-max function with \( m = 100 \) bits and the trap function with \( m = 4 \) bits. Finally, they concluded that the deme size with considerable level of migration should be much smaller than when the demes are completely isolated.

Arnaldo et al. [13] performed a systematic analysis of the correlation between island topologies and problem structure. They found that the link between island topologies and problem structures is highly complex, and concluded that there is no single island topology can be the best for different types of problems i.e. different problem structures require different island dGAs’ topologies.

The population size is also considered as a key value that can widely affect the execution time of PGA. Therefore, some researchers, like in [254], use deme sizing
models to predict the execution time of the PGA, and compare it with the execution
time of serial GA to find a same quality solution of same problem. The researchers in
[40] used two different models in their research to predicted the speed-ups for the two
bounding cases, one to control the population size and another to monitor the
communications time. The practical results showed that the speedup is not very
significant for completely isolated deme, while it enhanced with an accepted level of
communication between demes. However, they also showed that there always an
optimal number of demes and an optimal population size that maximizes the speedup of
any PGA.

Parallel GAs are very complex and, of course, there are many problems that are
still need more investigations. A few examples are: (1) to determine the migration rate
that makes distributed demes behave like a single panmictic population, (2) to
determine an adequate communications topology that permits the mixing of good
solutions, but that does not result in excessive communication costs, (3) find if there is
an optimal number of demes that maximizes reliability.

2.6 Conclusion

In this chapter a general overview of evolutionary algorithms was given, and a
simple formalisation was developed to describe optimization problems and associated
operators. Also, the main difficulties were discussed, in terms of balancing between
exploration and exploitation in the search space. In addition, this chapter reviewed some
important issues of the most representative work on parallel genetic algorithms. The
review started by classifying the work on this field into four categories: global master-
slave parallelization, fine-grained algorithms, multiple-deme, and hierarchical parallel
GAs. Some of the most important contributions in each of these categories were
analysed, to try to identify the issues that affect the design and the implementation of
each class of parallel GAs on existing parallel computers.

The research on parallel GAs is dominated by multiple-deme algorithms. This
class of parallel GAs is very complex, and its behaviour is affected by many parameters.
We also reviewed publications on master-slave and fine-grained parallel GAs and
realized that the combination of different parallelization strategies can result in faster
algorithms. It is particularly important to consider the hybridization of parallel
techniques in the light of recent results which predict the existence of an optimal number of demes. The next chapter begins our research investigations in this area.
CHAPTER III
PARALLEL MODELS AND TOPOLOGIES

Distributed Genetic Algorithms (dGAs) have received considerable attention because of their potential to reduce the execution time in complex applications [141]. One common method to parallelize genetic algorithms is to use multiple demes (populations) that occasionally exchange some individuals in a process called migration. The dGA is based on the distributed population structure that has the potential of providing better optimal values and is suited for parallel implementation. The distributed genetic algorithm executes a conventional genetic algorithm on each of the distributed populations; dGAs are difficult to configure because they are controlled by many parameters that affect their efficiency and accuracy like the population size, migration, recombination and mutation rates [36, 37].

This chapter explores the performance of three dGAs. The main difference between these three dGAs is the interconnection topology of the demes, but, as far as possible, all other aspects are the same. Thus, the main idea of this chapter is to investigate the difference in performance that can arise entirely due to interconnection topology. Each of the three dGAs we explore is a physical master-slave multiple-deme parallelization with a set of sixteen subpopulations.

3.1 Overview

The existing separate studies with dGAs pose some common problems to researchers. The first problem comes from the high variety of different parallel implementations. Some of these implementations have the same behaviour as a sequential GA, although others do not. It is only in recent years that an effort is being made to propose unified methodologies for designing and studying sequential and parallel GAs.

The large variety of different dGAs is partly generated by the fact that dGAs can be heterogeneous, either at the algorithm or hardware level, or both. Hardware heterogeneity means that some of the demes will execute on different computing platforms. Algorithm heterogeneity means, for example, that different types of GA might operate in different demes. In this thesis we focus on aspects of algorithm
heterogeneity. At this algorithm (or software) level, we can distinguish various sublevels according to the source of the heterogeneity:

a. Parameter level. The simplest example of software heterogeneity is to use the same GA in each island but varying the parameters of selection, recombination, mutation, and/or migration. These parameters could be initially fixed [120, 220], randomly chosen during the evolution [7, 8, 118], or follow an adaptive strategy [86, 194, 223].

b. Operator level. At this level the heterogeneity is introduced by using different genetic operators in each deme, with the GAs otherwise being the same [5, 244].

c. Genotype level. This is a more subtle kind of heterogeneity where each subpopulation stores locally encoded solutions represented with different encoding schemata [126, 181].

d. Algorithm level. This is the most general heterogeneity class at the software level. Each subpopulation can potentially run a different (evolutionary or non-evolutionary) algorithm [109, 117, 262].

Note that the algorithm level heterogeneity contains all previous levels, since, for example, a dGA with parameter-based heterogeneity is also algorithm-based heterogeneous. There also exist tools for the production of evolutionary algorithms not directly matching this classification, e.g., by allowing the automatic distribution of the computation, thus facilitating the creation of heterogeneous dGAs [9, 261].

Another orthogonal level of heterogeneity can be defined in relation to the relationship maintained among the subpopulations of the dGA [107]. Basically, if the amount of resources (individuals) of each subpopulation is not fixed during the evolution, i.e. the size of a subpopulation is made dependent on the current success of its strategy, and then it can be considered that the subpopulations are competing. Otherwise, it seems that the subpopulations collaborate to find the optimum. Hence, we differentiate between competition-based heterogeneity [109, 126, 181, 262], and collaboration-based heterogeneity [117, 261].
3.2 The Parallel Models

Here, we study the physical parallelization of a heterogeneous dGA which is based on Herrera and Lozano's gradual distributed real-coded GA (GD-RCGA) [117]. This model of search is a kind of distributed technique that runs sixteen populations concurrently using three different topologies with sparse migrations of individuals among them. The GD-RCGA model is suitable for the optimization of continuous functions, because it includes in the basic improvement loop of the algorithm the utilization of crossover operators for real-valued genes, engineered with fuzzy logic technology to deal with the traditional “fuzzy” GA concepts of exploration and exploitation. In our implementation of GD-RCGA we also use other kinds of traditional crossover operators like one-point crossover, two-point crossover, and m-points crossover.

Our implementation of GD-RCGA is configured as a master-slave model with sixteen subpopulations. It performs sparse migrations of individuals inside and outside two sets of eight sub-populations (islands). These models present different levels of heterogeneity. On one hand, it is parameter-level heterogeneous since the subpopulations use different mutation operators. But its subpopulations also utilize different crossover operators, so it can also be considered as operator-level heterogeneous. On the other hand, these models exhibits collaboration-based heterogeneity, since its subpopulations cooperate, not compete, in order to perform the search.

We explore three different versions of this approach, which use different interconnection topologies. In the following we explain these three versions. Following that, we will explain the core master and client processes involved in the distributed operation.

3.2.1 Model I

This model consists of one master running on a separate platform and sixteen clients (subpopulations) distributed on eight isolated platforms, two subpopulations each. Each one is connected directly to the master, and all these connections are established by using sockets technology. Next we give a high-level overview of the communication between the master and the slaves:

When any one of the subpopulations generates a new 'best' individual, better than previously achieved in that subpopulation, it will send this new individual to the master.
process. The master will store this chromosome if it is better than the chromosome it currently stores as the 'best so far', and, if so, it will send a copy of that chromosome to the slave subpopulation whose own best chromosome is the worst. In 'Model I', these activities are achieved in the context of the specific interconnection topology illustrated in Figure 3-1.

![Figure 3-1: Model I, Master-Slave Multiple-Deme dGA, all clients connected to master process](image)

In Figure 3-1 we can notice the high density of communications over the network. Despite the fact that we use one machine to hold two populations, the number of connection link between clients and master is equal to n, where n is the number of clients; this may affect the master's workload, especially since the master is responsible for transferring data between clients when required.

### 3.2.2 Model II

This model also consist of one master and sixteen clients distributed on eight machines, it similar to model I, but each two clients resident on one machine is connected locally to each other and only one of them is connected to the master process.
In this case: on the one hand, the number of connection between master and clients will be half of that in model I, and the amount of traffic between master and clients will similarly be halved; as far as the master is concerned, each client is equivalent to a single population (in the sense of model I). On the other hand, additional traffic is involved in communication between the two subpopulations on each client. Every time one of these subpopulations achieves a new best chromosome, this is copied to its sister subpopulation on the same machine. Figure 3-2 shows this topology.

![Figure 3-2: Model II, Master-Slave Multiple-Deme dGA, half clients connected to master process](image)

The 50% savings in master/client connectivity is clear from Figure 3-2 (when compared with Figure 3-1).

### 3.2.3 Model III

This model is quite different from models I and II, and based on a 3D Cube topology, which in turn is based on Alba et al's model [8]. In line with models I and II,
model III also has one master and sixteen clients. However the 16 clients are grouped into two cubes, as shown in Figure 3-3. Each client within the cube is connected directly to three other clients, and each client maintains a single subpopulation. At regular intervals, each client transmits its current best chromosome to one of its three direct neighbours (chosen uniformly at random). This process and topology reflects Alba et al.'s model [8]. The latter model used a single cube (eight clients); however our model III reflects a straightforward way to explore Alba et al's approach while being able to compare fairly with other models using sixteen subpopulations.

Thus this model contains two connected cubes. Migration between the two cubes also happens at fixed intervals, where the best chromosome from cube 1 is sent to a random client in cube 2, and vice versa. Figure 3-3 illustrates this model. The migration between cubes is handled by the master process, which also stores the best chromosomes so far for each client.

![Model III Diagram](image)

Figure 3-3: Model III, Master-Slave Multiple-Deme dGA, 2 × 3D Cubes based on Alba et al. [8] model

In our implementation of Model III, one machine runs the master process, and eight other machines run the client processes, each handling two subpopulations. The genetic algorithms in this model use the same "fuzzy connective-based crossover"
operators as used in Alba et al's version [8], more details about this type of crossover can be found in the Appendix A.

3.3 Master and Client Processes

The three main algorithms we investigate in this chapter (and throughout the thesis) are dGAs, implemented in a physically parallel and asynchronous environment. In each case, the algorithm runs a collection of separate GAs, with occasional migration of chromosomes between them. The GA aspects of each algorithm are always implemented in serial on a single machine, and the details of the GAs appear elsewhere in this thesis. In this section we will describe in detail, with pseudocode, the way that we have implemented the master and client processes which manage the individual GAs and the migration of chromosomes between them.

3.3.1 Master Side Process

Every GA has some parameters that should be determined at the beginning of its run, such as dimension (chromosome length), population size, target fitness value, stopping criteria, and other parameters related to the problem that the GA aims to solve, all these parameters are set by the master process. Then, when the GAs are running, the master has various other responsibilities related to book-keeping and migration. We can broadly summarize the responsibilities of the master process as follows:

1. Connect/Disconnect clients
2. Set GA problem (fitness function)
3. Set GA parameters
4. Start/Stop all processes
5. Receive/Store clients data
6. Send data to clients
7. Control clients processes

3.3.2 Client’s Side Process

All clients are same in each model; generally it’s enough to know how to set up one client to copy it to all others. It’s holding more than one thread working at same time to achieve its work and parallelization, one of these threads to make and keep connections over Ethernet network alive, master-clients or client-client, another thread
holding execution of GA including crossover and mutation. We can summarize client responsibilities by following points:

1. Connect master-clients or client-client
2. Initialize GA problem
3. Initialize GA parameters
4. Run GA process (population, evaluation, mutation and recombination)
5. Sending / Receiving data

The selecting schema for all models depends on individuals’ fitness. The individual which is carrying better fitness has more chance to be selected for recombination/mutation process. Almost 30% of current population of whole chromosomes are replaced with new chromosomes and remaining 70% are passed into next population through recombination/mutation operators.

In both model I and model II, Algorithm 3-1 uses a random choice of crossover operator selected from (one-point, two-points, m-points, uniform and guided crossover). In model III, Algorithm 3-1 uses Alba et al's fuzzy connective-based crossover operators (F-Crossover, S-Crossover, M-Crossover and L-Crossover). See Appendix A.5 for more details of all crossovers used in this thesis. The following pseudocode details the master and client side processes.

**Algorithm 3-1:** Pseudo code of a dGA on both master and client sides

**Master Process:**

**Thread1:**

Connect Master with Clients;
For each Clients
    Send all parameters;
    Send “Start” token;
End for
Run Thread2; // receive data
Run Thread3; // send data

**Thread2:**

Repeat
For each Client
  Receive individuals from clients
  If Optimal-individual = Null
    Set Optimal-individual;
    Set Optimal-Fitness-Value;
  Else
    if new individual better than Optimal-individual
      Set Optimal-individual;
      Set Optimal-Fitness-Value;
    End if;
  End if;
End for;
Until reach one of stopping criteria;

Thread3:
Repeat
  If Optimal-individual updated
    Send Optimal-individual to worst client;
  End if;
  Wait;
Until reach one of stopping criteria;

Client Process:

Thread 1:  // connect and receive data from master
Connect with master/client
If “Start” token received
  Run Thread2; // receive data from master/client
  Run Thread3; // run GA process
  Run Thread4; // send data to master/client
End if

Thread 2:
While connected with master/client
  receive individuals from master/client;
  If new individual is better than Optimal-individual
    update Optimal-individual;
    update Optimal-Fitness-Val;
    add individual to population
    remove worst individual form population
  End if
End while
Thread 3:

Initialize Population;
Set Optimal-individual;
Set Optimal-Fitness-Val;
Repeat
  Count++;
  For selected c1, c2
    crossover(c1,c2); // ==> c1’ and c2’
    with Pm mutate(c1’); // ==> c1"
    with Pm mutate(c2’); // ==> c2"
    find fit(c1");
    find fit(c2");
  If c1” is better than Optimal-individual
    send-New-Data = True;
    update Optimal-individual; // <= c1" 
    update Optimal-Fitness-Val; // <= fit(c1")
    add c1” to population;
    remove worst individual form population;
  End if
  If c2” is better than Optimal-individual
    send-New-Data = True;
    update Optimal-individual; // <= c2" 
    update Optimal-Fitness-Val; // <= fit(c2")
    add c2” to population;
    remove worst individual form population;
  End if
  new-population = best 70% of current population;
  new-population += 30% new chromosomes;
  population = new-population;
Until reach one of stopping criteria;

Thread 4:

While Thread3 is running
  If Send-New-Data
    send Optimal-individual to master/client;
    Send-New-Data = False;
    wait;
  End if;
End while;
3.4 Benchmark Test Functions

In this study we have analysed the results of minimization experiments on six well-known test functions. The used benchmark functions, which are more detailed in [236], used with both their classical and 'shifted' versions for each test function in order to help identify strengths and weaknesses in our models and algorithms. The utilized benchmark functions include Sphere, Rosenbrock, Schwefel, Rastrigin, Griewangk and Ackley. In the Appendix B, Table B-1 shows their formulation with normal versions, while Table B-2 illustrates the shifted ones. Figure 3-4 illustrates the landscape for 2D versions of each of these functions. The dimension of the search space is 30 for all test functions. Each fitness function has its particular features:

1. **Sphere** function (fSph) [52] or sometime so-called DeJong first function, is the simplest test function in our benchmark set. Sphere is continuous, strictly convex, and unimodal.

2. **Rosenbrock’s** valley function (fRos) [207] also called the banana function banana or DeJong second function. This fitness function is a continuous, non-separable (with nonlinear interactions among variables), and unimodal function, with the optimum located in a steep parabolic valley with a flat bottom. These features make the search direction have to continually change to reach the optimum. Experiments show that it is even more difficult than some multimodal benchmarks.

3. **Schwefel** function (fSch) [215] is a continuous and unimodal function. Its difficulty concerns the fact that searching along the coordinate axes only gives a poor rate of convergence because the gradient of fSch is not oriented along the axes. It presents similar difficulties to fRos, but its valley is much narrower. Therefore, the search algorithms are potentially prone to convergence in the wrong direction.

4. **Rastrigin** function (fRas) [177] is a scalable, continuous, and multimodal function. The Rastrigin fitness function is based on the DeJong function fSph with the addition of cosine modulation \( a \cdot \cos(w \cdot x_i) \) in order to produce frequent local minima. Thus, this test function is highly multimodal.
5. **Griewangk** function ($f_{Gri}$) [242] is a continuous and multimodal function. This function has many widely spread local minima regularly distributed which make it difficult to be optimized.

6. **Ackley** function ($f_{Ack}$) [66] is a continuous, non-separable, multi-modal and scalable function.
Figure 3-4: Benchmark test function from [157]; a) sphere, b) Rosenbrock, c) Rastrigin, d) Schwefel, e) and f) Griewangk, g) and h) Ackley
3.5 Experiments

In this section, we present the parameters used in each of the three models, we describe the experimental setup, and we then present and discuss the results obtained. The used population size for all models is 15 individuals per subpopulation. The migration is performed every 25 generations. The probability update of an individual by mutation (mutation rate) is \( P_m=0.05 \). The crossover probability is \( P_c=0.6 \). Two of stopping criteria have been utilized to terminate the whole process for all clients. The first criterion is a predefined maximum number of generations \( \text{max}_\text{gen}=10000 \), while the second one is to reach the predefined globally optimum fitness values for each benchmark fitness function which are illustrated in Table B-1 and Table B-2 for both shifted and normal versions.

The implementation platform is built from a cluster of eight personal computers plus one separate machine. Each of the first eight machines of the cluster holds two clients, and the ninth machine is reserved for the master process. These nine machines each ran Microsoft windows XP Professional SP3 on, an Intel Pentium IV 2.99 GHz processor with 2 GB of RAM. The machines are interconnected by a Fast-Ethernet (100 Mbps) network. We have compiled our programs with Microsoft Visual Studio 2012, Version 12.0.40219.1 SP1, Microsoft .NET Framework Version 4.5 SP1.

Table 3-1: dGA parameters and attributes

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithm</td>
<td>dGAT1, dGAT2, dGAT3</td>
</tr>
<tr>
<td>Population Size</td>
<td>15</td>
</tr>
<tr>
<td>Maximum Generation</td>
<td>10000</td>
</tr>
<tr>
<td>Number of Runs</td>
<td>20</td>
</tr>
<tr>
<td>Number of Sub-Populations</td>
<td>1, 8, 16</td>
</tr>
<tr>
<td>Dimension</td>
<td>30D</td>
</tr>
<tr>
<td>Crossover Rate</td>
<td>0.6</td>
</tr>
<tr>
<td>Mutation Rate</td>
<td>0.05</td>
</tr>
<tr>
<td>Migration Rate</td>
<td>Every 25 generation</td>
</tr>
</tbody>
</table>
Selection Operator

<table>
<thead>
<tr>
<th>Selection Operator</th>
<th>roulette wheel selection</th>
</tr>
</thead>
</table>

Mutation Operators

<table>
<thead>
<tr>
<th>Mutation Operators</th>
<th>uniform, non-uniform, multiple uniform and boundary mutation</th>
</tr>
</thead>
</table>

Crossover Operator

<table>
<thead>
<tr>
<th>Crossover Operator</th>
<th>one-point, two-point, m-points and fuzzy connective-based crossover</th>
</tr>
</thead>
</table>

Objective

<table>
<thead>
<tr>
<th>Objective</th>
<th>Benchmark fitness functions #12</th>
</tr>
</thead>
</table>

For each individual experiment (with specific parameter settings) we performed 20 independent runs. Our focus in this thesis is on how fast each model is able to reach the global optimum; we therefore record, for each trial, whether or not the global optimum is reached, and, if so, the execution time (in ms) taken to reach the global optimum. Table 3-1 summarises the dGA parameters and configuration choices

3.5.1 Experimental Results

3.5.1.1 The 16 Clients dGAs Tests

Let us now proceed with the analysis of the results. The execution times of three models working on 16 clients for twelve problems are presented below. Table 3-2 shows mean execution times of sequential genetic algorithm SGA compared with dGAT1, dGAT2, dGAT3. An initial clear observation is that the shifted versions of the functions (in the lower table) clearly need more time before optimal fitness is reached. The increase in execution time is clearly expected as shifted versions of the functions are more challenging. Figure 3-5 illustrates the mean execution time of the test benchmarks. The results clearly suggest that dGAT2 tends to perform better than dGAT1 and dGAT3. Meanwhile, examination of the comparison between dGAT1 and dGAT3 shows that their relative performance is quite problem dependent, with dGAT1 performing better than dGAT3 in some cases, and vice versa. In attempt to gain a better understanding of these initial results, the following speedup metric has been utilized, which essentially normalises the results with regard to the performance of SGA.
Table 3-2: Mean execution time (ms) to optimize benchmark functions for SGA and dGAT1, dGAT2, dGAT3 over a cluster of 16 clients

<table>
<thead>
<tr>
<th>Method</th>
<th>( f_{Sph} )</th>
<th>( f_{Ros} )</th>
<th>( f_{Ras} )</th>
<th>( f_{Sch} )</th>
<th>( f_{Gri} )</th>
<th>( f_{Ack} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>SGA</td>
<td>72515</td>
<td>375703</td>
<td>35197</td>
<td>1963761</td>
<td>77949</td>
<td>75139</td>
</tr>
<tr>
<td>dGAT1</td>
<td>828</td>
<td>40036</td>
<td>730</td>
<td>12680</td>
<td>859</td>
<td>809</td>
</tr>
<tr>
<td>dGAT2</td>
<td>778</td>
<td>37325</td>
<td>688</td>
<td>10664</td>
<td>702</td>
<td>677</td>
</tr>
<tr>
<td>dGAT3</td>
<td>808</td>
<td>41643</td>
<td>735</td>
<td>13132</td>
<td>756</td>
<td>803</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method</th>
<th>( f_{S-Sph} )</th>
<th>( f_{S-Ros} )</th>
<th>( f_{S-Ras} )</th>
<th>( f_{S-Sch} )</th>
<th>( f_{S-Gri} )</th>
<th>( f_{S-Ack} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>SGA</td>
<td>5142212</td>
<td>10119573</td>
<td>1435685</td>
<td>1350589</td>
<td>1350589</td>
<td>1296606</td>
</tr>
<tr>
<td>dGAT1</td>
<td>51219</td>
<td>750837</td>
<td>56009</td>
<td>48797</td>
<td>40909</td>
<td>45273</td>
</tr>
<tr>
<td>dGAT2</td>
<td>46363</td>
<td>551186</td>
<td>43877</td>
<td>43906</td>
<td>29812</td>
<td>30890</td>
</tr>
<tr>
<td>dGAT3</td>
<td>49833</td>
<td>718053</td>
<td>55459</td>
<td>50891</td>
<td>58575</td>
<td>51809</td>
</tr>
</tbody>
</table>
Figure 3-5 Benchmark mean execution time (ms) for dGAT1, dGAT2 and dGAT3
Table 3-3 reports this metric, showing the speedup over SGA obtained by each of the three models. This table presents speedup values for each model for all tested problems depending on execution times $T_{sequential}$ for SGA. When looking at Table 3-3, we can get a performance overview of each model. The rightmost column in this table gives the overall average speedup values. When looking at the average speed up for all models, it can be clearly seen that the model T2 showing a better performance than T1 and T3 for all test fitness function (the shifted and not shifted problems). The average speedup performances of T1 and T3 are very similar; however it seems that T1 is more successful for the harder problems, and T3 more successful for the easier problems. Figure 3-6 illustrates this comparison more clearly.

Table 3-3: The Speedup $S_N$ values of $dGAT1$, $dGAT2$ and $dGAT3$ over SGA of 30D benchmark problems

<table>
<thead>
<tr>
<th>Method</th>
<th>$f_{Sph}$</th>
<th>$f_{Ros}$</th>
<th>$f_{Ras}$</th>
<th>$f_{Sch}$</th>
<th>$f_{Gri}$</th>
<th>$f_{Ack}$</th>
<th>Mean $S_N$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$dGAT1$</td>
<td>87.58</td>
<td>9.38</td>
<td>48.22</td>
<td>154.87</td>
<td>90.74</td>
<td>92.88</td>
<td>80.61</td>
</tr>
<tr>
<td>$dGAT2$</td>
<td>93.21</td>
<td>10.07</td>
<td>51.16</td>
<td>184.15</td>
<td>111.04</td>
<td>110.99</td>
<td>93.44</td>
</tr>
<tr>
<td>$dGAT3$</td>
<td>89.75</td>
<td>9.02</td>
<td>47.89</td>
<td>149.54</td>
<td>103.11</td>
<td>93.57</td>
<td>82.15</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method</th>
<th>$f_{S-Sph}$</th>
<th>$f_{S-Ros}$</th>
<th>$f_{S-Ras}$</th>
<th>$f_{S-Sch}$</th>
<th>$f_{S-Gri}$</th>
<th>$f_{S-Ack}$</th>
<th>Mean $S_N$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$dGAT1$</td>
<td>100.4</td>
<td>13.48</td>
<td>25.63</td>
<td>27.68</td>
<td>33.01</td>
<td>28.64</td>
<td>38.14</td>
</tr>
<tr>
<td>$dGAT2$</td>
<td>110.91</td>
<td>18.36</td>
<td>32.72</td>
<td>30.76</td>
<td>45.3</td>
<td>41.97</td>
<td>46.67</td>
</tr>
<tr>
<td>$dGAT3$</td>
<td>103.19</td>
<td>14.09</td>
<td>25.89</td>
<td>26.54</td>
<td>23.06</td>
<td>25.03</td>
<td>36.3</td>
</tr>
</tbody>
</table>

Karp and Flatt [138] have devised an interesting metric for measuring the performance of any parallel algorithm that can help to identify much more subtle effects than using speedup alone. The authors call it the serial fraction of the algorithm. Another method to analyse parallel algorithm performance may be parallel efficiency first proposed by [108].

55
The parallel efficiency differs from our speedup metric (which is sometimes known as scaled efficiency). In order to enrich our understanding of the effects of parallelism for these three models Table 3-4 and Table 3-5 provide more detailed analyses. These tables present the two parallel metrics mentioned above, the parallel efficiency (PE) and the serial fraction (SF). In this context the parallel efficiency (PE) and the serial fraction (SF) can be defined, respectively, as:

\[ PE = \frac{S_N}{N} \]  

(3-2)

Where \( N = 16 \) is number of processors and \( S_N \) is the speedup.

\[ SF = \frac{1}{S_N} - \frac{1}{N} \]  

(3-3)
Table 3-4: The \textit{dGAT1, dGAT2 and dGAT3 Parallel Efficiency PE values}

<table>
<thead>
<tr>
<th>Method</th>
<th>\textit{fSph}</th>
<th>\textit{fRos}</th>
<th>\textit{fRas}</th>
<th>\textit{fSch}</th>
<th>\textit{fGri}</th>
<th>\textit{fAck}</th>
<th>Mean PE</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textit{dGAT1}</td>
<td>5.474</td>
<td>0.586</td>
<td>3.014</td>
<td><strong>9.679</strong></td>
<td>5.671</td>
<td>5.805</td>
<td><strong>5.038</strong></td>
</tr>
<tr>
<td>\textit{dGAT2}</td>
<td>5.826</td>
<td>0.629</td>
<td>3.198</td>
<td><strong>11.509</strong></td>
<td>6.94</td>
<td>6.937</td>
<td><strong>5.84</strong></td>
</tr>
<tr>
<td>\textit{dGAT3}</td>
<td>5.609</td>
<td><strong>0.564</strong></td>
<td>2.993</td>
<td><strong>9.346</strong></td>
<td>6.444</td>
<td>5.848</td>
<td><strong>5.134</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method</th>
<th>\textit{fS-Sph}</th>
<th>\textit{fS-Ros}</th>
<th>\textit{fS-Ras}</th>
<th>\textit{fS-Sch}</th>
<th>\textit{fS-Gri}</th>
<th>\textit{fS-Ack}</th>
<th>Mean PE</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textit{dGAT1}</td>
<td>6.275</td>
<td>0.843</td>
<td>1.602</td>
<td>1.73</td>
<td>2.063</td>
<td>1.79</td>
<td><strong>2.384</strong></td>
</tr>
<tr>
<td>\textit{dGAT2}</td>
<td>6.932</td>
<td>1.148</td>
<td>2.045</td>
<td>1.923</td>
<td>2.831</td>
<td>2.623</td>
<td><strong>2.917</strong></td>
</tr>
<tr>
<td>\textit{dGAT3}</td>
<td>6.449</td>
<td>0.881</td>
<td>1.618</td>
<td>1.659</td>
<td>1.441</td>
<td>1.564</td>
<td><strong>2.269</strong></td>
</tr>
</tbody>
</table>

Table 3-5: The \textit{dGAT1, dGAT2 and dGAT3 Serial Fraction SF values}

<table>
<thead>
<tr>
<th>Method</th>
<th>\textit{fSph}</th>
<th>\textit{fRos}</th>
<th>\textit{fRas}</th>
<th>\textit{fSch}</th>
<th>\textit{fGri}</th>
<th>\textit{fAck}</th>
<th>Mean SF</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textit{dGAT1}</td>
<td>-0.054</td>
<td>0.047</td>
<td>-0.045</td>
<td><strong>-0.06</strong></td>
<td>-0.055</td>
<td>-0.055</td>
<td><strong>-0.037</strong></td>
</tr>
<tr>
<td>\textit{dGAT2}</td>
<td>-0.055</td>
<td>0.039</td>
<td>-0.046</td>
<td><strong>-0.061</strong></td>
<td><strong>-0.057</strong></td>
<td><strong>-0.057</strong></td>
<td><strong>-0.04</strong></td>
</tr>
<tr>
<td>\textit{dGAT3}</td>
<td>-0.055</td>
<td><strong>0.052</strong></td>
<td>-0.044</td>
<td><strong>-0.06</strong></td>
<td>-0.056</td>
<td>-0.055</td>
<td><strong>-0.036</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method</th>
<th>\textit{fS-Sph}</th>
<th>\textit{fS-Ros}</th>
<th>\textit{fS-Ras}</th>
<th>\textit{fS-Sch}</th>
<th>\textit{fS-Gri}</th>
<th>\textit{fS-Ack}</th>
<th>Mean SF</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textit{dGAT1}</td>
<td><strong>-0.056</strong></td>
<td>0.012</td>
<td>-0.025</td>
<td>-0.028</td>
<td>-0.034</td>
<td>-0.029</td>
<td><strong>-0.027</strong></td>
</tr>
<tr>
<td>\textit{dGAT2}</td>
<td><strong>-0.057</strong></td>
<td>-0.009</td>
<td>-0.034</td>
<td>-0.032</td>
<td>-0.043</td>
<td>-0.041</td>
<td><strong>-0.036</strong></td>
</tr>
<tr>
<td>\textit{dGAT3}</td>
<td><strong>-0.056</strong></td>
<td>0.009</td>
<td>-0.025</td>
<td>-0.026</td>
<td>-0.02</td>
<td>-0.024</td>
<td><strong>-0.024</strong></td>
</tr>
</tbody>
</table>

The serial fraction should in theory stay constant as we vary \( N \) for any given parallelised algorithm. For example, even if the speedup value is small (e.g., efficiency around 87%), if \( SF \) remains constant for different values of \( N \) this suggests that the low efficiency is due to limited opportunity to exploit parallelism, rather than inefficiencies in the parallelisation itself. On the other hand, a smooth increase in serial fraction as \( N \)
increases provides a warning that the granularity of the parallelisation is too fine. A third scenario is possible in which a significant reduction in serial fraction $SF$ occurs as we rise $N$; this indicates super-linear speedup. If super-linear speedup occurs, then serial fraction $SF$ would take a negative value. Serial fraction is therefore a useful metric for analysing the performance of a dGA in terms of how it scales with the number of processors. In fact in this research we do not focus on scalability in this sense, but focus on the relative performance of different topologies and migration schemes with fixed numbers of processors. Nevertheless, it is known that a single data point for serial fraction provides useful information. For example, low values (below 0.1) suggest effective use of parallelism (even if parallel efficiency seems poor), while negative values are indicative of super-linear speedup. In later experiments in this and other chapters we also examine versions of the models with 8 client machines, and are thereby able to apply some limited reasoning to discuss how serial fraction changes in each case.

From Table 3-4, we can see that $dGAT2$ again seems to be the most effective of the three models in terms of parallel efficiency. The $dGAT2$ model has the best value for parallel efficiency for every test problem. As before, however, when we compare $dGAT1$ with $dGAT3$, we see that $dGAT1$ seems better on the harder (shifted) cases, while $dGAT3$ seems better on the easier cases. Some high positive values of serial fraction, (for example, $dGAT3$ not shifted Rosenbrock serial fraction is 0.052) point out that the parallel algorithm can be still further improved to better profit from the parallel platform, but we must not forget that Rosenbrock is a particularly difficult multimodal benchmark function, and a serial fraction value of 0.052, despite being less good than the values for all other models and problems, still signifies performance close to linear speedup.

Finally, it is noticeable that all of the best serial fraction (most negative) values are achieved by $dGAT2$, adding to the growing evidence that $dGAT2$ is the most effective of the three models under study. Figure 3-7 and Figure 3-8 illustrates parallel efficiency and serial fraction of all models.

With the intention of understanding the significance of our findings, a statistical comparison between the different studied parallel models using paired 1-tailed T-Tests has been performed over the execution times of the previous tests for both eight and
sixteen clients. These statistical tests have been done via a cut-off significance level of 95% \((p\text{-value} = 0.05)\) assuming unequal variance.

Figure 3-7: The \textit{dGAT1, dGAT2 and dGAT3 Serial Fraction}

Figure 3-8: The \textit{dGAT1, dGAT2 and dGAT3 Parallel Efficiency}
First, Table 3-6 summarises the results of paired T-tests between SGA and each of the dGA models for each benchmark function, for the sixteen-clients versions. Next, Table 3-7 presents the p-values for pairwise comparisons among the three dGA models for each benchmark function.

**Table 3-6: The 16 clients paired 1-tailed T-Test p-values for the comparison between SGA execution times and dGAT1, dGAT2, dGAT3**

<table>
<thead>
<tr>
<th>Method</th>
<th>fSph</th>
<th>fRos</th>
<th>fRas</th>
<th>fSch</th>
<th>fGri</th>
<th>fAck</th>
</tr>
</thead>
<tbody>
<tr>
<td>dGAT1</td>
<td>1.67E-19</td>
<td>8.03E-19</td>
<td>2.24E-19</td>
<td>9.68E-20</td>
<td>2.84E-19</td>
<td>7.94E-23</td>
</tr>
<tr>
<td>dGAT2</td>
<td>1.04E-19</td>
<td>2.33E-19</td>
<td>1.07E-19</td>
<td>4.61E-20</td>
<td>1.86E-19</td>
<td>2.99E-23</td>
</tr>
<tr>
<td>dGAT3</td>
<td>1.63E-19</td>
<td>4.57E-19</td>
<td>2.18E-19</td>
<td>9.24E-20</td>
<td>2.81E-19</td>
<td>7.98E-23</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method</th>
<th>fS-Sph</th>
<th>fS-Ros</th>
<th>fS-Ras</th>
<th>fS-Sch</th>
<th>fS-Gri</th>
<th>fS-Ack</th>
</tr>
</thead>
<tbody>
<tr>
<td>dGAT1</td>
<td>1.63E-19</td>
<td>3.84E-19</td>
<td>3.54E-19</td>
<td>3.18E-20</td>
<td>6.26E-21</td>
<td>2.78E-21</td>
</tr>
<tr>
<td>dGAT2</td>
<td>1.50E-19</td>
<td>2.03E-19</td>
<td>3.02E-19</td>
<td>2.25E-20</td>
<td>4.88E-21</td>
<td>1.99E-21</td>
</tr>
<tr>
<td>dGAT3</td>
<td>1.63E-19</td>
<td>3.72E-19</td>
<td>5.35E-19</td>
<td>3.75E-20</td>
<td>7.54E-21</td>
<td>2.32E-21</td>
</tr>
</tbody>
</table>

**Table 3-7: The 16 clients paired 1-tailed T-Test p-values for the comparison between the execution times of dGAT1, dGAT2 and dGAT3**

<table>
<thead>
<tr>
<th>Method</th>
<th>fSph</th>
<th>fRos</th>
<th>fRas</th>
<th>fSch</th>
<th>fGri</th>
<th>fAck</th>
</tr>
</thead>
<tbody>
<tr>
<td>dGAT1- dGAT2</td>
<td>0.075706</td>
<td>0.022935</td>
<td>0.300461</td>
<td>0.000827</td>
<td>0.000863</td>
<td>0.001301</td>
</tr>
<tr>
<td>dGAT3- dGAT2</td>
<td>0.140129</td>
<td>0.001113</td>
<td>0.020119</td>
<td>0.000244</td>
<td>0.015027</td>
<td>0.000886</td>
</tr>
<tr>
<td>dGAT1- dGAT3</td>
<td>0.371131</td>
<td>0.175238</td>
<td>0.072602</td>
<td>0.449363</td>
<td>0.015615</td>
<td>0.370726</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method</th>
<th>fS-Sph</th>
<th>fS-Ros</th>
<th>fS-Ras</th>
<th>fS-Sch</th>
<th>fS-Gri</th>
<th>fS-Ack</th>
</tr>
</thead>
<tbody>
<tr>
<td>dGAT1- dGAT2</td>
<td>1.8E-05</td>
<td>2.27E-06</td>
<td>8.68E-07</td>
<td>0.004555</td>
<td>3.96E-09</td>
<td>5.82E-10</td>
</tr>
<tr>
<td>dGAT3- dGAT2</td>
<td>0.003514</td>
<td>3.45E-08</td>
<td>5.56E-07</td>
<td>0.06067</td>
<td>3.74E-13</td>
<td>3.72E-12</td>
</tr>
<tr>
<td>dGAT1- dGAT3</td>
<td>0.015001</td>
<td>0.172523</td>
<td>0.207641</td>
<td>0.064551</td>
<td>7.43E-09</td>
<td>0.027929</td>
</tr>
</tbody>
</table>
From the results in Table 3-6 it can be clearly seen that all models with the associated dGAs show superior performance in comparisons with the SGA for all benchmark test functions. Since we are essentially testing 12 hypotheses, each time using the same set of SGA results, we should apply the Bonferroni correction; in this case, for each model we should multiply the p-values by 12, and this very clearly maintains the level of confidence well above 95%.

By looking at Table 3-7 (the sixteen-client versions), considering again the Bonferroni correction, we can conclude that the superiority of \( dGAT2 \) over \( dGAT1 \) is significant in eight of the twelve cases. When we compare \( dGAT3 \) with \( dGAT2 \), we find again that that eight of the twelve comparisons yields a statistically significant outperformance. For the other cases, we can see that the results suggest that \( dGAT2 \) is better than \( dGAT3 \) (from Table 3-2), however the variance across trials and the number of tests done so far means we are not able to conclude the superiority of \( dGAT2 \) with our chosen confidence level. When we compare \( dGAT1 \) with \( dGAT3 \), we find that no conclusive claim can be made in eleven of the twelve cases.
3.5.1.2 The 8 Clients dGAs Tests

To gain further understanding of the essential differences between our three models, all of the above experiments were repeated, but this time with eight clients rather than sixteen. These tests have been done using the same algorithms as described earlier in this chapter. Figure 3-9 illustrates the new structures of the studied models. The modifications made to the three topologies simply amount to disconnecting eight clients from models T1 and T2, while for model T3 we simply use a single eight-client cube, and no longer need the master process to moderate migration between two cubes but it still exist for monitoring purposes. Experiments were done using the same benchmark fitness functions with the same dGA parameters and configuration choices, again with 20 trials for each function for each model. Table 3-8 shows the mean execution times of these experiments.

Table 3-8: The dGAT1, dGAT2 and dGAT3 mean execution times (ms) for 8 clients

<table>
<thead>
<tr>
<th>Method</th>
<th>fSph</th>
<th>fRos</th>
<th>fRas</th>
<th>fSch</th>
<th>fGri</th>
<th>fAck</th>
</tr>
</thead>
<tbody>
<tr>
<td>dGAT1</td>
<td>2261</td>
<td>246484</td>
<td>1562</td>
<td>34059</td>
<td>2233</td>
<td>2031</td>
</tr>
<tr>
<td>dGAT2</td>
<td>1565</td>
<td>193983</td>
<td>1562</td>
<td>25316</td>
<td>1542</td>
<td>2000</td>
</tr>
<tr>
<td>dGAT3</td>
<td>1802</td>
<td>313005</td>
<td>2172</td>
<td>49369</td>
<td>1856</td>
<td>2359</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method</th>
<th>fS-Sph</th>
<th>fS-Ros</th>
<th>fS-Ras</th>
<th>fS-Sch</th>
<th>fS-Gri</th>
<th>fS-Ack</th>
</tr>
</thead>
<tbody>
<tr>
<td>dGAT1</td>
<td>138503</td>
<td>1762936</td>
<td>153054</td>
<td>528545</td>
<td>165531</td>
<td>141634</td>
</tr>
<tr>
<td>dGAT2</td>
<td>112712</td>
<td>1396264</td>
<td>114120</td>
<td>426094</td>
<td>143793</td>
<td>132812</td>
</tr>
<tr>
<td>dGAT3</td>
<td>137969</td>
<td>1795062</td>
<td>132423</td>
<td>713984</td>
<td>163831</td>
<td>146461</td>
</tr>
</tbody>
</table>

When comparing the results for the eight-client versions (in Table 3-8) with the previous results for the sixteen-client versions (Table 3-2), we can notice that dGAT2’s performance still dominates. Table 3-9 shows the speedup over SGA (SM) of the three eight-client models.
Figure 3-9: The three models with 8 clients, (a) model T1, (b) model T2, (c) model T3
the master process does not appear in model T3
Table 3-10 shows the serial fraction ($SF_M$) of these models, and Table 3-11 shows speedup ($SN_M$) using the following formula, to show how performance has changed as we move from sixteen to eight clients.

$$S_{NM} = \frac{S_N}{S_M} \quad , \quad N = 16 \quad , \quad M = 8 \quad (3-4)$$

When we examine the serial fraction values (Table 3-10) one very clear observation is that these values are worst for the non-shifted Rosenbrock function. As also reflected in Table 3-9 (speedup over SGA), this indicates sub-linear speedup. The reached $SF$ values mean that the studied parallel algorithm with their topologies could not reach super-linear speedup but they are very close to it or to near-linear speedup. However, in all other cases we see super-linear speedup. While for those $SF_M$ values which are more than 0.1 can say that the near-linear speedup has been achieved.

Table 3-9: The Speedup $S_M$ values of $dGAT1$, $dGAT2$ and $dGAT3$ with 8 clients

<table>
<thead>
<tr>
<th>Method</th>
<th>$f$-Sph</th>
<th>$f$-Ros</th>
<th>$f$-Ras</th>
<th>$f$-Sch</th>
<th>$f$-Gri</th>
<th>$f$-Ack</th>
<th>Mean $S_M$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$dGAT1$</td>
<td>32.07</td>
<td>1.52</td>
<td>22.53</td>
<td>57.66</td>
<td>34.91</td>
<td>36.99</td>
<td><strong>30.95</strong></td>
</tr>
<tr>
<td>$dGAT2$</td>
<td>46.35</td>
<td>1.94</td>
<td>22.53</td>
<td>77.57</td>
<td>50.56</td>
<td>37.57</td>
<td><strong>39.42</strong></td>
</tr>
<tr>
<td>$dGAT3$</td>
<td>40.24</td>
<td>1.2</td>
<td>16.2</td>
<td>39.78</td>
<td>42.01</td>
<td>31.85</td>
<td><strong>28.55</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method</th>
<th>$f$-Sph</th>
<th>$f$-Ros</th>
<th>$f$-Ras</th>
<th>$f$-Sch</th>
<th>$f$-Gri</th>
<th>$f$-Ack</th>
<th>Mean $S_M$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$dGAT1$</td>
<td>37.13</td>
<td>5.74</td>
<td>9.38</td>
<td>18.79</td>
<td>8.16</td>
<td>9.15</td>
<td><strong>14.73</strong></td>
</tr>
<tr>
<td>$dGAT2$</td>
<td>45.62</td>
<td>7.25</td>
<td>12.58</td>
<td>23.3</td>
<td>9.39</td>
<td>9.76</td>
<td><strong>17.98</strong></td>
</tr>
<tr>
<td>$dGAT3$</td>
<td>37.27</td>
<td>5.64</td>
<td>10.84</td>
<td>13.91</td>
<td>8.24</td>
<td>8.85</td>
<td><strong>14.13</strong></td>
</tr>
</tbody>
</table>

When we examine the rightmost column of Table 3-10 we can see super-linear speedup reflected in mainly negative; and otherwise very low $SF_M$ values. Similarly, the
rightmost column of Table 3-11 shows that doubling the number of clients led to better than doubled performance for each of our models when averaged over all test functions. Interestingly we also see that, for the easier functions, the improved performance is more marked for $dGAT1$ and $dGAT3$, however the improvement is more marked for $dGAT2$ when we consider the harder (shifted) functions. Figure 3-10 illustrate serial fraction of these model using 8 clients.

Finally we note that, for any given model, there is considerable variation in the performance metrics when we consider the different benchmark functions. If we considering a standard deterministic benchmark function, (e.g. factorizing a large prime number), then the standard parallel performance metrics would be useful in understanding the performance of different parallel models. This is still true for dGAs, however a complication arises that for different dGA applications (different optimisation landscapes), the pattern and degree of communication between clients and between master and clients will vary greatly, and this affects the parallel performance metrics. It is therefore of interest to find dGA topologies and migration schemes that performing well in this sense over a range of different landscapes.
Table 3-11: The speedup $S_{NM}$ of $dGAT1$, $dGAT2$ and $dGAT3$

<table>
<thead>
<tr>
<th>Method</th>
<th>$fS_{Sph}$</th>
<th>$fS_{Ros}$</th>
<th>$fS_{Ras}$</th>
<th>$fS_{Sch}$</th>
<th>$fS_{Gri}$</th>
<th>$fS_{Ack}$</th>
<th>Mean $S_{NM}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$dGAT1$</td>
<td>2.7</td>
<td>2.35</td>
<td>2.73</td>
<td>1.47</td>
<td>4.05</td>
<td>3.13</td>
<td>2.74</td>
</tr>
<tr>
<td>$dGAT2$</td>
<td>2.43</td>
<td>2.53</td>
<td>2.6</td>
<td>1.32</td>
<td>4.82</td>
<td>4.3</td>
<td>3</td>
</tr>
<tr>
<td>$dGAT3$</td>
<td>2.77</td>
<td>2.5</td>
<td>2.39</td>
<td>1.91</td>
<td>2.8</td>
<td>2.83</td>
<td>2.53</td>
</tr>
</tbody>
</table>

Figure 3-10: The $dGAT1$, $dGAT2$ and $dGAT3$ Serial Fraction $SF_{M}$ with 8 clients

Finally, Table 3-12 presents the statistical analysis, as the same comparisons as Table 3-7 of 16 clients, but for the eight-clients versions.
Now turning to the comparisons among the models in Table 3-7 and Table 3-12, looking at the eight-client version comparisons (Table 3-12), we see the same overall picture as for the 16 clients, except that the statistical significance of \textit{dGAT2}'s outperformance of \textit{dGAT1} and \textit{dGAT3} is true in (in both cases) only five of the twelve benchmark functions.

We can conclude in general that \textit{dGAT2} clearly has an advantage over both \textit{dGAT1} and \textit{dGAT3}, since in all pairwise statistical tests, \textit{dGAT2} is either found to be superior, or there is no statistically significant difference. Also the evidence for \textit{dGAT2}'s superiority grows as we move from eight-client to sixteen-client versions.

### 3.6 Conclusion

In this study we have constructed and used three master/client models of distributed genetic algorithms dGAs using physical parallelism. The constructed models depend on coarse-grained master-slave architectures. These models have been tested by using them to solve twelve different well-known fitness functions for optimization, where each one of these fitness functions has varied properties and complexity.
The first stage of testing was done using sixteen-client versions of the models, and also using a single-client sequential genetic algorithm (SGA) for benchmark comparison. In these tests, execution times to reach the target fitness were measured, and then we calculated a number of metrics of performance: speedup S, parallel efficiency PE and serial fraction SF, to help us gain an understanding of performance. We noticed that the parallel models with their associated dGAs generally achieved super-linear speedup. We concluded that \textit{dGAT2} was the superior dGA model of the three tested. We also found that \textit{dGAT1} seems superior to \textit{dGAT3} for normal (easier) benchmark functions while \textit{dGAT3} seems better than \textit{dGAT1} for the shifted (difficult) ones, although these comparisons were generally not statistically significant.

After initially testing sixteen-client versions of each model, we then tested eight-client versions. All dGA tests were repeated for the eight-client versions, and the same performance metrics were again calculated. The results revealed that each of our three dGA models seem able to achieve super-linear speedup for most of the benchmark functions. This suggests that each of our dGAs makes good use of the distributed architecture; however speedup-with-N is not a focus of this research, and in future chapters we will focus on using the sixteen-client versions.

Finally, we performed some basic statistical analyses to understand the significance of the various experimental results. The statistical analysis used 1-tailed paired T-tests using a standard confidence level of 95% (p-value threshold of 0.05). The statistical analysis confirmed that \textit{dGAT2} has superior performance to both \textit{dGAT1} and \textit{dGAT3} for most benchmark problems, with increasing levels of evidence for this conclusion as we went from eight clients to sixteen clients.

In the following chapter, we will focus on using adaptive migration schemes, and explore how such adaptive migration affects the performance of each of the three models.
CHAPTER IV

INVESTIGATING ADAPTIVE MIGRATION STRATEGIES

Without migration between demes, demes would be isolated and no advantage would be gained from the distributed environment. A simple migration strategy, allowing chromosomes from one deme to occasionally migrate to other demes, leads to beneficial performance.

The previous chapter used a simple migration strategy in each model, where new 'best-so-far' chromosomes in a deme were regularly copied to one or more neighbouring demes. There are many potential strategies that could be used for the migration process. "How often and who to migrate" is a complex question that can have many answers [145]. This chapter will focus on using an adaptive migration scheme to see if it can enhance the performance of the studied models (\textit{dGAT1}, \textit{dGAT2} and \textit{dGAT3}) used in Chapter III. The main idea behind the adaptive migration scheme investigated in this chapter is to make migration sensitive to the current 'progress' of a deme.

4.1 Overview

There is a huge space of potential alternatives for migration schemes. The common migration scheme is typically to send copies of good chromosomes from some populations directly to other populations (where they may for example, overwrite the worst chromosomes in the receiving subpopulations). When the broad dynamics of such a dEA are considered, it is intuitively clear that this form of migration corresponds to balancing useful exploration in the search (since there are independent subpopulations, mostly non-interacting) with occasional exploitation (promoted by migration), in a way that is not achieved by standard single-population EA designs. As mentioned, this typically leads to improved performance in terms of both solution quality and speed.

With regard to the migration rate, some distributed EA developers have a preference to do migration at each new generation [127, 190] or at randomly chosen periods [27, 163, 239]. The most important advantage of these methods is the simplicity which makes them able to be implemented easily via network of processors or demes.
But from another viewpoint, these migration schemes are too simple. They do not consider whether or not the target deme might benefit from the new chromosome, and are equally likely to send new chromosomes to demes that are progressing well and demes that are progressing poorly. Meanwhile, considering only frequency of migration, a too low migration frequency tends to get trapped into premature convergence, while a too high frequency affects the parallel evolution and system performance due to communication overload. Designing a good balance between these two extremes, and somehow considering when migration is a good idea and when it isn't, could potentially enhance the performance of dEAs.

Lobo et al. [154] have proposed an adaptive GA which works even when optimal number of individuals is not known. In this method parallel searches are conducted with different numbers of individuals, expecting one of the populations to have the appropriate number of individuals that yields good results. However, this method is not truly adaptive in the sense that the appropriate number of individuals is not learnt but is obtained by trial and error. The usefulness of this approach is questionable since it seems wasteful to perform many blind searches in parallel. Similarly, some of the applications of the parameter-less genetic algorithms [153] and multi-objective rule mining using genetic algorithms are discussed in [87].

An adaptive GA which runs three GAs in parallel is proposed in [121]. Here at each epoch (period) fitness-values of elite individuals are compared and the numbers of individuals are changed according to the results. For example, if GA with the largest number of individuals provides best results, then in the next epoch all GAs are given a large number of individuals. However, the optimum number of individuals required by a population depends on which region of the search space the individuals are in and is not the same for all subpopulations.

An adaptive GA where the mutation rate for an individual is encoded in the gene of an individual is proposed in [18]. The system was proposed with the hope that finally individuals with good mutation rate survive. However, only individuals with low mutation rate survive in the later phases of the search. An adaptive GA that determines mutation and crossover rate of an individual by its location in a two dimensional lattice plane is proposed in [147]. The algorithm maintains the diversity of these parameters by limiting the number of individuals in each lattice cell.

A meta-GA is a GA whose task is to optimise the parameters of another GA. However, in this process the number of evaluations needed is high and the process is a
costly one. One such meta-GA is proposed in [139]. A GA that adapts mutation and crossover rates in an Island model is proposed in [241]. Here adaptation is based on average fitness of the population. Parameters of a population here are updated to those of a neighbouring population with high average fitness.

The breeder genetic algorithm BGA depends on a set of control parameters and genetic operators. It is shown that strategy adaptation by competing subpopulations makes the BGA more robust and more efficient in [214]. Each subpopulation uses a different strategy which competes with other subpopulations. Some experiments on multi-parent reproduction in an adaptive genetic algorithm framework are performed in [67]. An adaptive mechanism based on competing subpopulations is incorporated into the algorithm in order to detect the best crossovers. A parallel genetic algorithm with dynamic mutation probability is presented in [151]. This algorithm is based on the farming model of parallel computation. The basic idea of the dynamic establishing mutation rate is presented. Similarly, an adaptive parallel genetic algorithm AMDPGA for optimization is discussed in [255].

A major problem in the use of genetic algorithms is premature convergence. An approach for dealing with this problem in the distributed genetic algorithm model is addressed in [117]. Its basic idea is to keep, in parallel, several subpopulations that are processed by genetic algorithms, with each one being independent of the others. But all these algorithms either consider mutation rate or crossover rate as a dynamic parameter, but not both at the same time. The application of a breeder genetic algorithm to the problem of parameter identification for an adaptive finite impulse filter is addressed in [173]. A population-based optimization method, called Free Search (FS) is also discussed in [189].

4.2 The dEA with Adaptive Migration Schemes dEA-AMS

As mentioned before, the studied dEAs are implemented in a physically parallel and asynchronous environment, using a collection of standard workstations via sockets technology. In all cases, the algorithm is controlled via a master thread and several client threads. The basic operation in all models is as follows.

1. Initially, connections are established between the master and its slaves via Fast Ethernet local network,
2. All involved threads are initialised (master and clients) to send/receive solutions between islands and to run and evaluate current population/solutions using dEAs.

3. The master thread (M) receives continual updates from each client thread (C) whenever a client has achieved a new best fitness (locally, i.e. in its own subpopulation).

4. When M determines that one of these chromosomes is a new global best, M sends this to the client that currently has the worst “best local fitness” according to M’s latest information. In the ‘fixed’ scheme, this happens with probability $P = 1$.

5. However, when the migration scheme is adaptive, M distributes the best chromosome to the worst client with an adaptive probability.

6. Meanwhile, client threads continue to operate the evolution of their own subpopulation on a single processor, and incorporate new chromosomes as and when they are received from M (or neighbour clients for models T2 and T3). Whenever a new chromosome is received by a client C, if it is fitter than the current best in that population, then it is included in the population and the current worst at C is discarded. At frequent intervals, each client sends its best chromosome to M.

The first model (T1) operates in precisely the way above, and uses a straightforward architecture in which the master is directly connected to each client. In models 2 and 3, the topology is altered, and the master thread connects to a restricted number of clients. In each case, the master connects to a group of clients, and the clients within a group communicate directly with each other.

Below is the pseudocode of Algorithm 4-1 which clarifies the operation of the master’s and client’s threads. For T1, the pseudocode is exact. For T2 and T3, there are differences that will be clarified later. In essence, however, each of T2 and T3 defines a set of groups of clients, and each client operates as both a master and a client within its group, while an overall master thread operates over the groups.
Algorithm 4-1: Pseudocode of AdGA on both master and client sides

**Master Process:**

Thread1:
Connect Master with Clients;
For each Clients
    Send all parameters;
    Send “Start” token;
    Set Client-Sending-Counter = 0;
    Set Client-Effectiveness-Counter = 0;
End for
Set Window = w;
Run Thread2; // receive data
Run Thread3; // send data
Run Thread4; // Migration adaptation

Thread2:
Repeat
    For each Client
        Client-Sending-Counter++;
        Receive individuals from clients
        If Optimal-individual = Null
            Client-Effectiveness-Counter++;
            Set Optimal-individual;
            Set Optimal-Fitness-Value;
        Else
            if new individual better than Optimal-individual
                Client-Effectiveness-Counter++;
                Set Optimal-individual;
                Set Optimal-Fitness-Value;
            End if;
        End for;
    Until reach one of stopping criteria;

Thread3:
Repeat
    If Optimal-individual updated
        Send Optimal-individual to worst client;
    End if;
Wait;
Until reach one of stopping criteria;

**Thread4:**

While Thread2 is running:
   For each Client
      With probability P
         Send “Increase Sending Rate” token
      With probability P’=1-P
         Send “Decrease Sending Rate” token
   End For
   If Client-Effectiveness-Counter >= Window
      Send “Reset Window” token for all clients;
      Set Client-Effectiveness-Counter = 0;
   End if;
   Wait;
Loop

**Client Process:**

**Thread 1:**  // connect and receive data from master
   Connect with master/client
   If “Start” token received
      Run Thread2; // receive data from master/client
      Run Thread3; // run GA process
      Run Thread4; // send data to master/client
      Run Thread5; // Migration adaptation
   End if

**Thread 2:**

While connected with master/client
   receive individuals from master/client;
   If new individual is better than Optimal-individual
      update Optimal-individual;
      update Optimal-Fitness-Val;
      add individual to population
      remove worst individual form population
   End if
End while
Thread 3:

Initialize Population;
Set Optimal-individual;
Set Optimal-Fitness-Val;
Set Sending-Data-Rate = R;
Set Sending-New-Data-Count = 0;
Repeat
  Count++;
  For selected c1, c2
    crossover(c1,c2); // ==> c1' and c2'
    with Pm mutate(c1'); // ==> c1"
    with Pm mutate(c2'); // ==> c2"
    find fit(c1"
    find fit(c2"
    If c1" is better than Optimal-individual
      Sending-New-Data-Count++;
      If Sending-New-Data-Count / Sending-Data-Rate ≥ 1
        Set Send-New-Data = True;
        update Optimal-individual; // <= c1"
        update Optimal-Fitness-Val; // <= fit(c1")
        add c1" to population;
        remove worst individual form population;
      End if
    If c2" is better than Optimal-individual
      Sending-New-Data-Count++;
      If Sending-New-Data-Count / Sending-Data-Rate ≥ 1
        Set Send-New-Data = True;
        update Optimal-individual; // <= c2"
        update Optimal-Fitness-Val; // <= fit(c2")
        add c2" to population;
        remove worst individual form population;
      End if
    new-population = best 70% of current population;
    new-population += 30% new chromosomes;
    population = new-population;
  Until reach one of stopping criteria;

Thread 4:

While Thread3 is running
  If Send-New-Data
    send Optimal-individual to master/client;
    Send-New-Data = False;
wait;
End if;
End while;

Thread 5:
While Thread3 is running
  If “Increase Sending Rate” Token Received
    Sending-Data-Rate--
  Else If “Decrease Sending Rate” Token Received
    Sending-Data-Rate++
  If “Reset Window” Token Received
    Set Sending-Data-Rate=R;
Loop;

Here, in comparison between Algorithm 3-1 and Algorithm 4-1, some modifications and updates can be noticed on both the master and client side. These differences mainly appear in Thread4 on the master side and Thread5 on the client side, which are responsible for adaptation control, where other differences appear in bold on both sides. The adaptation control depends on clients’ local optima. Each time a client sends a new solution (local optimum) to the master; the master registers a new local optimum and updates its records for that client. If the master's records show that this new local optimum is a new global best, then it will be sent to the 'worst' client i.e. the client who best-so-far is currently the worst.

At fixed intervals, the master considers the recent performance of each client; good clients will be asked to increase the migration’s rate (initially set to be every 25 generations), while less-successful clients will be asked to decrease their migration’s rate. After a fixed number of generations (in our study 100 and 300) all clients will be enforced to reset the migration’s rate to the initial value.
4.3 Experiments

4.3.1 Benchmark Test Functions

The experiments in this section were performed on the same benchmark functions described in section 3.4. We test shifted and normal version of these functions, Table B-1 and Table B-2 respectively summarises the normal and shifted versions. However in this chapter we also consider 50-dimensional and 100-dimensional versions of these problems.

4.3.2 The Distributed GA Parameters

Before presenting the results of our exploration of adaptive migration schemes, we set out here the parameters and other aspects of the experiments in this chapter. The used population size for all models is 15 individuals per subpopulation. The probability of update of an individual by mutation (mutation rate) is $P_m=0.05$. The crossover probability is $P_r=0.6$. Two stopping criteria have been utilized to terminate the whole process for all clients. The first criterion is GA work imposed a predefined maximum number of generation $max_{gen}=10000$, while the second one is to reach the predefined fitness optimum values for each benchmark fitness function which are shown in Table B-1 and Table B-2.

The computing system is built from a cluster of eight personal computers plus one separate machine. Thus, each machine of the cluster holds two clients, where the additional separate one is reserved for the master process. These nine machines run Microsoft Windows XP Professional SP3, an Intel Pentium IV 2.99 GHz processor, and 2 GB of memory. The machines are interconnected by a Fast-Ethernet (100 Mbps) network. We have compiled our programs by Microsoft Visual Studio 2012, Version 12.0.30729.1 SP1, Microsoft .NET Framework Version 4.5 SP1.

Each individual experiment (applying a specific algorithm to a specific optimization problem) was repeated 20 times with independent random seeds. Results focus on the trials that reach target fitness within the maximum number generations. Finally, Table 4-1 summarises the parameters and other aspects of the experimental setup.
Table 4-1: dGA parameters and attributes

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithm</td>
<td>AdGAT1, AdGAT2, AdGAT3</td>
</tr>
<tr>
<td>Population Size</td>
<td>15</td>
</tr>
<tr>
<td>Maximum Generation</td>
<td>10000</td>
</tr>
<tr>
<td>Number of Runs</td>
<td>20</td>
</tr>
<tr>
<td>Number of Sub-Populations</td>
<td>16</td>
</tr>
<tr>
<td>Problem Dimensions</td>
<td>30D, 50D, 100D</td>
</tr>
<tr>
<td>Crossover Rate</td>
<td>0.6</td>
</tr>
<tr>
<td>Mutation Rate</td>
<td>0.05</td>
</tr>
<tr>
<td>Migration Rate</td>
<td>Adapted</td>
</tr>
<tr>
<td>Adaptive Migration Scheme</td>
<td>100, 300 Generations</td>
</tr>
<tr>
<td>Selection Operator</td>
<td>roulette wheel selection</td>
</tr>
<tr>
<td>Mutation Operators</td>
<td>uniform, non-uniform, multiple uniform and boundary mutation</td>
</tr>
<tr>
<td>Crossover Operator</td>
<td>one-point, two-point, m-points and fuzzy connective-based crossover</td>
</tr>
<tr>
<td>Objective</td>
<td>12 Benchmark fitness functions</td>
</tr>
</tbody>
</table>

4.3.3 The Experimental Results

To provide a more challenging test for our new adaptive models (AdGAT1, AdGAT2 and AdGAT3), our experimental study extends beyond that in Chapter III by considering higher-dimensional versions of the benchmark problems. Meanwhile, we explore only two values of the 'adaptation window reset' parameter, namely 100 and 300. These were quite arbitrary choices, chosen to enable a basic grasp of the effect of this parameter, however much further work is needed to properly investigate it. The adaptation reset window was chosen to be 100 and 300 generation (i.e. after 100/300 generations master reset the migration rate for all clients). The adaptation reset window...
help the master processes to reset all clients to the original migration rate which allow extra control and avoid and/or reduce the convergence to local optima.

4.3.3.1 The 30-Dimensional Test Problems

The test results of the three models working on 16 clients each solving the twelve benchmark fitness functions over a cluster of machines is presented below. Table 4-2, shows the success rates of $AdGAT1$, $AdGAT2$ and $AdGAT3$, while Table 4-3 recaps the success rates of the non-adaptive methods from Chapter III on the same problems.

From Table 4-2 it can be noticed that most problem were solved in almost 100% of the trials, especially the simpler (non-shifted) versions. Generally, the success rate performance of the adaptive approaches is slightly better than that of the non-adaptive approaches on these problems.

For example, on the Schwefel function, the non-adaptive scheme, $dGAT2$ achieved 18 successful runs out of 20 on the shifted version and 17 out of 20 on the non-shifted version, while $AdGAT2$ with an adaptive-reset window of 100 achieved 18 successful runs out of 20 for the non-shifted version and achieved 19 successful runs out of 20 for the shifted version. These results give us our first snapshot of the difference in performance between the adaptive approaches and the non-adaptive approaches of the previous chapter.

<table>
<thead>
<tr>
<th>Method</th>
<th>fSph</th>
<th>fRos</th>
<th>fRas</th>
<th>fSch</th>
<th>fGri</th>
<th>fAck</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>AdGAT1</td>
<td>20(20)</td>
<td>20(20)</td>
<td>20(20)</td>
<td>18(18)</td>
<td>20(20)</td>
<td>20(20)</td>
<td>118(118)</td>
</tr>
<tr>
<td>AdGAT2</td>
<td>20(20)</td>
<td>20(20)</td>
<td>20(20)</td>
<td>20(18)</td>
<td>20(20)</td>
<td>20(20)</td>
<td>120(118)</td>
</tr>
<tr>
<td>AdGAT3</td>
<td>20(20)</td>
<td>20(20)</td>
<td>20(20)</td>
<td>20(18)</td>
<td>20(20)</td>
<td>20(20)</td>
<td>120(118)</td>
</tr>
</tbody>
</table>

Table 4-2: The $AdGAT1$, $AdGAT2$ and $AdGAT3$ success rates out of 20 to optimize the 30D benchmark functions with adaption regime 100(300).

<table>
<thead>
<tr>
<th>Method</th>
<th>fS-Sph</th>
<th>fS-Ros</th>
<th>fS-Ras</th>
<th>fS-Sch</th>
<th>fS-Gri</th>
<th>fS-Ack</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>AdGAT1</td>
<td>19(20)</td>
<td>19(18)</td>
<td>20(20)</td>
<td>18(18)</td>
<td>20(20)</td>
<td>20(20)</td>
<td>116(116)</td>
</tr>
<tr>
<td>AdGAT2</td>
<td>20(20)</td>
<td>19(19)</td>
<td>20(20)</td>
<td>19(19)</td>
<td>20(20)</td>
<td>20(20)</td>
<td>118(118)</td>
</tr>
<tr>
<td>AdGAT3</td>
<td>20(20)</td>
<td>18(17)</td>
<td>20(20)</td>
<td>18(18)</td>
<td>20(20)</td>
<td>20(20)</td>
<td>116(115)</td>
</tr>
</tbody>
</table>
Table 4-3: The dGAT1, dGAT2 and dGAT3 success rates out of 20 to optimize the 30D benchmark functions.

<table>
<thead>
<tr>
<th>Method</th>
<th>fSph</th>
<th>fRos</th>
<th>fRas</th>
<th>fSch</th>
<th>fGri</th>
<th>fAck</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>dGAT1</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>16</td>
<td>20</td>
<td>20</td>
<td>116</td>
</tr>
<tr>
<td>dGAT2</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>17</td>
<td>18</td>
<td>17</td>
<td>112</td>
</tr>
<tr>
<td>dGAT3</td>
<td>17</td>
<td>17</td>
<td>20</td>
<td>15</td>
<td>20</td>
<td>20</td>
<td>109</td>
</tr>
</tbody>
</table>

Table 4-4: The Speedup values of the adapted models AdGAT1, AdGAT2 and AdGAT3 of adaptive regime 100(300) over the dGAT1, dGAT2 and dGAT3 of 30D benchmark problems.

<table>
<thead>
<tr>
<th>Method</th>
<th>fSph</th>
<th>fRos</th>
<th>fRas</th>
<th>fSch</th>
<th>fGri</th>
<th>fAck</th>
<th>Mean $S_N$</th>
</tr>
</thead>
<tbody>
<tr>
<td>AdGAT1</td>
<td>0.93(0.7)</td>
<td>0.55(0.88)</td>
<td>0.44(0.3)</td>
<td>0.63(0.51)</td>
<td>0.86(1.95)</td>
<td>1.11(1.11)</td>
<td><strong>0.77(0.61)</strong></td>
</tr>
<tr>
<td>AdGAT2</td>
<td>1.29(1.08)</td>
<td>1.07(1.09)</td>
<td>0.54(0.35)</td>
<td>1.97(1.27)</td>
<td>1.41(1.28)</td>
<td>1.92(1.18)</td>
<td><strong>1.07(1.04)</strong></td>
</tr>
<tr>
<td>AdGAT3</td>
<td>1.25(1.13)</td>
<td>0.8(0.43)</td>
<td>0.51(0.33)</td>
<td>1.78(0.92)</td>
<td>0.86(0.99)</td>
<td>1.02(1.09)</td>
<td><strong>1.01(0.82)</strong></td>
</tr>
</tbody>
</table>

In attempt to gain some more insight, in the following we show the speedup and serial fraction of the mean execution times for successful runs. The speedup and serial fraction have previously detailed in Chapter III. Table 4-4 shows the speedup between the adapted models (AdGAT1, AdGAT2 and AdGAT3) and the previous models (dGAT1, dGAT2 and dGAT3). While the Table 4-5 shows the serial fraction values.
From Table 4-4 and Table 4-5 we can see that there seems to be a slightly difference between the adaptive regime 100 and 300 in terms of speedup and serial fraction of the 30D problems. For example the speedup of $AdGAT1$ of $fSch$ fitness function is 0.63 of adaptive regime 100 and 0.51 of adaptive regime 300. Also, the serial fraction of $fS-Sch$ function equal to 1.41 of adaptive regime 100 and 1.33 of adaptive regime 300. It can be noticed that the adaptive regime 100 showing a better performance than 300 for most benchmark fitness functions. However, the “Mean $S_N$” from Table 4-4 reveals at least a small enhancement of performance of $AdGAT2$ in comparison with $dGAT2$.

Table 4-5: The Serial Fraction values of the adapted models $AdGAT1$, $AdGAT2$ and $AdGAT3$ of adaptive regime 100(300) over the $dGAT1$, $dGAT2$ and $dGAT3$ of the 30D benchmark problems.

<table>
<thead>
<tr>
<th>Method</th>
<th>$fSph$</th>
<th>$fRos$</th>
<th>$fRas$</th>
<th>$fSch$</th>
<th>$fGri$</th>
<th>$fAck$</th>
<th>Mean SF</th>
</tr>
</thead>
<tbody>
<tr>
<td>$AdGAT1$</td>
<td>1.08(1.46)</td>
<td>2.32(2.72)</td>
<td>2.34(3.52)</td>
<td>1.61(2.01)</td>
<td>1.17(0.48)</td>
<td>0.89(0.89)</td>
<td>1.32(1.68)</td>
</tr>
<tr>
<td>$AdGAT2$</td>
<td>0.76(0.66)</td>
<td>0.72(0.83)</td>
<td>1.92(2.98)</td>
<td>0.67(0.77)</td>
<td>0.7(0.4)</td>
<td>1.09(0.84)</td>
<td>0.93(0.96)</td>
</tr>
<tr>
<td>$AdGAT3$</td>
<td>0.79(0.88)</td>
<td>1.27(2.4)</td>
<td>2.05(3.19)</td>
<td>0.67(0.79)</td>
<td>1.17(1.01)</td>
<td>0.97(0.91)</td>
<td>0.99(1.23)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method</th>
<th>$fS-Sph$</th>
<th>$fS-Ros$</th>
<th>$fS-Ras$</th>
<th>$fS-Sch$</th>
<th>$fS-Gri$</th>
<th>$fS-Ack$</th>
<th>Mean SF</th>
</tr>
</thead>
<tbody>
<tr>
<td>$AdGAT1$</td>
<td>0.93(0.98)</td>
<td>0.74(0.78)</td>
<td>0.81(0.94)</td>
<td>1.41(1.33)</td>
<td>1.67(1.67)</td>
<td>0.42(0.37)</td>
<td>1.06(1.11)</td>
</tr>
<tr>
<td>$AdGAT2$</td>
<td>0.71(0.75)</td>
<td>0.58(0.65)</td>
<td>0.91(0.98)</td>
<td>0.51(0.61)</td>
<td>0.92(0.8)</td>
<td>0.36(0.36)</td>
<td>0.58(0.65)</td>
</tr>
<tr>
<td>$AdGAT3$</td>
<td>1.1(1.05)</td>
<td>1.04(1.01)</td>
<td>0.97(1.08)</td>
<td>1.21(0.89)</td>
<td>0.95(0.98)</td>
<td>0.91(0.97)</td>
<td>1.02(0.99)</td>
</tr>
</tbody>
</table>

Also, considering the mean speedup and mean serial fraction values, it can be noticed that the adaptive models show a more marked enhancement in their performance with the shifted versions of the benchmark functions. The Figure 4-1 and Figure 4-2 show the speedup values of 30-dimensional tests of adaptive regime 100, 300 respectively. The Figure 4-3 and Figure 4-4 show the serial fraction values of 30-dimensional tests of adaptive regime 100, 300 respectively.
Overall, considering the results on the 30-dimensional functions, the adaptive models seem to show only a slight advantage in performance. We next investigate the
adaptive models, again comparing them with the non-adaptive versions from Chapter III, in the more challenging context of higher-dimensional versions of the problems.

Figure 4-3: The $AdGAT_1$, $AdGAT_2$ and $AdGAT_3$ Serial Fraction values of the (100) adaptive regime of the 30D benchmark test problems.

Figure 4-4: The $AdGAT_1$, $AdGAT_2$ and $AdGAT_3$ Serial Fraction values of the (300) adaptive regime of the 30D benchmark test problems.
4.3.3.2 The 50-Dimensional Test Problems

Let us now move to some tests of the adaptive models with higher dimensional problems of 50D. Table 4-6 shows the success rates of the adaptive models for the 50D problems, while Table 4-7 shows the success rates for the non-adaptive models on the same problems.

We can see from Table 4-7 that the success-rates of the non-adaptive models drop significantly on the 50D problems, especially on the shifted versions. For example, $dGAT_1$ successfully optimized $fRas$ 17 times out of 20, and $fRos$ 14 times out of 20, while achieving 14 out of 20 and 0 out of 20 respectively on the shifted versions. From Table 4-6 we can notice that the adaptive models perform appreciably better than the non-adaptive approaches on the 50D problems.

Considering the three adaptive models, it seems that $AdGAT_2$ shows some performance domination over the others. Meanwhile, $AdGAT_3$ shows similar performance to $AdGAT_1$ for most benchmark problems, but seems slightly better than $AdGAT_1$ when using the (300) adaptive regime. For example the total successful runs of $AdGAT_2$ for the non-shifted benchmarks are 115(113) out of 120 for the 100(300) adaptive regime; the comparative results for $AdGAT_1$ and $AdGAT_3$ are respectively 114(110) and 114(111). On the shifted versions, $AdGAT_2$ achieves 96(94) successful runs out of 120 for the 100(300) adaptive regimes, while $AdGAT_1$ shows 95(93) and $AdGAT_3$ also shows 95(93). On the 50D functions, the adaptive approaches start to show a clear advantage compared to the non-adaptive approaches, however, when considering success rate alone, $AdGAT_1$, $AdGAT_2$ and $AdGAT_3$ do not show significant differences between each other. We now look at speedup and serial fraction of the mean execution times for the 50D problems, to see if this sheds further light on performance.

Table 4-8 shows the speedup of the adaptive models over the non-adaptive models for the 50D tests with the two adaptive regimes 100(300), while Table 4-9 shows the corresponding serial fraction values. From Table 4-8 it appears that all adaptive models show better performance compared to their corresponding non-adaptive versions. Each shows a speedup >1 for most benchmark test functions. Considering the mean speedup values, one can notice that the adaptive models all show speedup >1. Also, it seems that the adaptive regime (100) achieves more effective migration control than the adaptive regime (300).
However, the mean speedup figures from Table 4-8 suggest that there is no a significant difference between the adaptive regimes (100) and (300) on these problems, with regime (100) showing only a slight improvement over regime (300).

**Table 4-6**: The $AdGAT_1$, $AdGAT_2$ and $AdGAT_3$ success rates out of 20 to optimize benchmark functions with 50D adaption regime 100(300).

<table>
<thead>
<tr>
<th>Method</th>
<th>$f_{Sph}$</th>
<th>$f_{Ros}$</th>
<th>$f_{Ras}$</th>
<th>$f_{Sch}$</th>
<th>$f_{Gri}$</th>
<th>$f_{Ack}$</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>$AdGAT_1$</td>
<td>20(20)</td>
<td>16(16)</td>
<td>19(18)</td>
<td>19(18)</td>
<td>20(20)</td>
<td>20(18)</td>
<td>114(110)</td>
</tr>
<tr>
<td>$AdGAT_2$</td>
<td>20(20)</td>
<td>18(16)</td>
<td>18(18)</td>
<td>19(19)</td>
<td>20(20)</td>
<td>20(20)</td>
<td>115(113)</td>
</tr>
<tr>
<td>$AdGAT_3$</td>
<td>20(20)</td>
<td>18(16)</td>
<td>17(18)</td>
<td>19(17)</td>
<td>20(20)</td>
<td>20(20)</td>
<td>114(111)</td>
</tr>
</tbody>
</table>

**Table 4-7**: The $dGAT_1$, $dGAT_2$ and $dGAT_3$ success rates out of 20 to optimize the 50D benchmark functions.

<table>
<thead>
<tr>
<th>Method</th>
<th>$f_{Sph}$</th>
<th>$f_{Ros}$</th>
<th>$f_{Ras}$</th>
<th>$f_{Sch}$</th>
<th>$f_{Gri}$</th>
<th>$f_{Ack}$</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>$dGAT_1$</td>
<td>19</td>
<td>14</td>
<td>17</td>
<td>16</td>
<td>18</td>
<td>17</td>
<td>101</td>
</tr>
<tr>
<td>$dGAT_2$</td>
<td>18</td>
<td>16</td>
<td>17</td>
<td>17</td>
<td>18</td>
<td>16</td>
<td>102</td>
</tr>
<tr>
<td>$dGAT_3$</td>
<td>19</td>
<td>17</td>
<td>17</td>
<td>16</td>
<td>18</td>
<td>19</td>
<td>106</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method</th>
<th>$f_{S-Sph}$</th>
<th>$f_{S-Ros}$</th>
<th>$f_{S-Ras}$</th>
<th>$f_{S-Sch}$</th>
<th>$f_{S-Gri}$</th>
<th>$f_{S-Ack}$</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>$dGAT_1$</td>
<td>20(20)</td>
<td>0(0)</td>
<td>17(17)</td>
<td>18(18)</td>
<td>20(20)</td>
<td>20(18)</td>
<td>95(93)</td>
</tr>
<tr>
<td>$dGAT_2$</td>
<td>20(20)</td>
<td>0(0)</td>
<td>18(18)</td>
<td>18(18)</td>
<td>20(20)</td>
<td>20(18)</td>
<td>96(94)</td>
</tr>
<tr>
<td>$dGAT_3$</td>
<td>20(20)</td>
<td>0(0)</td>
<td>18(18)</td>
<td>17(17)</td>
<td>20(20)</td>
<td>20(18)</td>
<td>95(93)</td>
</tr>
</tbody>
</table>
For example the mean speedup value for not shifted versions of $AdGAT_1$ is 1.55(1.5) for the adaptive regime 100(300) and it comes to be 1.38(1.37) for the shifted versions. On the other hand, $AdGAT_2$ seems clearly superior to $AdGAT_1$ and $AdGAT_3$ for both the shifted and not shifted versions.

Table 4-8: The Speedup values of the adapted models $AdGAT_1$, $AdGAT_2$ and $AdGAT_3$ of adaptive regime 100(300) over the $dGAT_1$, $dGAT_2$ and $dGAT_3$ with 50D benchmark functions.

<table>
<thead>
<tr>
<th>Method</th>
<th>$fSph$</th>
<th>$fRos$</th>
<th>$fRas$</th>
<th>$fSch$</th>
<th>$fGri$</th>
<th>$fAck$</th>
<th>Mean $S_N$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$AdGAT_1$</td>
<td>1.2(1.16)</td>
<td>1.77(1.76)</td>
<td>2.35(2)</td>
<td>1.82(2.02)</td>
<td>1.02(0.97)</td>
<td>1.17(1.12)</td>
<td>1.55(1.50)</td>
</tr>
<tr>
<td>$AdGAT_2$</td>
<td>2.57(2.52)</td>
<td>6.94(5.24)</td>
<td>1.08(1.07)</td>
<td>2.27(2.31)</td>
<td>2.24(2.15)</td>
<td>1.49(1.42)</td>
<td>2.77(2.45)</td>
</tr>
<tr>
<td>$AdGAT_3$</td>
<td>0.88(0.82)</td>
<td>3.73(3.27)</td>
<td>1.05(1.14)</td>
<td>2.55(2.56)</td>
<td>0.72(0.65)</td>
<td>0.9(0.82)</td>
<td>1.64(1.54)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method</th>
<th>$fS-Sph$</th>
<th>$fS-Ros$</th>
<th>$fS-Ras$</th>
<th>$fS-Sch$</th>
<th>$fS-Gri$</th>
<th>$fS-Ack$</th>
<th>Mean $S_N$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$AdGAT_1$</td>
<td>1.01(0.98)</td>
<td>-</td>
<td>0.97(0.98)</td>
<td>1.63(1.62)</td>
<td>2.27(2.17)</td>
<td>1.02(1.11)</td>
<td>1.38(1.37)</td>
</tr>
<tr>
<td>$AdGAT_2$</td>
<td>0.98(0.89)</td>
<td>-</td>
<td>3.3(3.41)</td>
<td>3.39(1.52)</td>
<td>2.44(2.35)</td>
<td>2.67(2.85)</td>
<td>2.55(2.20)</td>
</tr>
<tr>
<td>$AdGAT_3$</td>
<td>1.08(1.04)</td>
<td>-</td>
<td>1.26(1.23)</td>
<td>1.35(1.36)</td>
<td>1.08(1.04)</td>
<td>1.02(1.02)</td>
<td>1.16(1.14)</td>
</tr>
</tbody>
</table>

The performance of the adaptive parallel models is slightly dropped down with the shifted versions (as expected) as the shifted copies are much harder than the normal versions which require more time to be optimized. Figure 4-5 and Figure 4-6 show the speedup values on the 50D functions of adaptive regimes 100 and 300 respectively. The serial fraction results from Table 4-9 align with our observations of speedup values. By looking at the serial fraction values of the adaptive models, it appears that each of them shows a near linear speedup for shifted and non-shifted test problems. Considering the mean serial fraction values one can notice that all $SF$ values are less than 1, and while the mean serial fraction values for $AdGAT_2$ are less than 0.5 for both adaptive regimes, $AdGAT_2$ maintains performance domination over the other adaptive models. The Figure 4-7 and Figure 4-8 show the serial fraction values of 50-dmentional tests of adaptive regime 100, 300 respectively.
Table 4-9: The Serial Fraction values of the adapted models AdGAT1, AdGAT2 and AdGAT3 of adaptive regime 100(300) over the dGAT1, dGAT2 and dGAT3 of the 50D benchmark problems.

<table>
<thead>
<tr>
<th>Method</th>
<th>$f_{Sph}$</th>
<th>$f_{Ros}$</th>
<th>$f_{Ras}$</th>
<th>$f_{Sch}$</th>
<th>$f_{Gri}$</th>
<th>$f_{Ack}$</th>
<th>Mean SF</th>
</tr>
</thead>
<tbody>
<tr>
<td>AdGAT1</td>
<td>0.83(0.85)</td>
<td>0.54(0.54)</td>
<td>0.39(0.47)</td>
<td>0.52(0.46)</td>
<td>0.98(1.04)</td>
<td>0.84(0.89)</td>
<td>0.62(0.64)</td>
</tr>
<tr>
<td>AdGAT2</td>
<td>0.35(0.36)</td>
<td>0.09(0.25)</td>
<td>1.26(1.34)</td>
<td>0.4(0.68)</td>
<td>0.41(0.43)</td>
<td>0.65(0.68)</td>
<td>0.32(0.37)</td>
</tr>
<tr>
<td>AdGAT3</td>
<td>1.15(1.24)</td>
<td>0.22(0.26)</td>
<td>0.95(0.87)</td>
<td>0.35(0.35)</td>
<td>1.42(1.58)</td>
<td>1.12(1.23)</td>
<td>0.59(0.62)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method</th>
<th>$f_{S-Sph}$</th>
<th>$f_{S-Ros}$</th>
<th>$f_{S-Ras}$</th>
<th>$f_{S-Sch}$</th>
<th>$f_{S-Gri}$</th>
<th>Mean SF</th>
</tr>
</thead>
<tbody>
<tr>
<td>AdGAT1</td>
<td>0.99(1.02)</td>
<td>-</td>
<td>1.03(1.02)</td>
<td>0.59(0.59)</td>
<td>0.4(0.43)</td>
<td>0.71(0.71)</td>
</tr>
<tr>
<td>AdGAT2</td>
<td>1.02(1.13)</td>
<td>-</td>
<td>0.26(0.25)</td>
<td>0.25(0.64)</td>
<td>0.37(0.39)</td>
<td>0.35(0.42)</td>
</tr>
<tr>
<td>AdGAT3</td>
<td>0.92(0.96)</td>
<td>-</td>
<td>0.78(0.8)</td>
<td>0.72(0.72)</td>
<td>0.92(0.96)</td>
<td>0.97(0.98)</td>
</tr>
</tbody>
</table>

Figure 4-5: The AdGAT1, AdGAT2 and AdGAT3 Speedup values of the (100) adaptive regime of the 50D benchmark test problems.
Figure 4-6: The AdGAT1, AdGAT2 and AdGAT3 Speedup values of the (300) adaptive regime of the 50D benchmark test problems.

Figure 4-7: The AdGAT1, AdGAT2 and AdGAT3 Serial Fraction values of the (100) adaptive regime of the 50D benchmark test problems.
Results on the 50D test functions seem to have provided extra evidence that the adaptive models tend to outperform the non-adaptive models, and that *AdGAT2* is superior to the other two adaptive models. The differences in performance were more marked than we saw on the 30D benchmarks. Next we turn to our experiments on the 100D versions of the benchmark functions.

With the intention of providing a deep look over the adaptive models, a statistical comparison has been used to check the performance of adaptive parallel models. We did paired 1-tailed T-Tests comparing the execution times of the various methods, focussing on the 50D benchmarks. For the sake of simplicity, we restrict our statistical analysis to the (100) adaptive regime. The statistical tests were done with respect to a cut-off significance level (p-value = 0.05) assuming unequal variance. Table 4-10 summarises the T-Test p-values for comparisons between *AdGAT1*, *AdGAT2* and *AdGAT3* on the 50D functions. The summary in Table 4-10 confirms that *AdGAT2* is superior in performance to the other adaptive models on all of the benchmarks for the 50D versions. We can also see that *AdGAT2*’s advantage over the other two is more marked for *AdGAT1* in the non-shifted cases, and more marked for *AdGAT3* in the shifted cases. However, the statistical results show that there is no significant difference between *AdGAT1* and *AdGAT3* for most of the benchmark functions.
Table 4-10: The 50D benchmarks paired 1-tailed T-Test p-values for the comparison between the execution times of adaptive models AdGAT1, AdGAT2 and AdGAT3

<table>
<thead>
<tr>
<th>Method</th>
<th>fSph</th>
<th>fRos</th>
<th>fRas</th>
<th>fSch</th>
<th>fGri</th>
<th>fAck</th>
</tr>
</thead>
<tbody>
<tr>
<td>AdGAT1- AdGAT2</td>
<td>0.004939</td>
<td>0.000158</td>
<td>0.000546</td>
<td>0.00259</td>
<td>0.000553</td>
<td>0.004797</td>
</tr>
<tr>
<td>AdGAT3- AdGAT2</td>
<td>0.009894</td>
<td>0.000386</td>
<td>0.008681</td>
<td>0.006111</td>
<td>0.006978</td>
<td>0.003747</td>
</tr>
<tr>
<td>AdGAT1- AdGAT3</td>
<td>0.082039</td>
<td>0.092774</td>
<td>0.070518</td>
<td>0.053417</td>
<td>0.049747</td>
<td>0.051552</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method</th>
<th>fS-Sph</th>
<th>fS-Ros</th>
<th>fS-Ras</th>
<th>fS-Sch</th>
<th>fS-Gri</th>
<th>fS-Ack</th>
</tr>
</thead>
<tbody>
<tr>
<td>AdGAT1- AdGAT2</td>
<td>0.021771</td>
<td>-</td>
<td>0.000911</td>
<td>0.003473</td>
<td>0.000572</td>
<td>0.009649</td>
</tr>
<tr>
<td>AdGAT3- AdGAT2</td>
<td>0.001246</td>
<td>-</td>
<td>0.000478</td>
<td>0.001214</td>
<td>0.000177</td>
<td>0.005047</td>
</tr>
<tr>
<td>AdGAT1- AdGAT3</td>
<td>0.085118</td>
<td>-</td>
<td>0.054845</td>
<td>0.055059</td>
<td>0.043013</td>
<td>0.067046</td>
</tr>
</tbody>
</table>

4.3.3.3 The 100-Dimensional Test Problems

This section illustrates the tests of the adaptive models with 100-Dimensional benchmark problems. Table 4-11 shows the success rates of the adaptive models for the 100D problems, while Table 4-12 shows the success rates for the non-adaptive models on the same problems.

Table 4-12 reveals that the non-adaptive models are significantly worse in the 100D problems. The performance degradation is less marked with the non-shifted (easier) benchmarks; for example dGAT was still able to solve fSch 16 times out of 20, and fAck 15 times out of 20.

Comparing Table 4-11 with Table 4-12, we can see that the performance of the adaptive models is clearly better than the non-adaptive models. Comparing the three adaptive models, one can notice that AdGAT2 still has performance dominance, while AdGAT3 and AdGAT1 again show similar performance to each other, although with
AdGAT1 slightly better than AdGAT3 for the non-shifted versions, and vice versa for the shifted versions.

Table 4-11: The AdGAT1, AdGAT2 and AdGAT3 success rates out of 20 to optimize the 100D benchmark functions with adaption regime 100(300).

<table>
<thead>
<tr>
<th>Method</th>
<th>fSph</th>
<th>fRos</th>
<th>fRas</th>
<th>fSch</th>
<th>fGri</th>
<th>fAck</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>AdGAT1</td>
<td>18(18)</td>
<td>17(16)</td>
<td>19(18)</td>
<td>17(17)</td>
<td>19(19)</td>
<td>19(19)</td>
<td>109(107)</td>
</tr>
<tr>
<td>AdGAT2</td>
<td>20(20)</td>
<td>17(17)</td>
<td>18(18)</td>
<td>20(19)</td>
<td>20(20)</td>
<td>20(20)</td>
<td>114(114)</td>
</tr>
<tr>
<td>AdGAT3</td>
<td>19(19)</td>
<td>17(16)</td>
<td>17(18)</td>
<td>19(17)</td>
<td>19(19)</td>
<td>20(20)</td>
<td>111(109)</td>
</tr>
</tbody>
</table>

Table 4-12: The dGAT1, dGAT2 and dGAT3 success rates out of 20 to optimize the 100D benchmark functions.

<table>
<thead>
<tr>
<th>Method</th>
<th>fSph</th>
<th>fRos</th>
<th>fRas</th>
<th>fSch</th>
<th>fGri</th>
<th>fAck</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>dGAT1</td>
<td>14</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>15</td>
<td>15</td>
<td>89</td>
</tr>
<tr>
<td>dGAT2</td>
<td>17</td>
<td>15</td>
<td>16</td>
<td>17</td>
<td>17</td>
<td>16</td>
<td>98</td>
</tr>
<tr>
<td>dGAT3</td>
<td>14</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>17</td>
<td>15</td>
<td>94</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method</th>
<th>fS-Sph</th>
<th>fS-Ros</th>
<th>fS-Ras</th>
<th>fS-Sch</th>
<th>fS-Gri</th>
<th>fS-Ack</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>dGAT1</td>
<td>20(20)</td>
<td>0(0)</td>
<td>17(17)</td>
<td>17(17)</td>
<td>19(19)</td>
<td>20(18)</td>
<td>93(91)</td>
</tr>
<tr>
<td>dGAT2</td>
<td>20(20)</td>
<td>0(0)</td>
<td>18(18)</td>
<td>18(18)</td>
<td>19(19)</td>
<td>20(18)</td>
<td>95(93)</td>
</tr>
<tr>
<td>dGAT3</td>
<td>20(20)</td>
<td>0(0)</td>
<td>17(17)</td>
<td>17(17)</td>
<td>18(18)</td>
<td>20(18)</td>
<td>92(90)</td>
</tr>
</tbody>
</table>
For example the total successful runs of $AdGAT2$ for not shifted benchmarks are 114(114) for the 100(300) adaptive scheme out of 120 (the total number of runs), while $AdGAT1$ has 109(107) and $AdGAT3$ has 111(109) successful runs out of 120 for the 100(300) adaptive scheme. The adaptive model $AdGAT2$ for shifted benchmark fitness functions has 95(93) successful runs for the 100(300) adaptive regime out of 120, while $AdGAT1$ shows 93(91) and $AdGAT3$ shows 92(90) for the 100(300) adaptive scheme. These results give a general view of the fixed and adaptive models performance for the 100D problems. However, the speedup and serial fraction of the mean execution times for successful runs have been detailed below.

Table 4-13 shows the speedup of the adaptive models over their corresponding non-adaptive versions, on the 100-D test problems, while Table 4-14 shows the corresponding serial fraction values.

<table>
<thead>
<tr>
<th>Method</th>
<th>$fS$-Sph</th>
<th>$fS$-Ros</th>
<th>$fS$-Ras</th>
<th>$fS$-Sch</th>
<th>$fS$-Gri</th>
<th>$fS$-Ack</th>
<th>Mean $S_N$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$AdGAT1$</td>
<td>1.86(1.97)</td>
<td>3.77(3.74)</td>
<td>2.13(2.59)</td>
<td>2(1.98)</td>
<td>1.19(1.36)</td>
<td>1.64(1.65)</td>
<td>2(2.21)</td>
</tr>
<tr>
<td>$AdGAT2$</td>
<td>2.61(3.06)</td>
<td>5.5(5.57)</td>
<td>2.41(2.36)</td>
<td>2.79(2.86)</td>
<td>2.3(2.56)</td>
<td>1.74(1.66)</td>
<td>2.89(3.01)</td>
</tr>
<tr>
<td>$AdGAT3$</td>
<td>2.04(2.03)</td>
<td>2.38(2.48)</td>
<td>2.59(2.93)</td>
<td>2.56(2.65)</td>
<td>1.51(1.76)</td>
<td>1.57(1.67)</td>
<td>2.11(2.25)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method</th>
<th>$fS$-Sph</th>
<th>$fS$-Ros</th>
<th>$fS$-Ras</th>
<th>$fS$-Sch</th>
<th>$fS$-Gri</th>
<th>$fS$-Ack</th>
<th>Mean $S_N$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$AdGAT1$</td>
<td>1.44(1.51)</td>
<td></td>
<td>1.95(2.16)</td>
<td>1.43(1.47)</td>
<td>1.8(1.88)</td>
<td>2.01(2.11)</td>
<td>1.73(1.83)</td>
</tr>
<tr>
<td>$AdGAT2$</td>
<td>2.01(2.06)</td>
<td></td>
<td>2.57(2.9)</td>
<td>2.49(2.51)</td>
<td>3.26(3.28)</td>
<td>2.63(2.63)</td>
<td>2.59(2.68)</td>
</tr>
<tr>
<td>$AdGAT3$</td>
<td>1.49(1.54)</td>
<td></td>
<td>1.66(1.7)</td>
<td>1.58(1.71)</td>
<td>1.26(1.32)</td>
<td>1.22(1.26)</td>
<td>1.44(1.51)</td>
</tr>
</tbody>
</table>

From Table 4-13 it appears that all adaptive models show a good performance over the fixed ones, where the speedup $S_N$ exceed 2, ($S_N \geq 2$), for most benchmark test functions; this confirms previous observations. The mean speedup values indicate that the adaptive models solve these problems generally twice as fast as their corresponding
non-adaptive versions; also, the (300) adaptive regime seems to achieve better migration control than the (100) regime in these cases. This observation about the migration regime conflicts with what we saw for the 50D tests, and is perhaps explained by the fact that the problems become considerably more challenging when we move from 50D to 100D and progress tends to be slower, and require longer performance sampling windows to achieve good migration control.

However, the mean speedup values from Table 4-13 reveal no appreciable difference between the two adaptive regimes, and we can tentatively include that adaptive regime (300) just gives a slight advantage over regime (100) in these cases. In addition, by comparing the speedup values between Table 4-8 and Table 4-13, we can see that the speedup between adaptive and non-adaptive models is more marked on the 100D problems than on the 50D problems. This observation is closely related to the significant deterioration in performance we see in the non-adaptive methods as we increase the dimensionality. For example the mean speedup values with the adaptive regime 100(300) of \textit{AdGAT1} on 50D problems are 1.55(1.5) for normal problems and 1.38(1.37) for the shifted versions, while for \textit{AdGAT1} on 100D problems the corresponding figures are 2.1(2.21) for the non-shifted problems and 1.73(1.83) for the shifted problems.

On the other hand, the adaptive model \textit{AdGAT2} still has domination in performance among the other adaptive models \textit{AdGAT1} and \textit{AdGAT3}. Also, the adaptive model \textit{AdGAT3} has a better performance than \textit{AdGAT1} for not shifted test problems, while the adaptive parallel model \textit{AdGAT1} perform better for the shifted benchmark functions. For example, the mean speedup value for normal versions of \textit{AdGAT2} is 2.89(3.01) for the adaptive regime 100(300) and it comes to be 2.1(2.21) for the \textit{AdGAT1} and 2.11(2.25) for \textit{AdGAT3}. Figure 4-9 and Figure 4-10 illustrate the speedup values of 100-Dmentional benchmarks of the adaptive scheme 100 and 300 respectively.

The serial fraction results shown in Table 4-14 provide additional evidence that aligns with the speedup values we have discussed. Again, the serial fraction values of the adaptive parallel models reveal that all adaptive models show near linear speedup for all test problems. By noticing the mean serial fraction values we notice that most \textit{SF} values are in the range of $[0.12, 0.83]$. For example, the mean serial fraction values for not shifted versions of \textit{AdGAT2} is 0.3(0.29) for the adaptive regime 100(300) while it
comes to be 0.51(0.47) for the AdGAT1 and 0.46(0.43) for AdGAT3. The Figure 4-11 and Figure 4-12 show the serial fraction values of 100-dimensional tests of adaptive regime 100, 300 respectively.

Table 4-14: The Serial Fraction values of the adapted models AdGAT1, AdGAT2 and AdGAT3 of adaptive regime 100(300) over the dGAT1, dGAT2 and dGAT3 of the 100D benchmark problems.

<table>
<thead>
<tr>
<th>Method</th>
<th>fSph</th>
<th>fRos</th>
<th>fRas</th>
<th>fSch</th>
<th>fGri</th>
<th>fAck</th>
<th>Mean SF</th>
</tr>
</thead>
<tbody>
<tr>
<td>AdGAT1</td>
<td>0.51(0.48)</td>
<td>0.22(0.22)</td>
<td>0.44(0.35)</td>
<td>0.47(0.47)</td>
<td>0.83(0.72)</td>
<td>0.59(0.58)</td>
<td>0.44(0.42)</td>
</tr>
<tr>
<td>AdGAT2</td>
<td>0.34(0.28)</td>
<td>0.13(0.12)</td>
<td>0.38(0.39)</td>
<td>0.32(0.31)</td>
<td>0.4(0.35)</td>
<td>0.55(0.57)</td>
<td>0.3(0.29)</td>
</tr>
<tr>
<td>AdGAT3</td>
<td>0.46(0.46)</td>
<td>0.38(0.36)</td>
<td>0.35(0.3)</td>
<td>0.35(0.34)</td>
<td>0.64(0.54)</td>
<td>0.61(0.57)</td>
<td>0.44(0.41)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method</th>
<th>fS-Sph</th>
<th>fS-Ros</th>
<th>fS-Ras</th>
<th>fS-Sch</th>
<th>fS-Gri</th>
<th>fS-Ack</th>
<th>Mean SF</th>
</tr>
</thead>
<tbody>
<tr>
<td>AdGAT1</td>
<td>0.67(0.64)</td>
<td>-</td>
<td>0.48(0.43)</td>
<td>0.68(0.66)</td>
<td>0.53(0.5)</td>
<td>0.46(0.44)</td>
<td>0.55(0.52)</td>
</tr>
<tr>
<td>AdGAT2</td>
<td>0.46(0.45)</td>
<td>-</td>
<td>0.35(0.3)</td>
<td>0.36(0.36)</td>
<td>0.26(0.26)</td>
<td>0.34(0.34)</td>
<td>0.34(0.33)</td>
</tr>
<tr>
<td>AdGAT3</td>
<td>0.65(0.63)</td>
<td>-</td>
<td>0.58(0.56)</td>
<td>0.61(0.56)</td>
<td>0.78(0.74)</td>
<td>0.81(0.78)</td>
<td>0.67(0.64)</td>
</tr>
</tbody>
</table>

Figure 4-9: The AdGAT1, AdGAT2 and AdGAT3 Speedup values of the (100) adaptive regime of the 100D benchmark test problems.
Figure 4-10: The AdGAT1, AdGAT2 and AdGAT3 Speedup values of the (300) adaptive regime of the 100D benchmark test problems.

Figure 4-11: The AdGAT1, AdGAT2 and AdGAT3 Serial Fraction values of the (100) adaptive regime of the 100D benchmark test problems.
Figure 4-12: The AdGAT1, AdGAT2 and AdGAT3 Serial Fraction values of the (300) adaptive regime of the 100D benchmark test problems.

With the intention of providing a deep look over the adaptive models, a statistical comparison has been used to check the performance of adaptive parallel models. We did paired 1-tailed T-Tests comparing the execution times of the various methods, focussing on the 100D benchmarks. For the sake of simplicity, we restrict our statistical analysis to the (100) adaptive regime. The statistical tests were done with respect to a cut-off significance level (p-value = 0.05) assuming unequal variance. Table 4-15 summarises the T-Test p-values for comparisons between AdGAT1, AdGAT2 and AdGAT3 on the 100D functions.

Table 4-15 confirms the superiority of AdGAT2 in the case of the 100D functions. Inspection of the p-values also confirms the following observation; as we move to higher dimensional problems, the superiority of AdGAT2 over the other models becomes accelerated. However, the statistical results show that there is no significant difference between AdGAT1 and AdGAT3 for most of the benchmark functions.
### 4.4 Conclusion

In this chapter we developed a general approach to adaptive migration in dGAs, in which the migration of chromosomes between demes is influenced by the current performance of the demes. That is, successful demes are asked to distribute their new best results to other demes more frequently. Three new dGAs were explored, AdGAT1, AdGAT2 and AdGAT3, respectively based on the three basic topologies introduced in Chapter III. Experiments were done to test and compare these models on a set of 36 benchmark functions, comprising 12 functions and the 30D, 50D and 100D versions of each. The experiments were done with physical topologies consisting of 16 clients with one master for each topology.

Depending on the benchmark fitness function dimensions and complexity, these tests separated into three different stages. The first stage of test has been done using the test problem with 30 dimensions each the second stage with 50 dimensions while the third stage has done with 100 dimensions. In the first stage, we used the successful run rate to compare between the adaptive and fixed models. The results come with no big clarification of any difference between the adaptive models and the fixed ones. These results give a brief review of difference between the studied models (the fixed and

---

**Table 4-15: The 100D benchmarks paired 1-tailed T-Test p-values for the comparison between the execution times of adaptive models AdGAT1, AdGAT2 and AdGAT3**

<table>
<thead>
<tr>
<th>Method</th>
<th>fSph</th>
<th>fRos</th>
<th>fRas</th>
<th>fSch</th>
<th>fGri</th>
<th>fAck</th>
</tr>
</thead>
<tbody>
<tr>
<td>AdGAT1-AdGAT2</td>
<td>0.00147</td>
<td>0.000421</td>
<td>0.000745</td>
<td>0.000259</td>
<td>0.000453</td>
<td>0.000843</td>
</tr>
<tr>
<td>AdGAT3-AdGAT2</td>
<td>0.004593</td>
<td>0.001393</td>
<td>0.00105</td>
<td>0.000153</td>
<td>0.003161</td>
<td>0.001073</td>
</tr>
<tr>
<td>AdGAT1-AdGAT3</td>
<td>0.051231</td>
<td>0.043021</td>
<td>0.039924</td>
<td>0.063195</td>
<td>0.011885</td>
<td>0.030217</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method</th>
<th>fS-Sph</th>
<th>fS-Ros</th>
<th>fS-Ras</th>
<th>fS-Sch</th>
<th>fS-Gri</th>
<th>fS-Ack</th>
</tr>
</thead>
<tbody>
<tr>
<td>AdGAT1-AdGAT2</td>
<td>0.001046</td>
<td>-</td>
<td>0.001541</td>
<td>0.004614</td>
<td>0.001086</td>
<td>0.007654</td>
</tr>
<tr>
<td>AdGAT3-AdGAT2</td>
<td>0.000265</td>
<td>-</td>
<td>0.000303</td>
<td>0.000825</td>
<td>0.019376</td>
<td>0.001142</td>
</tr>
<tr>
<td>AdGAT1-AdGAT3</td>
<td>0.057038</td>
<td>-</td>
<td>0.021051</td>
<td>0.013325</td>
<td>0.080075</td>
<td>0.046217</td>
</tr>
</tbody>
</table>
adaptive ones) of how their performance will change if the adaptive mechanism employed. The mean speedup and serial fraction values showed that the adaptive model demonstrating a small enhancement in their performance with the shifted version of benchmark functions than normal versions.

In the second stage (50D), adaptive model performed better than the fixed one with all benchmark problems, the successful run rate, speedup and serial fraction confirm these results. However, $AdGAT_2$ has the superiority in the performance than other adaptive models, while model $AdGAT_1$ show a better performance with normal test function and model $AdGAT_3$ demonstrates a better performance with the shifted models. The adaptive scheme 100 proves a better control than the 300 scheme for most benchmark problems.

In the third stage (100D), the adaptive model enhanced their performance to show more superiority than the fixed models with all benchmark problems, the successful run rate, speedup and serial fraction confirm these results. Also, the adaptive parallel model $AdGAT_2$ has the performance domination over the other adaptive models, while model $AdGAT_1$ show a better performance with normal test function and model $AdGAT_3$ demonstrates a better performance with the shifted models. In this stage of test, the adaptive scheme 300 has a better control than the 100 scheme for most benchmark problems.

Finally, statistical analyses were done to establish the significance of the above observations. This was done by using paired 1-tailed T-Test with cut-off significance level ($p$-value $= 0.05$). The T-Test outcomes confirmed our basic observations, showing that the adaptive parallel model $AdGAT_2$ has superior performance to $AdGAT_1$ and $AdGAT_3$ for all benchmark problems with the 50D and 100D cases. Also, the T-Test values confirm that there is no significant difference between $AdGAT_1$ and $AdGAT_3$ for most test functions.

In the next chapter, we will explore an alternative approach to adaptive search in the dGA context. In this case, adaptation of migration will be related to the levels of deme diversity, rather than levels of deme performance.
CHAPTER V
INVESTIGATING DIVERSITY-BASED GUIDANCE
IN DISTRIBUTED GENETIC ALGORITHMS

One of the key issues in the design and control of EAs is that of maintaining population diversity. In this context, diversity usually relates to how different from each other the individual chromosomes in the EA's population are. The improved performance of dEAs is commonly attributed to the fact that they are naturally better at maintaining diversity in the population. Nevertheless the standard approaches still tend to delay premature convergence, rather than eliminate it. Maintaining high levels of diversity within the population is generally found to be significantly helpful for a wide range of problems, and especially for optimization of dynamic problems. In dynamic problems, maintaining high population diversity enhances the possibilities of reaching the peak when the problem landscape is changed. The importance of diversity in EAs naturally leads to suggestions for adaptation of parameters and other aspects, based on population diversity. That is, if diversity is too low, we need to take action to increase it, and if diversity is too high, we may benefit from reducing it.

In this chapter we therefore consider diversity-guidance in the context of the dGAs that we began investigating in Chapter III.

5.1 A Brief Overview of Diversity-Guided Control in EAs

The basic idea of diversity-guided control in an EA is to monitor the diversity of the population, and take action when the diversity level has moved below a threshold. This action would be designed to lead to a boost in diversity - e.g. either certain parameters of the algorithm will be changed, or new candidate solutions are introduced into the population, or both. Mauldin [164] was among the first to describe the concept of controlling the diversity level of the population in such ways, to maintain a balance between exploration and exploitation.

Prominent and successful examples of such approaches include the work of Ursem et al. [246, 247]. In [246], excellent results were achieved by using signals from population diversity monitoring, based on which the EA would switch between two
quite distinct modes of operation, favouring exploitation and exploration respectively. Meanwhile, an interesting and recent approach is by [42]. In this approach, based on signals from diversity monitoring, 'diversified artificial chromosomes’ (DACs) are generated and introduced to boost diversity at particular times. Another approach is the Shifting-Balance Genetic Algorithm [184]. The SBGA calculates a so-called containment factor between two subpopulations, which is based on Hamming distances between all members of the two populations. The distance is calculated between each member of population A and all members of population B. The factor determines the ratio between individuals selected on fitness and individuals selected to increase the distance between the two populations. While the third, and more distantly related, approach is the Forking Genetic Algorithm FGA, which uses specialized diversity measures to turn a subset of the population into a subpopulation [245]. The Diversity-Control-Oriented Genetic Algorithm (DCOGA) [221, 222] uses a diversity measure based on Hamming distance to calculate a survival probability for the individuals. A low Hamming distance between the individual and the current best individual is translated into a low survival probability. Though, the diversity level is used within the selection method.

Meanwhile, in the context of dEAs, diversity-guided adaptation is less explored. This is probably because premature convergence is less often seen as an issue for dEAs, although, the problem still exists and is revealed when addressing particularly difficult or high dimensional problems. Methods for diversity guided control of search behaviour, such as those described briefly above, have shown significant benefit in single-processor EAs. However diversity-guided control in dEAs is an open issue that is little explored as yet. In this study we therefore explore the use of a diversity-guided control mechanism in a dEA.

5.2 Diversity-Guided Control in Distributed Evolutionary Algorithm

The approach to diversity-guided control in Chang et al. [42] centred on the use of 'diversified artificial chromosomes’ (DAC). These chromosomes are injected into the population when diversity is too low, and the injection stops when diversity has reached a suitable level. We need to define how this process could operate within a dEA, which
has several subpopulations interconnected by a certain topology. We also need to define diversity metric, and we need to set thresholds of this metric for determining when to start and stop injecting DACs and we need to define how we create the DACs. We deal with these issues in the following subsections.

5.2.1 Monitoring Diversity and Injection of DACs

As discussed, population diversity is a key issue in EAs, and to balance the levels of exploration and exploitation appropriately, suitable control of population diversity can be considered essential during the evolutionary process [252], and can significantly raise the chances of finding global optima.

The essential idea in our approach is to measure a client's diversity level during the evolutionary process and once the diversity drops down to a threshold level, the master process will produce some artificial chromosomes, ACs, and inject them into that client. The intention is that this will help rapidly increase the population diversity in that client. The evolutionary processes will reduce the diversity level again, however, the using of control mechanism will cause artificial chromosomes be re-generated whenever the diversity level drops below the threshold level.

The basic element of diversity measurement is a distance metric that gives the distance between any two chromosomes. In common with [42], Chang et al. used Hamming distance for this purpose. The Hamming distance between chromosomes \( c_1 \) and \( c_2 \) is the proportion of genes for which the values in \( c_1 \) and \( c_2 \) are different. We define population diversity as the mean Hamming distance over all distinct pairs of chromosomes in the population. This is updated continually in each subpopulation.

In the remainder, we use the same terminology to describe Diversified Artificial Chromosomes (DACs) and associated constructs as is used in [42] adapting these terms in the context of our distributed architecture. DACs emerge from an 'archive pool' maintained by the master process. The master process, as we will see, continually receives updates from each island indicating a new (locally) best chromosome, and maintains an archive containing the most recent 100 such chromosomes (covering all islands). When the archive is full, the next updated chromosome received will overwrite the current worst in the archive, if it has better (or equal) fitness to this current worst. When it is determined that an island needs an injection of a DAC (when the diversity level of that island falls between the predefined lower thresholds \( d_{\text{high}} \) and upper \( d_{\text{low}} \)), a single chromosome is selected from the archive pool, and injected into the island, where
it will overwrite the current worst fit chromosome in each of the two subpopulations in that island.

When injecting a DAC into island $d$, the choice of DAC to select from the archive is as follows: we draw a sample of 10 from the archive pool that came from islands other than $d$, and choose the one most distant in Hamming distance from the current best in $d$, breaking ties randomly. Finally, when the diversity of an island falls below the lower threshold, the subpopulations in that island are both reinitialised by using the archive pool. This is done by using the same population regeneration mechanism used in the underlying EA, filling the new subpopulations with children of crossover and mutation bred from the archive pool.

5.2.2 An artificial chromosome generation mechanism

As discussed in the above section, the generation of ACs will control diversity and convergence in the population. Our approach to building and maintaining the AC archive is to collect high-fitness recent chromosomes generated in different sub-populations, and use these latter to generate artificial chromosomes with good diversity. Initially, the archive pool is filled from the series of new local best chromosomes encountered in each subpopulation. When the master archive size becomes to be equal to the population size, new current best chromosomes are incorporated into the archive by first removing the chromosome with the lowest diversity (i.e. the chromosome in the archive whose mean distance to all others in the archive is smallest).

All the archived chromosomes in the master’s archive pool will be sorted into 10 groups, which will be used latter to generate artificial chromosome. Each group will then nominate one chromosome based on the diversity of each chromosome in same group. After selecting 10 chromosomes from all groups, the process will crossover and mutate each two chromosomes to generate 20 new offspring. These 30 chromosomes are representing the artificial chromosomes; a competition selection procedure will select one chromosome from regular chromosomes and the other from artificial chromosomes for crossover and mutation.

Afterward, chromosomes with better fitness evaluation will be reserved in the archive. Consequently, chromosomes with higher diversity are preserved to the next generation. Following is the pseudocode to generate the ACs:
Set $n = \frac{\text{archive poop size}}{10}$; 
Initial $\text{chromo\_group}[10,n]$; 
Initial $\text{sub\_group1}[10]$; 
Initial $\text{sub\_group2}[20]$; 
Initial $\text{chromo1}$; 
Initial $\text{chromo2}$; 
Initial $\text{dac}$; 
Sort archive pool into $\text{chromo\_group}$; 
For each group in $\text{chromo\_group}$ 
\quad Calculate diversity; 
\quad Find best diversified chromosome; 
\quad Save the diversified chromosomes in $\text{sub\_group1}$; 
End for 
Apply mutation and crossover for $\text{sub\_group1}$ and save the results in $\text{sub\_group2}$; 
For each chromosome in $\text{sub\_group1}$ & $\text{sub\_group2}$ 
\quad Calculate fitness; 
End for; 
Select one chromosome from archive and save it in $\text{chromo1}$; 
Select one chromosome from $\text{sub\_group1}$ & $\text{sub\_group2}$ and save it in $\text{chromo2}$; 
Apply mutation and crossover for $\text{chromo1}$ & $\text{chromo2}$ to produce $\text{chromo1'}$ & $\text{chromo2'}$; 
Calculate fitness for $\text{chromo1'}$ & $\text{chromo2'}$; 
Set $\text{dac} = \text{best fitness (chromo1, chromo2, chromo1', chromo2')}$; 
Send $\text{dac}$ to worst client; 
Replace worst chromosome in archive pool with $\text{dac}$;
5.2.3 Integrating DAC-Injection Based Diversity Control In The dEA

The overall flow of control in \textit{DGDGA} is characterised as follows:

- While each island evolves independently, every time the local best so-far improves on one of the island’s subpopulations, a copy is migrated to its neighbour, and to the master process.
- Every chromosome received by the master process is added by the master process into an archive pool, which will be used to generate diversified artificial chromosomes (DAC).
- Each island regularly calculates its diversity level and communicates this to the master process (an island’s diversity is the minimum of the diversity levels for its two subpopulations).
- The master process selects DACs and sends them to all sub-populations with diversity level below $d_{\text{high}}$.
- When an island receives a DAC, it injects it into both of its subpopulations, and each overwrites the chromosome in that subpopulation with worst fitness.
- If a sub-populations diversity level drops below the minimum threshold level $d_{\text{low}}$, the subpopulation is reinitialised.

Effectively, each subpopulation in the \textit{DGDGA} switches between three modes of operation. When its diversity level is above the predefined threshold $d_{\text{high}}$, it operates a standard evolutionary algorithm. In this standard mode, it communicates its diversity level and new best chromosome to the master process, but never receives DACs to inject. When its diversity level is below this threshold, but above $d_{\text{low}}$, it will continually receive fresh injections of DACs from the master process. But, if the diversity level falls below $d_{\text{low}}$, it will be reinitialised by breeding from the master’s archive pool.

Through the dynamic control of population diversity, \textit{DGDGA} will inject the DAC into the population to prevent premature convergence and extend the searching spaces which are unexplored in the traditional GA for locating a better solution. As observed in the evolutionary process, when the fitness reaches a local optimum, the chromosomes within the last couples of population will be very homogenous. Thus, the genetic operators cannot further generate better chromosomes to jump away from the local optimum. It also indicates that the diversity of chromosome at this moment is very low.
5.2.4 Population Diversity Measures

Diversity measures are strongly problem-dependent. There are basically two distinct ways to measure the diversity of an individual. First, we can measure the difference or distance between two genotypes, while the other approach is to use a structural variation measure based on a mathematical foundation (e.g. entropy). There are many different diversity measure approaches, such as Hamming distance, Euclidean distance, and Information entropy. The Hamming distance metric has been commonly used, since it has particularly wide applicability. We use Hamming distance in our approach, and for convenience we define Hamming distance below.

The Hamming distance is employed between two arrays of equal dimension. To measure the individual diversity, we compare each gene with the best fitness chromosome and others chromosomes. The parameter \((I)\) is an indicate function which gives the total number of values where \(x_i \neq y_j\), the Hamming distance can be formulated as:

\[
\text{Hamming Distance}(X,Y) = \frac{1}{L} \sum_{l=1}^{L} I(X,Y), \quad L = 1,\ldots,n
\]  

\[
I(X,Y) = \sum_{j=0}^{i} l_j, \quad l_j = \begin{cases} x_j = y_j, & 0 \\ x_j \neq y_j, & 1 \end{cases}
\]  

5.3 The Algorithm Design \textit{DGdGA}

In essence, our DGdGA framework defines a set of groups of clients, and each client operates as both a master and a client within its group, while an overall master thread operates over the groups. The overall responsibilities of the master threads are as follows:

1. Establish connections between the clients
2. Establish and initialize parameters
3. Start and Terminate the optimization
4. Receive and store up to date data from clients
5. Distribute appropriate data to clients
6. Monitor and control population diversity

The client threads operate as follows.

1. Establish connections with master.
2. Establish and initialize parameters
3. Establish and initialize population
4. Run evaluation
5. Receive and store up to date data from master.
6. Distribute appropriate data to master (individuals and diversity level).

The overall flow of control in **DGdGA** is characterised as follows:

- Each island initializes its migration rate frequency (n). This is the number of generations it waits between making a copy of its local best so-far chromosome and migrating that copy to its neighbour, and to the master process.
- While each island evolves independently, every time the local best so-far improves on one of the island’s subpopulations, a copy is migrated to its neighbour, and to the master process.
- Every chromosome received by the master process is added by the master process into an archive pool, which will be used to generate diversified artificial chromosomes (DAC).
- Each island regularly calculates its diversity level and communicates this to the master process (an island’s diversity is the minimum of the diversity levels for its two subpopulations)
- The master process selects DACs and sends them to all sub-populations with diversity level below \( d_{high} \).
- When an island receives a DAC, it injects it into both of its subpopulations, and each overwrites the chromosome in that subpopulation with worst fitness.
- If a sub-populations diversity level drops below the minimum threshold level \( d_{low} \), the subpopulation is reinitialised.

The following pseudo-code describes the model algorithm for master and client.
**Algorithm 5.1:** Pseudo code of a dGA on both master and client sides

**Master Process:**

**Thread1:**
Connect Master with Clients;
For each Clients
    Send all parameters;
    Send “Start” token;
    **Initialize Current-Diversity-Level**;
End for

**Set Diversity Levels:** $d_{low} \& d_{high}$;
Run Thread2; // receive data
Run Thread3; // send data
Run Thread4; // Diversity Control

**Thread2:**
Repeat
    For each Client
        Receive individuals from clients
        If Optimal-individual = Null
            **Set Current-Diversity-Level**
            Set Optimal-individual;
            Set Optimal-Fitness-Value;
        Else
            if new individual better than Optimal-individual
                Set Optimal-individual;
                Set Optimal-Fitness-Value;
            End if
            if new-Diversity-Level ≠ Current-Diversity-Level
                Update Current-Diversity-Level;
            End if
            if new-Diversity-Level > $d_{high}$
                Add Client-Individual to Diversity-Pool;
            End if
        End if
    End for;
Until reach one of stopping criteria;
Thread3:
Repeat
   If Optimal-individual updated
       Send Optimal-individual to worst client;
   End if;
   Wait;
Until reach one of stopping criteria;

Thread4:
While Thread2 is running
   For each Client
      If Current-Diversity-Level ∈ ]d_{low}, d_{high}]
         Generate and Send DAC to worst Diversified client;
      End if;
      Wait;
      
      If Current-Diversity-Level ≤ d_{low}
         Reinitialize client population by sending “Reinitialize” token to this client;
      End if;
      Wait;
   End for;
Loop

Client Process:

Thread 1: // connect and receive data from master
Connect with master/client
If “Start” token received
   Run Thread2; // receive data from master/client
   Run Thread3; // run GA process
   Run Thread4; // send data to master/client
   Run Thread5; // Diversity control
End if

Thread 2:
While connected with master/client
   receive individuals from master/client;
   Set injected = false;
If new individual is better than Optimal-individual
update Optimal-individual;
update Optimal-Fitness-Val;
add individual to population
Set injected = true;
remove worst individual from population
End if
If new-Diversity-Level > Current-Diversity-Level
update Current-Diversity-Level;
if not injected
    add individual to population
    remove worst individual from population
End if
End if
End while

Thread 3:
Initialize Population;
Set Optimal-individual;
Set Optimal-Fitness-Val;
Initialize Current-Diversity-Level;
Repeat
    Count++;
    For selected c1, c2
        crossover(c1,c2); // ==> c1’ and c2’
        with Pm mutate(c1’); // ==> c1"
        with Pm mutate(c2’); // ==> c2"
        find fit(c1")
        find fit(c2")
    If c1" is better than Optimal-individual
        update Optimal-individual; // <= c1"
        update Optimal-Fitness-Val; // <= fit(c1")
        add c1" to population;
        remove worst individual from population;
    End if
    If c2" is better than Optimal-individual
        update Optimal-individual; // <= c2"
        update Optimal-Fitness-Val; // <= fit(c2")
        add c2" to population;
        remove worst individual from population;
    End if
    new-population = best 70% of current population;
new-population += 30% new chromosomes;
population = new-population;
Until reach one of stopping criteria;

Thread 4:
While Thread3 is running
  If Send-New-Data
    send Optimal-individual to master/client;
    Send-New-Data = False;
    wait;
  End if;
End while;

Thread 5:
While Thread3 is running
  If new-Diversity-Level > current-Diversity-Level
    Update current-Diversity-Level;
    Send new diversified individual to master
  End if;
  If “Reinitialize” Token Received
    Reinitialize population
  End if;
Loop;

Here, in comparison between Algorithm 3-1, Algorithm 4-1 and Algorithm 5-1, lots of modifications and updates can be noticed on the both master’s and client’s side. These differences mainly appear in **Thread4** for master and **Thread5** on client process. These modifications are responsible for diversity control, where other changes appear in bold on the both sides. The Diversity control depends on clients’ local optima and/or best new diversified individuals. Each time a client sends a new chromosome to the master, the master registers it as a new local optimum for that client and/or a new diversified individual. If this new local optimum is a new global optimum, then this will be sent to the worst client, while it will be added to the diversity pool if it considered as a new diversified individual.

Meanwhile, the master keeps a continuous diversity level measurement for each client. In case the diversity level for any client enters the diversity threshold interval
Then master will generate a new DAC from the diversity pool and send it to that client. If the diversity level for any client becomes $d_{low}$ then that client will be forced to reset its population. The diversity pool maximum capacity is determined to be 100 individuals. Thus, if a new diversified chromosome is nominated to be added to the diversity pool, the worst diversified individual will be removed from the pool.

5.4 Experiments

5.4.1 Benchmark Test Functions

In this chapter, we continue to test our algorithms with the set of benchmark functions used in chapters III and IV, and (as in Chapter IV); we will perform experiments on 30D, 50D and 100D variants of the functions.

5.4.2 The Distributed GA Parameters

Before presenting the results of our exploration of diversity-guided adaptive migration schemes, we set out here the parameters and other aspects of the experiments in this chapter. The used population size for all models is 50 individuals per subpopulation. The probability of update of an individual by mutation (mutation rate) is $P_m=0.05$. The crossover probability is $P_c=0.6$. Two stopping criteria have been utilized to terminate the whole process for all clients. The first criterion is GA work imposed a predefined maximum number of generation $max_{gen}=10000$, while the second one is to reach the predefined fitness optimum values for each benchmark fitness function which are shown in Table B-1 and Table B-2.

The computing system is built from a cluster of eight personal computers with additional separate one is reserved for the master process. Thus, each machine of the cluster holds two clients. These machines run Microsoft Windows XP Professional SP3, an Intel Pentium IV 2.99 GHz processor, and 2 GB of memory. The machines are interconnected by a Fast-Ethernet (100 Mbps) network. The used platform is Microsoft Visual Studio 2012, Version 12.0.30729.1 SP, Microsoft .NET Framework Version 4.5 SP1. Each individual experiment (applying a specific algorithm to a specific optimization problem) was repeated 20 times with independent random seeds. Results focus on the trials that reach target fitness within the maximum number generations.
Finally, Table 5-1 summarises the parameters and other aspects of the experimental setup.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithm</td>
<td>$DGdGAT1, DGdGAT2, DGdGAT3$</td>
</tr>
<tr>
<td>Population Size</td>
<td>50 x 2</td>
</tr>
<tr>
<td>Maximum Generation</td>
<td>10000</td>
</tr>
<tr>
<td>Number of Runs</td>
<td>20</td>
</tr>
<tr>
<td>Number of Sub-Populations</td>
<td>16</td>
</tr>
<tr>
<td>Problem Dimensions</td>
<td>30D, 50D, 100D</td>
</tr>
<tr>
<td>Crossover Rate</td>
<td>0.6</td>
</tr>
<tr>
<td>Mutation Rate</td>
<td>0.05</td>
</tr>
<tr>
<td>Diversity Thresholds ($d_{low}, d_{high}$)</td>
<td>$d_{low} \in {0.1, 0.2, 0.3, 0.4}$</td>
</tr>
<tr>
<td></td>
<td>$d_{high} \in {0.5, 0.6, 0.7, 0.8}$</td>
</tr>
<tr>
<td>Diversity Measurement</td>
<td>Hamming Distance</td>
</tr>
<tr>
<td>Selection Operator</td>
<td>roulette wheel selection</td>
</tr>
<tr>
<td>Mutation Operators</td>
<td>uniform, non-uniform, multiple uniform and boundary mutation</td>
</tr>
<tr>
<td>Crossover Operator</td>
<td>one-point, two-point, m-points and fuzzy connective-based crossover</td>
</tr>
<tr>
<td>Objective</td>
<td>12 Benchmark fitness functions</td>
</tr>
</tbody>
</table>

5.4.3 The Experimental Results

The main decision parameter that controls the adaptive migration process in our new scheme is the diversity threshold interval (which we simply call 'diversity threshold'). We experiment with 16 different settings for this interval, all within the
range of $[0.1, 0.8]$, and which are detailed in Table 5-1. The diversity threshold dictates how the master processes adapt the migration rates from the clients.

5.4.3.1 The 30-Dimensional Test Problems

Table 5-2 shows the success rates of our diversity-guided models on the 30D problems. For clarity, we used two columns to demonstrate the results for each model. The first column shows the total successful runs out of 240 (20 runs each for all 12 benchmark functions). The second column gives the mean value over the 12 problems for the percentage of successful trials. Figure 5-1 presents the first column graphically.

Table 5-2: The $DGdGAT_1$, $DGdGAT_2$ and $DGdGAT_3$ success rates out of 240 to optimize the 30D x 12 benchmark test functions using diversity threshold $d_{\text{min}} \in [0.1, 0.4] \& d_{\text{max}} \in [0.5, 0.8]$.

<table>
<thead>
<tr>
<th>Diversity Threshold</th>
<th>Method</th>
<th>$DGdGAT_1$</th>
<th></th>
<th></th>
<th>$DGdGAT_2$</th>
<th></th>
<th></th>
<th>$DGdGAT_3$</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Total</td>
<td>Mean %</td>
<td>Total</td>
<td>Mean %</td>
<td>Total</td>
<td>Mean %</td>
<td>Total</td>
<td>Mean %</td>
<td>Total</td>
<td>Mean %</td>
</tr>
<tr>
<td>0.1 - 0.5</td>
<td>203</td>
<td>84.58</td>
<td>202</td>
<td>84.17</td>
<td>201</td>
<td>83.75</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.1 - 0.6</td>
<td>203</td>
<td>84.58</td>
<td>207</td>
<td>86.25</td>
<td>203</td>
<td>84.58</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.1 - 0.7</td>
<td>209</td>
<td>87.08</td>
<td>216</td>
<td>90</td>
<td>207</td>
<td>86.25</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.1 - 0.8</td>
<td>211</td>
<td>87.92</td>
<td>217</td>
<td>90.42</td>
<td>209</td>
<td>87.08</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.2 - 0.5</td>
<td>221</td>
<td>92.08</td>
<td>220</td>
<td>91.67</td>
<td>225</td>
<td>93.75</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.2 - 0.6</td>
<td>223</td>
<td>92.92</td>
<td>227</td>
<td>94.58</td>
<td>226</td>
<td>94.17</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.2 - 0.7</td>
<td>231</td>
<td>96.25</td>
<td>234</td>
<td>97.50</td>
<td>231</td>
<td>96.25</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.2 - 0.8</td>
<td>231</td>
<td>96.25</td>
<td>233</td>
<td>97.08</td>
<td>232</td>
<td>96.67</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.3 - 0.5</td>
<td>221</td>
<td>92.08</td>
<td>217</td>
<td>90.42</td>
<td>216</td>
<td>90</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.3 - 0.6</td>
<td>224</td>
<td>93.33</td>
<td>219</td>
<td>91.25</td>
<td>220</td>
<td>91.67</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.3 - 0.7</td>
<td>225</td>
<td>93.75</td>
<td>229</td>
<td>95.42</td>
<td>225</td>
<td>93.75</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.3 - 0.8</td>
<td>228</td>
<td>95</td>
<td>229</td>
<td>95.42</td>
<td>225</td>
<td>93.75</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.4 - 0.5</td>
<td>192</td>
<td>80</td>
<td>195</td>
<td>81.25</td>
<td>192</td>
<td>80</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.4 - 0.6</td>
<td>192</td>
<td>80</td>
<td>197</td>
<td>82.08</td>
<td>193</td>
<td>80.42</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.4 - 0.7</td>
<td>198</td>
<td>82.5</td>
<td>203</td>
<td>84.58</td>
<td>199</td>
<td>82.92</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.4 - 0.8</td>
<td>204</td>
<td>85</td>
<td>208</td>
<td>86.67</td>
<td>205</td>
<td>85.42</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

From Table 5-2 we can notice that most problems have been successfully solved in rate >80% for most all models. However, the most effective diversity thresholds are
centralized in the range $d_{\min} = 0.2$, 0.3 and $d_{\max} = 0.7$, 0.8, where for example, the highest success rate 97.5% is achieved by the **DGdGAT2** with the diversity levels (0.2, 0.7). In fact, all three models show good performance with diversity thresholds in the region of $d_{\min} = 0.2$ and $d_{\max} = 0.7$, 0.8, with success rate $> 96\%$, while the worst results appear in the diversity range $(0.4, 0.5)$, for example, the success rate is 80% for **DGdGAT1**, 81.25% for **DGdGAT2**, and 80% for **DGdGAT3**. Note, the red coloured values in Table 5-2 indicate the mean success rate $\geq 95\%$.

When we compare the results for these diversity guided models with the non-adaptive models (Chapter III) we can notice that the diversity guided models seem to perform a little better, but not substantially better. For example, the non-adaptive models optimized the benchmark fitness functions with success rates in the rate range $[91.25, 94.17]$, while the diversity guided model improved upon this range with the diversity levels $d_{\min} = 0.2$ & $d_{\max} = 0.7$, 0.8. On other hand, the adaptive models (Chapter IV) exceed this with a minimum success rate of 97.5%. By inspection of Figure 5-1 we can see how the performance of the diversity guided models changes, for example the best performance of the three models **DGdGAT1**, **DGdGAT2** and **DGdGAT3** appear as we previously mentioned with diversity threshold $d_{\min} = (0.2, 0.3)$ & $d_{\max} = (0.6, 0.7, 0.8)$.

![Figure 5-1: The DGdGAT1, DGdGAT2 and DGdGAT3 success rates out of 240 to optimize the 30D x 12 benchmark test functions using diversity threshold $d_{\min} \in [0.1, 0.4]$ & $d_{\max} \in [0.5, 0.8]$.](image)

114
Table 5-3: The Speedup values of the diversity guided models $DGdGAT1, DGdGAT2$ and $DGdGAT3$ against the fixed models $dGAT1, dGAT2$ and $dGAT3$ of 30D benchmark problems.

<table>
<thead>
<tr>
<th>Diversity Threshold</th>
<th>$DGdGAT1$</th>
<th>$DGdGAT2$</th>
<th>$DGdGAT3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>min - max 0.1 - 0.5</td>
<td>0.98</td>
<td>1.09</td>
<td>0.97</td>
</tr>
<tr>
<td>0.1 - 0.6</td>
<td>0.99</td>
<td>1.11</td>
<td>1.05</td>
</tr>
<tr>
<td>0.1 - 0.7</td>
<td>1.03</td>
<td>1.16</td>
<td>1.11</td>
</tr>
<tr>
<td>0.1 - 0.8</td>
<td>1.06</td>
<td>1.17</td>
<td>1.11</td>
</tr>
<tr>
<td>0.2 - 0.5</td>
<td>1.07</td>
<td>1.14</td>
<td>1.02</td>
</tr>
<tr>
<td>0.2 - 0.6</td>
<td>1.09</td>
<td>1.17</td>
<td>1.04</td>
</tr>
<tr>
<td>0.2 - 0.7</td>
<td>1.13</td>
<td>1.363</td>
<td>1.11</td>
</tr>
<tr>
<td>0.2 - 0.8</td>
<td>1.2</td>
<td>1.365</td>
<td>1.22</td>
</tr>
<tr>
<td>0.3 - 0.5</td>
<td>0.96</td>
<td>1.01</td>
<td>0.91</td>
</tr>
<tr>
<td>0.3 - 0.6</td>
<td>0.91</td>
<td>0.96</td>
<td>0.93</td>
</tr>
<tr>
<td>0.3 - 0.7</td>
<td>0.91</td>
<td>0.97</td>
<td>0.91</td>
</tr>
<tr>
<td>0.3 - 0.8</td>
<td>0.97</td>
<td>1.08</td>
<td>0.94</td>
</tr>
<tr>
<td>0.4 - 0.5</td>
<td>0.78</td>
<td>0.8</td>
<td>0.7</td>
</tr>
<tr>
<td>0.4 - 0.6</td>
<td>0.78</td>
<td>0.83</td>
<td>0.72</td>
</tr>
<tr>
<td>0.4 - 0.7</td>
<td>0.81</td>
<td>0.87</td>
<td>0.8</td>
</tr>
<tr>
<td>0.4 - 0.8</td>
<td>0.94</td>
<td>0.97</td>
<td>0.95</td>
</tr>
</tbody>
</table>

Initial impressions therefore suggest that the diversity-guided scheme used in the new algorithms in this chapter can show a modest improvement over the non-adaptive models from Chapter III; however, they seem not as effective as the approach we used in Chapter IV. Meanwhile, it is interesting that the model T2 topology remains dominant in this context. In Table 5-3 we can see the speedup of the new diversity-guided models over their corresponding non-adaptive models, and in Table 5-4 we can see the serial fraction figures for the diversity-guided models.
From Table 5-3 we can notice again that the advantage of the diversity-guided models over the non-adaptive models is quite small, at least in the case of these 30D problems. We can also see that the speedup values for the three models in the range [0.7, 1.365]; in the best case, for example, $DGdGAT1$ manages 20% speedup and $DGdGAT2$ manages 33% speedup.

<table>
<thead>
<tr>
<th>Diversity Threshold</th>
<th>Method</th>
<th>Method</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>min - max</td>
<td>$DGdGAT1$</td>
<td>$DGdGAT2$</td>
<td>$DGdGAT3$</td>
</tr>
<tr>
<td>0.1 - 0.5</td>
<td>1.02</td>
<td>0.91</td>
<td>1.03</td>
</tr>
<tr>
<td>0.1 - 0.6</td>
<td>1.01</td>
<td>0.9</td>
<td>0.95</td>
</tr>
<tr>
<td>0.1 - 0.7</td>
<td>0.97</td>
<td>0.86</td>
<td>0.9</td>
</tr>
<tr>
<td>0.1 - 0.8</td>
<td>0.94</td>
<td>0.85</td>
<td>0.89</td>
</tr>
<tr>
<td>0.2 - 0.5</td>
<td>0.93</td>
<td>0.87</td>
<td>0.98</td>
</tr>
<tr>
<td>0.2 - 0.6</td>
<td>0.91</td>
<td>0.84</td>
<td>0.96</td>
</tr>
<tr>
<td>0.2 - 0.7</td>
<td>0.88</td>
<td>0.72</td>
<td>0.89</td>
</tr>
<tr>
<td>0.2 - 0.8</td>
<td>0.82</td>
<td>0.71</td>
<td>0.81</td>
</tr>
<tr>
<td>0.3 - 0.5</td>
<td>1.05</td>
<td>0.99</td>
<td>1.11</td>
</tr>
<tr>
<td>0.3 - 0.6</td>
<td>1.11</td>
<td>1.05</td>
<td>1.09</td>
</tr>
<tr>
<td>0.3 - 0.7</td>
<td>1.1</td>
<td>1.03</td>
<td>1.1</td>
</tr>
<tr>
<td>0.3 - 0.8</td>
<td>1.03</td>
<td>0.92</td>
<td>1.07</td>
</tr>
<tr>
<td>0.4 - 0.5</td>
<td>1.3</td>
<td>1.27</td>
<td>1.46</td>
</tr>
<tr>
<td>0.4 - 0.6</td>
<td>1.31</td>
<td>1.22</td>
<td>1.41</td>
</tr>
<tr>
<td>0.4 - 0.7</td>
<td>1.26</td>
<td>1.15</td>
<td>1.26</td>
</tr>
<tr>
<td>0.4 - 0.8</td>
<td>1.07</td>
<td>1.03</td>
<td>1.05</td>
</tr>
</tbody>
</table>
For example the overall speedup of \textit{DGdGAT1} is equal to 1.2 with diversity levels of \(d_{\text{min}} = 0.2\) \& \(d_{\text{max}} = 0.8\), and the overall speedup of \textit{DGdGAT2} is equal to 1.365 with diversity levels \(d_{\text{min}} = 0.2\) \& \(d_{\text{max}} = 0.7, 0.8\), while the overall speedup of \textit{DGdGAT3} is equal to 1.22 with diversity levels of \(d_{\text{min}} = 0.2\) \& \(d_{\text{max}} = 0.8\). As we can see, the "T2" model maintains superior performance over the other two models. When we consider the serial fraction values in Table 4-5, we can observe that, with all serial fraction values >0, we do not achieve super-linear speedup, however, the best performances come close to linear speedup.

This can be seen with the diversity levels of \(d_{\text{min}} = 0.2\) \& \(d_{\text{max}} = 0.7, 0.8\). For example, the mean serial fraction value is 0.82 for \textit{DGdGAT1} and 0.81 for \textit{DGdGAT3} while it is 0.71 for \textit{DGdGAT2} with diversity levels \(d_{\text{min}} = 0.2\) \& \(d_{\text{max}} = 0.8\). In Figure 5-2 and Figure 5-3 respectively, we show the speedup values and the serial fraction values graphically, enabling us to easily assess the relative performance of the three diversity-guided models.

In fact, when comparing the diversity guided models with the adaptive models (Chapter IV) we can notice that their parallel performances are very close to each other. By looking on Table 5-5 which shows the mean speedup values of the adaptive parallel models \textit{AdGAT1}, \textit{AdGAT2} and \textit{AdGAT3}, we can see that the maximum achieved speedup is 0.86 for \textit{AdGAT1}, 1.365 for \textit{AdGAT2} and 0.99 for \textit{AdGAT3}.

### Table 5-5: The mean Speedup values of the adapted models \textit{AdGAT1}, \textit{AdGAT2} and \textit{AdGAT3} of the 30D benchmark problems with adaptive scheme 100,300.

<table>
<thead>
<tr>
<th>Adaptive Scheme</th>
<th>\textit{AdGAT1}</th>
<th>\textit{AdGAT2}</th>
<th>\textit{AdGAT3}</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Normal Shifted Mean</td>
<td>Normal Shifted Mean</td>
<td>Normal Shifted Mean</td>
</tr>
<tr>
<td>100</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.77</td>
<td>0.95</td>
<td>0.86</td>
<td>1.07</td>
</tr>
<tr>
<td>300</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.61</td>
<td>0.91</td>
<td>0.76</td>
<td>1.04</td>
</tr>
</tbody>
</table>

The highest speedup values for the diversity guided parallel models are 1.2 for \textit{DGdGAT1}, 1.365 for \textit{DGdGAT2} and 1.22 for \textit{DGdGAT3} with diversity levels of \(d_{\text{min}} = 0.2\) \& \(d_{\text{max}} = 0.8\). Table 5-6 shows the mean serial fraction values of the
adaptive models along with shifted and not shifted versions of the 30D benchmark problems.

Table 5-6: The mean Serial Fraction values of the adapted models AdGAT1, AdGAT2 and AdGAT3 of the 30D benchmark problems with adaptive scheme 100,300.

<table>
<thead>
<tr>
<th>Adaptive Scheme</th>
<th>AdGAT1 Normal</th>
<th>AdGAT1 Shifted</th>
<th>AdGAT1 Mean</th>
<th>AdGAT2 Normal</th>
<th>AdGAT2 Shifted</th>
<th>AdGAT2 Mean</th>
<th>AdGAT3 Normal</th>
<th>AdGAT3 Shifted</th>
<th>AdGAT3 Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>1.32</td>
<td>1.06</td>
<td>1.17</td>
<td>0.93</td>
<td>0.58</td>
<td>0.715</td>
<td>0.99</td>
<td>1.02</td>
<td>1.005</td>
</tr>
<tr>
<td>300</td>
<td>1.68</td>
<td>1.11</td>
<td>1.337</td>
<td>0.96</td>
<td>0.65</td>
<td>0.78</td>
<td>1.23</td>
<td>0.99</td>
<td>1.099</td>
</tr>
</tbody>
</table>

From Table 5-5 and Table 5-6 we can tentatively conclude that the parallel performances of the diversity guided models are a slightly better than the adaptive models with the 30D benchmark test problems. The Figure 5-4 and Figure 5-5 respectively illustrate the comparison between the mean speedup and serial fraction values of the Adaptive (100 & 300) and Diversity Guided ((0.2, 0.7), (0.2, 0.8)) models of the 30D benchmark test problems.

Figure 5-2: The DGdGAT1, DGdGAT2 and DGdGAT3 Speedup values of the 30D benchmark test problems.
Overall, the results of the new diversity-guided models on the 30D benchmark functions - if we choose the appropriate range of diversity threshold - show a modest improvement over the non-adaptive parallel models of Chapter III, and similar levels of performance to the performance-based adaptive models in Chapter IV. We next turn to our experiments on 50D and 100D versions of the test functions, to see if these more challenging settings reveal clearer distinctions in relative performance.
Figure 5-4: Comparison between the mean Speedup values of the Adaptive (100 & 300) and Diversity Guided ($d_{min} = 0.2$ & $d_{max} = 0.7, 0.8$) models of the 30D benchmark test problems.

Figure 5-5: Comparison between the mean Serial Fraction values of the Adaptive (100 & 300) and the Diversity Guided ($d_{min} = 0.2$ & $d_{max} = 0.7, 0.8$) models of the 30D benchmark test problems.
5.4.3.2 The 50-Dimensional Test Problems

In this section we repeat the previous experiments testing the performance of \textit{DGdGAT1}, \textit{DGdGAT2} and \textit{DGdGAT3}, but this time we explore the 50-Dimension versions of the test problems.

Table 5-7 shows the success rates for the diversity-guided models on the 50D test functions, where each figure in the 'Total' columns is out of a maximum of 220. In this table and the remainder of this chapter, we omit the results for \textit{fS-Ros}, since none of the algorithms was ever able to solve this in the maximum time allowed - it therefore simplifies comparison if we consider only the 11 remaining benchmarks, and consequently a maximum number of 220 successful trials.

From Table 5-7, we can observe the expected reduction in success rates as we move to the 50D benchmarks. However, in this context, we can note two interesting points. First, the parallel diversity guided model \textit{DGdGAT2} still has the superior performance than the other models. Second, the diversity thresholds that led to best performance on the 30D functions are the same as those that led to best performance on the 50D functions $d_{\text{min}} = 0.2$ \& $d_{\text{max}} = 0.7, 0.8$. However, overall we can see that the relative performances of the three models have not been significantly changed in terms of success rates.

The best success rates for all models seem to be just above 96% with diversity level $d_{\text{min}} = 0.2$ \& $d_{\text{max}} = 0.8$, which is almost the same for the 30D benchmarks (see Table 5-2). The worst results appear in the diversity range $d_{\text{min}} = 0.4$ \& $d_{\text{max}} = 0.5$. For example, with the diversity threshold $d_{\text{min}} = 0.2$ \& $d_{\text{max}} = 0.8$ the success rate is 96.36% for \textit{DGdGAT1}, 97.73% for \textit{DGdGAT2}, and 96.82% for \textit{DGdGAT3}. While with the diversity threshold $d_{\text{min}} = 0.4$ \& $d_{\text{max}} = 0.5$ the success rate is 63.18% for \textit{DGdGAT1}, 69.55% for \textit{DGdGAT2}, and 65.91% for \textit{DGdGAT3}. Figure 5-6 shows how the performance of the diversity guided models change when the diversity levels change. We can see the success rate of the three models \textit{DGdGAT1}, \textit{DGdGAT2} and \textit{DGdGAT3} touch the highest peak with diversity threshold $d_{\text{min}} = 0.2, 0.3 \& d_{\text{max}} = 0.6, 0.7, 0.8$, while they drop to the lowest peak with diversity levels $d_{\text{min}} = 0.3, 0.4 \& d_{\text{max}} = 0.5, 0.6$. 

121
Table 5-7: The AdGAT1, AdGAT2 and AdGAT3 success rates out of 220 runs to optimize 50D benchmark functions using diversity threshold $d_{\text{min}} \in [0.1, 0.4]$ & $d_{\text{max}} \in [0.5, 0.8]$.

<table>
<thead>
<tr>
<th>Diversity Threshold</th>
<th>DGdGAT1</th>
<th></th>
<th>DGdGAT2</th>
<th></th>
<th>DGdGAT3</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Total</td>
<td>Mean %</td>
<td>Total</td>
<td>Mean %</td>
<td>Total</td>
<td>Mean %</td>
</tr>
<tr>
<td>min - max</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.1 - 0.5</td>
<td>168</td>
<td>76.36</td>
<td>164</td>
<td>74.55</td>
<td>162</td>
<td>73.64</td>
</tr>
<tr>
<td>0.1 - 0.6</td>
<td>180</td>
<td>81.82</td>
<td>185</td>
<td>84.09</td>
<td>180</td>
<td>81.82</td>
</tr>
<tr>
<td>0.1 - 0.7</td>
<td>186</td>
<td>84.55</td>
<td>191</td>
<td>86.82</td>
<td>183</td>
<td>83.18</td>
</tr>
<tr>
<td>0.1 - 0.8</td>
<td>187</td>
<td>85</td>
<td>199</td>
<td>90.45</td>
<td>187</td>
<td>85</td>
</tr>
<tr>
<td>0.2 - 0.5</td>
<td>177</td>
<td>80.45</td>
<td>182</td>
<td>82.73</td>
<td>179</td>
<td>81.36</td>
</tr>
<tr>
<td>0.2 - 0.6</td>
<td>201</td>
<td>91.36</td>
<td>205</td>
<td>93.18</td>
<td>202</td>
<td>91.82</td>
</tr>
<tr>
<td>0.2 - 0.7</td>
<td><strong>211</strong></td>
<td><strong>95.91</strong></td>
<td><strong>215</strong></td>
<td><strong>97.73</strong></td>
<td><strong>212</strong></td>
<td><strong>96.36</strong></td>
</tr>
<tr>
<td>0.2 - 0.8</td>
<td><strong>212</strong></td>
<td><strong>96.36</strong></td>
<td><strong>215</strong></td>
<td><strong>97.73</strong></td>
<td><strong>213</strong></td>
<td><strong>96.82</strong></td>
</tr>
<tr>
<td>0.3 - 0.5</td>
<td>173</td>
<td>78.64</td>
<td>177</td>
<td>80.45</td>
<td>175</td>
<td>79.55</td>
</tr>
<tr>
<td>0.3 - 0.6</td>
<td>199</td>
<td>90.45</td>
<td>201</td>
<td>91.36</td>
<td>199</td>
<td>90.45</td>
</tr>
<tr>
<td>0.3 - 0.7</td>
<td>202</td>
<td>91.82</td>
<td>208</td>
<td>94.55</td>
<td>204</td>
<td>92.73</td>
</tr>
<tr>
<td>0.3 - 0.8</td>
<td>207</td>
<td>94.09</td>
<td>209</td>
<td>95</td>
<td>204</td>
<td>92.73</td>
</tr>
<tr>
<td>0.4 - 0.5</td>
<td>139</td>
<td>63.18</td>
<td>153</td>
<td>69.55</td>
<td>145</td>
<td>65.91</td>
</tr>
<tr>
<td>0.4 - 0.6</td>
<td>174</td>
<td>79.09</td>
<td>176</td>
<td>80</td>
<td>177</td>
<td>80.45</td>
</tr>
<tr>
<td>0.4 - 0.7</td>
<td>177</td>
<td>80.45</td>
<td>181</td>
<td>82.27</td>
<td>173</td>
<td>78.64</td>
</tr>
<tr>
<td>0.4 - 0.8</td>
<td>179</td>
<td>81.36</td>
<td>186</td>
<td>84.55</td>
<td>180</td>
<td>81.82</td>
</tr>
</tbody>
</table>
We now look at the speedup and serial fraction metrics for the 50D tests, in attempt to glean more useful information about the performance of the diversity-guided models. Table 5-8 shows the speedup of the diversity guided models (DGdGAT1, DGdGAT2 and DGdGAT3) over their corresponding non-adaptive distributed models (dGAT1, dGAT2 and dGAT3), and Table 5-9 shows the serial fraction values of the diversity-guided models.

Table 5-8 reveals that the diversity-guided models exhibit good performance in comparison with the non-adaptive models. In the best cases, we find over 2.5-fold speedup when optimizing the 50D benchmark functions. Further inspection of Table 5-8 shows that the diversity guided models achieved speedup values >1 for most diversity thresholds, while DGdGAT2 still exhibits superior performance to the other models. In addition, it seems that the diversity thresholds [0.2, 0.7], [0.2, 0.8] and [0.3, 0.8] clearly allow the diversity-guided models to achieve more effective control of migration for each model; all of the experiments using these diversity levels achieved more than twofold speedup.
Table 5-8: The Speedup values of the diversity guided models The $DGdGAT_1$, $DGdGAT_2$ and $DGdGAT_3$ against the fixed models $dGAT_1$, $dGAT_2$ and $dGAT_3$ of 50D benchmark problems.

<table>
<thead>
<tr>
<th>Diversity Threshold min - max</th>
<th>$DGdGAT_1$</th>
<th>$DGdGAT_2$</th>
<th>$DGdGAT_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1 - 0.5</td>
<td>0.70541</td>
<td>1.11328</td>
<td>0.50512</td>
</tr>
<tr>
<td>0.1 - 0.6</td>
<td>0.85592</td>
<td>1.39304</td>
<td>0.69092</td>
</tr>
<tr>
<td>0.1 - 0.7</td>
<td>1.42069</td>
<td>1.83841</td>
<td>1.17339</td>
</tr>
<tr>
<td>0.1 - 0.8</td>
<td>0.86529</td>
<td>1.33349</td>
<td>0.65936</td>
</tr>
<tr>
<td>0.2 - 0.5</td>
<td>1.20529</td>
<td>1.58305</td>
<td>1.02957</td>
</tr>
<tr>
<td>0.2 - 0.6</td>
<td>1.05513</td>
<td>1.72756</td>
<td>1.05777</td>
</tr>
<tr>
<td>0.2 - 0.7</td>
<td>2.04988</td>
<td>2.39835</td>
<td>2.14908</td>
</tr>
<tr>
<td>0.2 - 0.8</td>
<td>2.31153</td>
<td>2.77334</td>
<td>2.49589</td>
</tr>
<tr>
<td>0.3 - 0.5</td>
<td>0.76946</td>
<td>1.16681</td>
<td>0.9676</td>
</tr>
<tr>
<td>0.3 - 0.6</td>
<td>0.55248</td>
<td>1.14136</td>
<td>0.76016</td>
</tr>
<tr>
<td>0.3 - 0.7</td>
<td>0.79658</td>
<td>1.62815</td>
<td>1.08494</td>
</tr>
<tr>
<td>0.3 - 0.8</td>
<td>2.03457</td>
<td>2.21856</td>
<td>2.06261</td>
</tr>
<tr>
<td>0.4 - 0.5</td>
<td>0.56072</td>
<td>1.08235</td>
<td>0.66024</td>
</tr>
<tr>
<td>0.4 - 0.6</td>
<td>1.01109</td>
<td>1.38</td>
<td>1.1436</td>
</tr>
<tr>
<td>0.4 - 0.7</td>
<td>1.15092</td>
<td>1.61916</td>
<td>1.30751</td>
</tr>
<tr>
<td>0.4 - 0.8</td>
<td>1.2963</td>
<td>1.7216</td>
<td>1.50365</td>
</tr>
</tbody>
</table>

However, the diversity level of $d_{min} = 0.2$ & $d_{max} = 0.8$ seems to be the best among the other diversity levels in controlling the studied parallel models to optimize the 50D benchmark fitness functions. For example, the speedup values with the diversity level of $d_{min} = 0.2$ & $d_{max} = 0.8$ are 2.31 for $DGdGAT_1$, 2.77 for $DGdGAT_2$ and 2.49 for $DGdGAT_3$. We can also observe that the serial fraction values in Table 5-9 support the findings about speedup values, in the sense that we can see there is no indication of super-linear speedup.

However, the best performance comes close to near linear speedup with serial fraction values <0.5. This can be seen with the diversity levels of $(d_{min} =$
0.2 & \( d_{\text{max}} = 0.7, 0.8 \) for the three models, and \( d_{\text{min}} = 0.3 \) & \( d_{\text{max}} = 0.8 \) for \( DGdGAT2 \) only. As an example, with diversity levels \( d_{\text{min}} = 0.2 \) & \( d_{\text{max}} = 0.8 \) the mean serial fraction value is 0.395 for \( DGdGAT1 \) and 0.361 for \( DGdGAT3 \) while it is 0.318 for \( DGdGAT2 \). In Figure 5-7 and Figure 5-8 we provide graphics that help illustrate the relative performances of the three diversity guided models on the 50D functions, in terms of speedup and serial fraction respectively.

<table>
<thead>
<tr>
<th>Diversity Threshold min - max</th>
<th>( DGdGAT1 )</th>
<th>( DGdGAT2 )</th>
<th>( DGdGAT3 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1 - 0.5</td>
<td>1.44546</td>
<td>0.89146</td>
<td>2.04504</td>
</tr>
<tr>
<td>0.1 - 0.6</td>
<td>1.17956</td>
<td>0.69904</td>
<td>1.47717</td>
</tr>
<tr>
<td>0.1 - 0.7</td>
<td>0.68414</td>
<td>0.51354</td>
<td>0.84238</td>
</tr>
<tr>
<td>0.1 - 0.8</td>
<td>1.16606</td>
<td>0.73324</td>
<td>1.55106</td>
</tr>
<tr>
<td>0.2 - 0.5</td>
<td>0.81832</td>
<td>0.60714</td>
<td>0.96936</td>
</tr>
<tr>
<td>0.2 - 0.6</td>
<td>0.94427</td>
<td>0.55077</td>
<td>0.94174</td>
</tr>
<tr>
<td>0.2 - 0.7</td>
<td>\textbf{0.45369}</td>
<td>\textbf{0.37808}</td>
<td>\textbf{0.42967}</td>
</tr>
<tr>
<td>0.2 - 0.8</td>
<td>\textbf{0.39479}</td>
<td>\textbf{0.31795}</td>
<td>\textbf{0.3607}</td>
</tr>
<tr>
<td>0.3 - 0.5</td>
<td>1.31959</td>
<td>0.84751</td>
<td>1.03572</td>
</tr>
<tr>
<td>0.3 - 0.6</td>
<td>1.86402</td>
<td>0.86789</td>
<td>1.33655</td>
</tr>
<tr>
<td>0.3 - 0.7</td>
<td>1.27239</td>
<td>0.58847</td>
<td>0.91649</td>
</tr>
<tr>
<td>0.3 - 0.8</td>
<td>0.64228</td>
<td>\textbf{0.43682}</td>
<td>0.5385</td>
</tr>
<tr>
<td>0.4 - 0.5</td>
<td>1.83565</td>
<td>0.91884</td>
<td>1.54891</td>
</tr>
<tr>
<td>0.4 - 0.6</td>
<td>0.9883</td>
<td>0.70628</td>
<td>0.86606</td>
</tr>
<tr>
<td>0.4 - 0.7</td>
<td>0.86013</td>
<td>0.59211</td>
<td>0.74913</td>
</tr>
<tr>
<td>0.4 - 0.8</td>
<td>0.75619</td>
<td>0.55291</td>
<td>0.64272</td>
</tr>
</tbody>
</table>
Table 5-10: The mean Speedup values of the adapted models AdGAT1, AdGAT2 and AdGAT3 of the 50D benchmark problems with adaptive scheme 100,300.

<table>
<thead>
<tr>
<th>Adaptive Scheme</th>
<th>AdGAT1</th>
<th></th>
<th></th>
<th>AdGAT2</th>
<th></th>
<th></th>
<th>AdGAT3</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Normal</td>
<td>Shifted</td>
<td>Mean</td>
<td>Normal</td>
<td>Shifted</td>
<td>Mean</td>
<td>Normal</td>
<td>Shifted</td>
<td>Mean</td>
<td>Mean</td>
</tr>
<tr>
<td>100</td>
<td>1.55</td>
<td>1.38</td>
<td>1.465</td>
<td>2.77</td>
<td>2.45</td>
<td>2.61</td>
<td>1.64</td>
<td>1.16</td>
<td>1.4</td>
</tr>
<tr>
<td>300</td>
<td>1.50</td>
<td>1.37</td>
<td>1.435</td>
<td>2.55</td>
<td>2.20</td>
<td>2.375</td>
<td>1.54</td>
<td>1.14</td>
<td>1.34</td>
</tr>
</tbody>
</table>

Table 5-11: The mean Serial Fraction values of the adapted models AdGAT1, AdGAT2 and AdGAT3 of the 50D benchmark problems with adaptive scheme 100,300.

<table>
<thead>
<tr>
<th>Adaptive Scheme</th>
<th>AdGAT1</th>
<th></th>
<th></th>
<th>AdGAT2</th>
<th></th>
<th></th>
<th>AdGAT3</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Normal</td>
<td>Shifted</td>
<td>Mean</td>
<td>Normal</td>
<td>Shifted</td>
<td>Mean</td>
<td>Normal</td>
<td>Shifted</td>
<td>Mean</td>
<td>Mean</td>
</tr>
<tr>
<td>100</td>
<td>0.62</td>
<td>0.71</td>
<td>0.661</td>
<td>0.32</td>
<td>0.35</td>
<td>0.342</td>
<td>0.59</td>
<td>0.85</td>
<td>0.695</td>
</tr>
<tr>
<td>300</td>
<td>0.64</td>
<td>0.71</td>
<td>0.677</td>
<td>0.37</td>
<td>0.42</td>
<td>0.382</td>
<td>0.62</td>
<td>0.87</td>
<td>0.729</td>
</tr>
</tbody>
</table>

Turning now to comparing the diversity-guided models of this chapter with the adaptive models of Chapter IV, in the context of the 50D functions, we can observe that the diversity-guided models seem slightly better. We can see this in Table 5-10, which shows the mean speedup values for the Chapter IV models, the maximum achievements for each model being 1.465 for AdGAT1, 2.61 for AdGAT2 and 1.4 for AdGAT3. These compare with speedups (over the non-adaptive Chapter III models) for the diversity-guided approaches (and with the best choice of diversity threshold parameters) of 2.31 for DGdGAT1, 2.77 for DGdGAT2 and 2.5 for DGdGAT3. Meanwhile, Table 5-11 shows the mean serial fraction values of the Chapter IV models, for both the non-shifted and shifted versions of the 50D benchmark functions.

From Table 5-10 and Table 5-11 we can observe that diversity-guided models can achieve between 50% and 73% better speedup than the adaptive models from Chapter IV, on the 50D benchmark test problems. Figure 5-9 and Figure 5-10 respectively illustrate the comparison between the mean speedup and serial fraction values of the diversity-guided models (DG - for the two best diversity threshold intervals), with the
Chapter IV models (AS - for the two explored adaptive window reset intervals 100 and 300).

Figure 5-7: The $DGdGAT1$, $DGdGAT2$ and $DGdGAT3$ Speedup values of the 50D benchmark test problems.

Figure 5-8: The $DGdGAT1$, $DGdGAT2$ and $DGdGAT3$ Serial Fraction values of the 50D benchmark test problems.
We now present the results of statistical analyses of some of the experiments in this chapter, in order to gain an understanding of the significance of the observations. We perform paired 1-tailed T-tests on the mean execution time figures in the various
experiments, and focus on the 50D benchmark functions. For clarity and simplicity, we restrict our statistical analyses to the diversity-guided cases that used the diversity threshold interval \((d_{\text{min}}, d_{\text{max}}) = (0.2, 0.8)\). The statistical tests have been done using a cut-off significance level of \((0.05)\), and assuming unequal variance.

Table 5-12 summarises the statistical test results for the 50D functions, comparing the three \(DGdGA\) models against each other, and also segmenting the findings according to either shifted or non-shifted versions of the benchmark functions.

From the results in Table 5-12 it is clear that the model based on topology T2, (i.e. model \(DGdGAT2\)), again shows superiority over the other models for all benchmarks. The T-Test p-values for \(DGdGAT1-DGdGAT2\) for all fitness functions are in the range \([0.001, 0.02387]\), while the T-Test p-values for \(DGdGAT3-DGdGAT2\) for all problems are in the range \([0.0011, 0.0097]\). Closer inspection shows that the confidence in \(DGdGAT2\)'s superiority over \(DGdGAT1\) is generally greater for the non-shifted functions than it is for the shifted functions, while no such obvious distinction arises when we consider \(DGdGAT2\)'s superiority over \(DGdGAT3\).

Table 5-12: The 50D benchmarks paired 1-tailed T-Test p-values for the comparison between the execution times of diversity guided models \(DGdGAT1, DGdGAT2\) and \(DGdGAT3\) using the diversity threshold of \((d_{\text{min}}, d_{\text{max}}) = (0.2, 0.8)\)

<table>
<thead>
<tr>
<th>Method</th>
<th>(fS)-(Sph)</th>
<th>(fS)-(Ros)</th>
<th>(fS)-(Ras)</th>
<th>(fS)-(Sch)</th>
<th>(fS)-(Gri)</th>
<th>(fS)-(Ack)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(DGdGAT1-DGdGAT2)</td>
<td>0.023871</td>
<td>-</td>
<td>0.002511</td>
<td>0.008573</td>
<td>0.003372</td>
<td>0.011449</td>
</tr>
<tr>
<td>(DGdGAT3-DGdGAT2)</td>
<td>0.003446</td>
<td>-</td>
<td>0.002678</td>
<td>0.006414</td>
<td>0.003177</td>
<td>0.006847</td>
</tr>
<tr>
<td>(DGdGAT1-DGdGAT3)</td>
<td>0.019918</td>
<td>-</td>
<td>0.028945</td>
<td>0.023459</td>
<td>0.038613</td>
<td>0.029046</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method</th>
<th>(fS)-(Sph)</th>
<th>(fS)-(Ros)</th>
<th>(fS)-(Ras)</th>
<th>(fS)-(Sch)</th>
<th>(fS)-(Gri)</th>
<th>(fS)-(Ack)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(DGdGAT1-DGdGAT2)</td>
<td>0.023871</td>
<td>-</td>
<td>0.002511</td>
<td>0.008573</td>
<td>0.003372</td>
<td>0.011449</td>
</tr>
<tr>
<td>(DGdGAT3-DGdGAT2)</td>
<td>0.003446</td>
<td>-</td>
<td>0.002678</td>
<td>0.006414</td>
<td>0.003177</td>
<td>0.006847</td>
</tr>
<tr>
<td>(DGdGAT1-DGdGAT3)</td>
<td>0.019918</td>
<td>-</td>
<td>0.028945</td>
<td>0.023459</td>
<td>0.038613</td>
<td>0.029046</td>
</tr>
</tbody>
</table>
Overall, the results on the 50D functions start to suggest that the diversity-guided approach to adaptation might have an advantage over the performance-based adaptation approach in Chapter IV, especially as problem dimensionality increases. We next continue this investigation with the 100D versions of the benchmark problems.

5.4.3.3 The 100-Dimensional Test Problems

This section presents the results of our experiments with the diversity-guided models on 100-Dimensional versions of the benchmark test problems. In common with all other sets of experiments in Chapter IV and in this chapter so far, we compare three versions of the diversity-guided approach, each of which relates to one of the basic topologies introduced in Chapter III, and each involving 16 client processors and one master processor. First, Table 5-13 summarises the results in terms of the success rates of \textit{DGdGAT1}, \textit{DGdGAT2} and \textit{DGdGAT3} on the 100D benchmark functions.

Again, as we increase problem dimensionality, and consequently make the problem landscape very significantly more difficult to search efficiently, we see a drop in success rates when compared with the 50D results. For example, for certain diversity threshold parameters, we can now see success rates dropping towards the 50% - 60% region. However, for good choices of diversity threshold parameters, the success rates have only slightly reduced. In fact, \textit{DGdGAT2} seems to be the least affected, in terms of success rates, by this increase in problem dimensionality. But it can also be said that, given appropriately good choices of the diversity threshold parameters, the diversity-guided models achieve success rates that are only marginally affected by this vast increase in problem dimensionality. Clearly, however, higher problem dimensionality seems to make the diversity-guided approach more sensitive to the threshold parameter settings.

However, in this context, we can consider two issues. First, the parallel diversity guided model \textit{DGdGAT2} still shows superior performance to than the other models. Second, the three diversity guided models show their best performance with the diversity thresholds $d_{\text{min}} = 0.2$ & $d_{\text{max}} = 0.7$, 0.8. The best success rates for all models are >95% with diversity level $d_{\text{min}} = 0.2$ & $d_{\text{max}} = 0.8$, which is similar to, but slightly lower than, the diversity threshold intervals that performed best for the 30D
and 50D benchmarks. Meanwhile, the worst results appear in the diversity range \( d_{\text{min}} = 0.4 \) & \( d_{\text{max}} = 0.5 \). For example, with the diversity threshold \( d_{\text{min}} = 0.2 \) & \( d_{\text{max}} = 0.8 \) the success rate is 95.91\% for \( DGdGAT_1 \), 96.82\% for \( DGdGAT_2 \), and 96.36\% for \( DGdGAT_3 \). While with the diversity threshold \( d_{\text{min}} = 0.4 \) & \( d_{\text{max}} = 0.5 \) the success rate is 50.91\% for \( DGdGAT_1 \), 59.09\% for \( DGdGAT_2 \), and 52.73\% for \( DGdGAT_3 \). Figure 5-11 shows the performance of the diversity guided models with the 100D test functions.

### Table 5-13: The \( DGdGAT_1, DGdGAT_2 \) and \( DGdGAT_3 \) mean success rates out of 220 to optimize the 100D benchmark functions with diversity threshold \( d_{\text{min}} \in [0.1, 0.4] \) & \( d_{\text{max}} \in [0.5, 0.8] \).

<table>
<thead>
<tr>
<th>Diversity Threshold</th>
<th>Method</th>
<th>( DGdGAT_1 )</th>
<th>Total</th>
<th>Mean %</th>
<th>( DGdGAT_2 )</th>
<th>Total</th>
<th>Mean %</th>
<th>( DGdGAT_3 )</th>
<th>Total</th>
<th>Mean %</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Total</td>
<td>Mean %</td>
<td></td>
<td>Total</td>
<td>Mean %</td>
<td>Total</td>
<td>Mean %</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( 0.1 - 0.5 )</td>
<td></td>
<td>172</td>
<td>78.18</td>
<td>180</td>
<td>81.82</td>
<td>171</td>
<td>77.73</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( 0.1 - 0.6 )</td>
<td></td>
<td>174</td>
<td>79.09</td>
<td>181</td>
<td>82.27</td>
<td>175</td>
<td>79.55</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( 0.1 - 0.7 )</td>
<td></td>
<td>176</td>
<td>80.00</td>
<td>182</td>
<td>82.73</td>
<td>178</td>
<td>80.91</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( 0.1 - 0.8 )</td>
<td></td>
<td>177</td>
<td>80.45</td>
<td>183</td>
<td>83.18</td>
<td>177</td>
<td>80.45</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( 0.2 - 0.5 )</td>
<td></td>
<td>197</td>
<td>89.55</td>
<td>199</td>
<td>90.45</td>
<td>197</td>
<td>89.55</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( 0.2 - 0.6 )</td>
<td></td>
<td>199</td>
<td>90.45</td>
<td>204</td>
<td>92.73</td>
<td>200</td>
<td>90.91</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( 0.2 - 0.7 )</td>
<td></td>
<td>210</td>
<td>95.45</td>
<td>212</td>
<td>96.36</td>
<td>211</td>
<td>95.91</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( 0.2 - 0.8 )</td>
<td></td>
<td>211</td>
<td>95.91</td>
<td>213</td>
<td>96.82</td>
<td>212</td>
<td>96.36</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( 0.3 - 0.5 )</td>
<td></td>
<td>185</td>
<td>84.09</td>
<td>188</td>
<td>85.45</td>
<td>181</td>
<td>82.27</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( 0.3 - 0.6 )</td>
<td></td>
<td>184</td>
<td>83.64</td>
<td>190</td>
<td>86.36</td>
<td>186</td>
<td>84.55</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( 0.3 - 0.7 )</td>
<td></td>
<td>188</td>
<td>85.45</td>
<td>192</td>
<td>87.27</td>
<td>189</td>
<td>85.91</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( 0.3 - 0.8 )</td>
<td></td>
<td>188</td>
<td>85.45</td>
<td>193</td>
<td>87.73</td>
<td>187</td>
<td>85</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( 0.4 - 0.5 )</td>
<td></td>
<td>112</td>
<td>50.91</td>
<td>130</td>
<td>59.09</td>
<td>116</td>
<td>52.73</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( 0.4 - 0.6 )</td>
<td></td>
<td>155</td>
<td>70.45</td>
<td>165</td>
<td>75.00</td>
<td>151</td>
<td>68.64</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( 0.4 - 0.7 )</td>
<td></td>
<td>169</td>
<td>76.82</td>
<td>172</td>
<td>78.18</td>
<td>170</td>
<td>77.27</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( 0.4 - 0.8 )</td>
<td></td>
<td>168</td>
<td>76.36</td>
<td>176</td>
<td>80.00</td>
<td>170</td>
<td>77.27</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

131
As before, we now look at other metrics to help assess the performance of the diversity guided models in comparison to each other, and in comparison to other models studied in this thesis. Table 5-14 shows the mean speedup of the diversity guided models (DGdGAT1, DGdGAT2 and DGdGAT3) using the diversity level of \( d_{\text{min}} \in [0.1, 0.4] \) & \( d_{\text{max}} \in [0.5, 0.8] \), over the fixed models (dGAT1, dGAT2 and dGAT3). While the Table 5-15 shows the serial fraction values.

From Table 5-14 we can clearly see that with an appropriate choice for the diversity threshold interval, the diversity-guided models show very good speedup over the non-adaptive distributed models of Chapter III. However, we can again see that, in the context of these higher dimensional problems, performance becomes highly sensitive to the threshold parameters. More than doubled speedup can be achieved by each model, but with poor choices for the diversity threshold interval the speedup reduces to around 0.14.

Nevertheless, actual speedup over the non-adaptive distributed model (with speedup value appreciably larger than 1) can be observed for a wide range of diversity
threshold values. Meanwhile we can clearly see that the diversity threshold interval [0.2, 0.8] is the most successful across all three models. These and related observations are visually apparent in Figure 5-7.

Table 5-14: The Speedup values of the diversity guided models The \textit{DGdGAT1}, \textit{DGdGAT2} and \textit{DGdGAT3} against the fixed models \textit{dGAT1}, \textit{dGAT2} and \textit{dGAT3} of 100D benchmark problems.

<table>
<thead>
<tr>
<th>Diversity Threshold min - max</th>
<th>\textit{DGdGAT1}</th>
<th>\textit{DGdGAT2}</th>
<th>\textit{DGdGAT3}</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1 - 0.5</td>
<td>0.311</td>
<td>0.467</td>
<td>0.288</td>
</tr>
<tr>
<td>0.1 - 0.6</td>
<td>0.326</td>
<td>0.529</td>
<td>0.32</td>
</tr>
<tr>
<td>0.1 - 0.7</td>
<td>0.356</td>
<td>0.565</td>
<td>0.347</td>
</tr>
<tr>
<td>0.1 - 0.8</td>
<td>0.429</td>
<td>0.673</td>
<td>0.367</td>
</tr>
<tr>
<td>0.2 - 0.5</td>
<td>1.37</td>
<td>1.51</td>
<td>1.435</td>
</tr>
<tr>
<td>0.2 - 0.6</td>
<td>1.501</td>
<td>1.909</td>
<td>1.571</td>
</tr>
<tr>
<td>0.2 - 0.7</td>
<td><strong>1.884</strong></td>
<td><strong>2.157</strong></td>
<td><strong>1.891</strong></td>
</tr>
<tr>
<td>0.2 - 0.8</td>
<td><strong>2.013</strong></td>
<td><strong>2.426</strong></td>
<td><strong>2.015</strong></td>
</tr>
<tr>
<td>0.3 - 0.5</td>
<td>0.41</td>
<td>0.71</td>
<td>0.608</td>
</tr>
<tr>
<td>0.3 - 0.6</td>
<td>0.55</td>
<td>0.81</td>
<td>0.653</td>
</tr>
<tr>
<td>0.3 - 0.7</td>
<td>0.65</td>
<td>0.91</td>
<td>0.753</td>
</tr>
<tr>
<td>0.3 - 0.8</td>
<td>0.753</td>
<td>1.024</td>
<td>0.87</td>
</tr>
<tr>
<td>0.4 - 0.5</td>
<td>0.14</td>
<td>0.167</td>
<td>0.134</td>
</tr>
<tr>
<td>0.4 - 0.6</td>
<td>0.149</td>
<td>0.216</td>
<td>0.205</td>
</tr>
<tr>
<td>0.4 - 0.7</td>
<td>0.239</td>
<td>0.379</td>
<td>0.206</td>
</tr>
<tr>
<td>0.4 - 0.8</td>
<td>0.306</td>
<td>0.425</td>
<td>0.219</td>
</tr>
</tbody>
</table>

The serial fraction values that we can observe in Table 5-15 are in line with our observations about speedup. Though we cannot see super-linear speedup (serial fraction below 0), however when the diversity threshold interval is appropriately well chosen we can see serial fraction values below 0.5, suggestion near linear speedup. As an example...
of that, using the diversity levels \( d_{\text{min}} = 0.2 \) & \( d_{\text{max}} = 0.8 \) the mean serial fraction value is 0.463 for \( \text{DGdGAT1} \) and 0.463 for \( \text{DGdGAT3} \) while it is 0.373 for \( \text{DGdGAT2} \). Table 5-17 illustrates the serial fraction values of the diversity guided parallel models on the 100-Dimension test functions.

In fact, by comparing the performance of the diversity guided models in solving the 100D benchmark problems with the adaptive models (Chapter IV) we can notice the parallel performances of the adaptive models are slightly better than diversity guided ones \( \text{DGdGAT1} \) and \( \text{DGdGAT2} \), while the \( \text{DGdGAT3} \) is slightly better than \( \text{AdGAT3} \). By looking on Table 5-16 which is showing the mean speedup values of the adaptive parallel models of 100D test problems, we can see that the maximum achieved speedup is 2.02 for \( \text{AdGAT1} \), 2.845 for \( \text{AdGAT2} \) and 1.88 for \( \text{AdGAT3} \). While the maximum speedup values of diversity guided parallel models is 2.013 for \( \text{DGdGAT1} \), 2.426 for \( \text{DGdGAT2} \) and 2.015 for \( \text{DGdGAT3} \) with diversity levels of \( d_{\text{min}} = 0.2 \) & \( d_{\text{max}} = 0.8 \).

![Figure 5-12: The DGdGAT1, DGdGAT2 and DGdGAT3 Speedup values of the 100D benchmark test problems.](image)

Table 5-17 presents the mean serial fraction values of the adaptive models along with shifted and not shifted versions of the 100D benchmark problems. From this table we can confirm that the adaptive models from Chapter IV, \( \text{AdGAT1} \) and \( \text{AdGAT2} \),
perform better than the diversity guided models $DGdGAT1$ and $DGdGAT2$ respectively, while the performance of the diversity guided model $DGdGAT3$ seems to be slightly better than that of the adaptive model $AdGAT3$.

By inspection of Table 5-17 we can see that the best achieved serial fraction values with the adaptive regime (300) are 0.461 for AdGAT1 0.308 for AdGAT2 and 0.501 for AdGAT3. Meanwhile, the corresponding best serial fraction values of the diversity guided parallel model are 0.463 for DGdGAT1, 0.373 for DGdGAT2 and 0.463 for DGdGAT3 with diversity levels of $d_{\text{min}} = 0.2$ & $d_{\text{max}} = 0.8$.

![Figure 5-13: The DGdGAT1, DGdGAT2 and DGdGAT3 Serial Fraction values of the 100D benchmark test problems.](image)

Using the figures in Table 5-14 and Table 5-16, we find that the relative speedups obtained on the 100D benchmarks by the diversity-guided models, with reference to the adaptive models of Chapter IV, range between -18% and 6.8%. These figures are calculated according to the best speedup values of each algorithm using the following formula:
\[
\frac{\text{mean}(\text{the diversity's best speedup values})}{\text{mean}(\text{the adaptive's best speedup values})} - 1
\]

(5-3)

The Figure 5-14 and Figure 5-15 respectively illustrate the comparison between the mean speedup and serial fraction values of the Adaptive (100 & 300) and Diversity Guided ((0.2, 0.7), (0.2, 0.8)) models on the 100D benchmark test problems.

Table 5-15: The Serial Fraction values of the diversity guided models The DGdGAT1, DGdGAT2 and DGdGAT3 against the fixed models dGAT1, dGAT2 and dGAT3 of the 100D benchmark problems.

<table>
<thead>
<tr>
<th>Diversity Threshold min - max</th>
<th>DGdGAT1</th>
<th>DGdGAT2</th>
<th>DGdGAT3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1 - 0.5</td>
<td>3.366</td>
<td>2.216</td>
<td>3.637</td>
</tr>
<tr>
<td>0.1 - 0.6</td>
<td>3.206</td>
<td>1.95</td>
<td>3.271</td>
</tr>
<tr>
<td>0.1 - 0.7</td>
<td>2.932</td>
<td>1.823</td>
<td>3.008</td>
</tr>
<tr>
<td>0.1 - 0.8</td>
<td>2.422</td>
<td>1.518</td>
<td>2.84</td>
</tr>
<tr>
<td>0.2 - 0.5</td>
<td>0.712</td>
<td>0.64</td>
<td>0.676</td>
</tr>
<tr>
<td>0.2 - 0.6</td>
<td>0.657</td>
<td>0.492</td>
<td>0.612</td>
</tr>
<tr>
<td>0.2 - 0.7</td>
<td>0.5</td>
<td>0.428</td>
<td>0.497</td>
</tr>
<tr>
<td>0.2 - 0.8</td>
<td>0.463</td>
<td>0.373</td>
<td>0.463</td>
</tr>
<tr>
<td>0.3 - 0.5</td>
<td>2.532</td>
<td>1.436</td>
<td>1.687</td>
</tr>
<tr>
<td>0.3 - 0.6</td>
<td>1.871</td>
<td>1.25</td>
<td>1.567</td>
</tr>
<tr>
<td>0.3 - 0.7</td>
<td>1.573</td>
<td>1.106</td>
<td>1.35</td>
</tr>
<tr>
<td>0.3 - 0.8</td>
<td>1.35</td>
<td>0.975</td>
<td>1.16</td>
</tr>
<tr>
<td>0.4 - 0.5</td>
<td>7.546</td>
<td>6.321</td>
<td>7.899</td>
</tr>
<tr>
<td>0.4 - 0.6</td>
<td>7.108</td>
<td>4.861</td>
<td>5.149</td>
</tr>
<tr>
<td>0.4 - 0.7</td>
<td>4.392</td>
<td>2.746</td>
<td>5.101</td>
</tr>
<tr>
<td>0.4 - 0.8</td>
<td>3.417</td>
<td>2.445</td>
<td>4.812</td>
</tr>
</tbody>
</table>
Table 5-16: The mean Speedup values of the adapted models $AdGAT_1$, $AdGAT_2$ and $AdGAT_3$ of the 100D benchmark problems with adaptive scheme 100,300.

<table>
<thead>
<tr>
<th>Adaptive Scheme</th>
<th>$AdGAT_1$</th>
<th>$AdGAT_2$</th>
<th>$AdGAT_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Normal</td>
<td>Shifted</td>
<td>Mean</td>
</tr>
<tr>
<td>100</td>
<td>2.1</td>
<td>1.73</td>
<td>1.915</td>
</tr>
<tr>
<td>300</td>
<td>2.21</td>
<td>1.83</td>
<td>2.02</td>
</tr>
</tbody>
</table>

Table 5-17: The mean Serial Fraction values of the adapted models $AdGAT_1$, $AdGAT_2$ and $AdGAT_3$ of the 100D benchmark problems with adaptive scheme 100,300.

<table>
<thead>
<tr>
<th>Adaptive Scheme</th>
<th>$AdGAT_1$</th>
<th>$AdGAT_2$</th>
<th>$AdGAT_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Normal</td>
<td>Shifted</td>
<td>Mean</td>
</tr>
<tr>
<td>100</td>
<td>0.44</td>
<td>0.55</td>
<td>0.49</td>
</tr>
<tr>
<td>300</td>
<td>0.42</td>
<td>0.52</td>
<td>0.461</td>
</tr>
</tbody>
</table>

Figure 5-14: Comparison between the mean Speedup values of the Adaptive (100 & 300) and Diversity Guided ($d_{min} = 0.2$ & $d_{max} = 0.7, 0.8$) models of the 100D benchmark test problems.
In order to gain an understanding of the significance of the observations, we perform paired 1-tailed T-tests on the mean execution time figures in the various experiments, and focus on 100D benchmark functions. For clarity and simplicity, we restrict our statistical analyses to the diversity-guided cases that used the diversity threshold interval \((d_{\min}, d_{\max}) = (0.2, 0.8)\). The statistical tests have been done using a cut-off significance level of 0.05), and assuming unequal variance.

Table 5-18 summarises the statistical test results for the 100D functions, comparing the three \(DGdGA\) models against each other, and also segmenting the findings according to either shifted or non-shifted versions of the benchmark functions.

Table 5-18 shows, again, that \(DGdGAT2\) significantly outperforms the other models on the 100 Dimension test functions in terms of mean execution time in successful runs, adding weight to our earlier observations of the speedup and serial fraction values. For example, the T-Test p-values of \(DGdGAT1-DGdGAT2\) for all fitness functions are in the range of \([0.00072, 0.011]\), while the T-Test p-values of \(DGdGAT3-DGdGAT2\) for all problems are in the range of \([0.0015, 0.0217]\).
Table 5-18: The 100D benchmarks paired 1-tailed T-Test p-values for the comparison between the execution times of diversity guided models DGdGAT1, DGdGAT2 and DGdGAT3 using the diversity threshold of \((d_{\text{min}}, d_{\text{max}}) = (0.2, 0.8)\)

<table>
<thead>
<tr>
<th>Method</th>
<th>fSph</th>
<th>fRos</th>
<th>fRas</th>
<th>fSch</th>
<th>fGri</th>
<th>fAck</th>
</tr>
</thead>
<tbody>
<tr>
<td>DGdGAT1- DGdGAT2</td>
<td>0.001447</td>
<td>0.000721</td>
<td>0.001245</td>
<td>0.002359</td>
<td>0.001953</td>
<td>0.002543</td>
</tr>
<tr>
<td>DGdGAT3- DGdGAT2</td>
<td>0.006093</td>
<td>0.001693</td>
<td>0.00155</td>
<td>0.003053</td>
<td>0.004661</td>
<td>0.003073</td>
</tr>
<tr>
<td>DGdGAT1- DGdGAT3</td>
<td><strong>0.053231</strong></td>
<td>0.043621</td>
<td>0.040924</td>
<td><strong>0.066295</strong></td>
<td>0.043385</td>
<td>0.032317</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method</th>
<th>fS-Sph</th>
<th>fS-Ros</th>
<th>fS-Ras</th>
<th>fS-Sch</th>
<th>fS-Gri</th>
<th>fS-Ack</th>
</tr>
</thead>
<tbody>
<tr>
<td>DGdGAT1- DGdGAT2</td>
<td>0.003746</td>
<td>-</td>
<td>0.003241</td>
<td>0.007914</td>
<td>0.002986</td>
<td>0.011054</td>
</tr>
<tr>
<td>DGdGAT3- DGdGAT2</td>
<td>0.003165</td>
<td>-</td>
<td>0.002303</td>
<td>0.004325</td>
<td>0.021776</td>
<td>0.004842</td>
</tr>
<tr>
<td>DGdGAT1- DGdGAT3</td>
<td><strong>0.060038</strong></td>
<td>-</td>
<td>0.042151</td>
<td>0.039525</td>
<td><strong>0.082475</strong></td>
<td>0.049917</td>
</tr>
</tbody>
</table>

However, on the 100D problems, the superiority of **DGdGAT2** over **DGdGAT3** is not always significant at the 95% confidence level, as we can see in the cases of fSph, fSch, fS-Sph and fS-Gri. For example, the T-Test p-values of **DGdGAT1-DGdGAT3** for not shifted fitness functions are in the range of [0.032, 0.066], while the T-Test p-values of **DGdGAT1-DGdGAT3** for shifted problems are in the range of [0.039, 0.082].

**5.5 Conclusion**

In this chapter we developed a distributed GA model that adapts the rate of migration on the basis of diversity levels among its subpopulations. The approach we developed in this chapter was again tested in the context of the three different interconnection topologies introduced in Chapter III (T1, T2 and T3), and we therefore tested three dGAs in this chapter, called **DGdGAT1, DGdGAT2** and **DGdGAT3**. A key parameter in our diversity-guided approach is the diversity-threshold interval. This is an
interval of diversity levels (hence defined by two parameters) that is central to the way that the $DGdGA$ models adapt the migration rates from the subpopulations. We tested 16 different overall settings for this interval. As in Chapter IV, all distributed models in this chapter physically comprised 16 clients and one master process. We continued to use the 12 benchmark test functions that were used in Chapters III and IV to compare and evaluate our algorithms. Also in common with Chapter IV, we presented the results in three stages, corresponding respectively to the 30D, 50D, and 100D versions of the benchmark functions.

When we examined the 30D benchmark results in the first stage, we compared the DGdGA models with their non-adaptive counterparts in Chapter III. We found that, given the right choice of diversity threshold interval, the $DGdGA$ models showed clearly better performance than their corresponding non-adaptive models. Whereas we saw mean success rates for the non-adaptive models in the range $[91.25\%, 94.17\%]$ on the 30D problems, the best choices of diversity threshold levels led to success rates never below $96.25\%$ for the diversity-guided models $d_{\min} = 0.2 \& d_{\max} = 0.7, 0.8$. These initial results showed that the diversity-guided models seemed to have potential merit, although revealed that performance could be sensitive to the diversity threshold parameters. Inspection of the mean speedup and serial fraction values then showed that the diversity guided models exhibited a small enhancement in their performance when compared to the non-adaptive (but still distributed) models in Chapter III. We saw speedup values (the diversity-guided models compared with the non-adaptive distributed models of Chapter III) in the range $[0.7, 1.365]$, and in the best cases, the diversity guided models were able to produce speedups 20\% (for $DGdGAT1$) and 33\% (for $DGdGAT2$) in comparison to the non-adaptive counterparts using the same topology.

When we looked at the 50D benchmark functions, we found that the diversity-guided model was again able to outperform the non-adaptive distributed model, but this time with a somewhat increased margin, and appearing better on every benchmark problem. Success rate, speedup and serial fraction figures confirmed these observations. However, we also started to see evidence of increased sensitivity of the diversity-guided models to the diversity threshold levels as the problem dimensionality increased. The best successful rates for all models were $>96\%$ with diversity level $d_{\min} = 0.2 \& d_{\max} = 0.8$. 

140
Also, we noticed that the model \textit{DGdGAT2} seemed clearly superior to the other two versions, while \textit{DGdGAT3} seemed marginally better than the model \textit{DGdGAT1}. For example, with the diversity threshold $d_{\text{min}} = 0.2$ \& $d_{\text{max}} = 0.8$ the success rates for the three models on the 50D functions were 96.36\% for \textit{DGdGAT1}, 97.73\% for \textit{DGdGAT2}, and 96.82\% for \textit{DGdGAT3}.

In the third stage of experiments, we looked at the 100D versions of the benchmark functions. This time, one of the clearest findings from the results was the further increased sensitivity of the performance of the diversity-guided models to the settings of the diversity threshold levels. While success rates were still high for well-chosen diversity threshold intervals, a poor choice of this interval led to success rates in the region of 50\% (never below 63\% in the case of the 50D functions).

We also observed that \textit{DGdGAT2} continued to perform very well in comparison to the other models, an almost maintained the same success rates (given the appropriate diversity threshold levels), despite the increase in dimensionality, while \textit{DGdGAT3} seemed to suffer the most in terms of drop in success rate.

Consideration of the speedup and serial fraction metrics generally aligned with our observations based on success rates. We noticed that, at best (i.e. with the appropriate choices of diversity threshold level), the diversity-guided models could do more than double the speed of optimization shown by the non-adaptive distributed models in Chapter III on the 100D functions.

Finally, we performed some statistical analyses over selected results to investigate which of our observations may be significant. These T-tests showed the \textit{DGdGAT2} was certainly superior in terms of execution time to each of \textit{DGdGAT1} and \textit{DGdGAT3}, for all benchmark problems at 50D and 100D. Also, the T-Tests confirmed that there was no statistical difference between \textit{DGdGAT1} and \textit{DGdGAT3} for most test functions.
CHAPTER VI
VALIDATION OF ALGORITHMS ON REAL-WORLD PROBLEMS

In all our previous experiments in Chapter III, Chapter IV and Chapter V we concentrated on using one group of test functions comprising 12 well studied function optimization problems. In order to broaden our understanding of the performance of our new distributed algorithms we continued to use the same test functions, but raised their difficulty by testing on higher dimensional variants of them. In this chapter we move away from standard test functions, which may not in general represent the characteristics of real-world functions. We now explore the performance of our algorithms on a range of real-world numerical optimization problems.

6.1 The Real-World Test Problems IEEE-CEC’11

There are many suites of benchmark functions for optimization problems, but in recent years researchers have expressed dissatisfaction with these suites, in terms of how well they represent the nature and difficulty of real-world problems. In this chapter we use a suite of problems that was designed to address this issue. The real-world benchmarks we use comprise 22 different test functions, detailed in technical report [50], which were used in the competition on testing evolutionary algorithms during the conference IEEE-CEC 2011. The winner of the competition winner was an algorithm called GA-MPC [69], and in this chapter we will often refer to the results of GA-MPC on these functions, to provide a general reference point regarding the quality of the results of our algorithms. These 22 test functions vary widely in terms of the number of parameters to be optimized (dimension), from only one dimension (problems P_3 and P_4) to 216 dimensions (problem P_{12}). The range of size, difficulty and nature of the 22 CEC 2011 problems provides a broad test for the algorithms developed in this thesis, and the aim of these experiments is to further understand the general applicability of our new algorithms. We summarise the test problems in Table B-11.
A total of 14 algorithms took part in the CEC 2011 competition, and, as mentioned above, the algorithm that won the competition was called \textit{GA-MPC} [69]. This algorithm consisted of a new form of crossover operator with a randomized operator that attempted with probability $P$ to escape from local minima. Also, \textit{GA-MPC} employed mechanisms that attempt to ensure that good population diversity is maintained.

\section*{6.2 The Test Results}

In all of our experiments on the CEC 2011 test suite, the parameters, attributes and physical distributed setup for our algorithms are as we used before for the main experiments in Chapters III, IV and V, always consisting of 16 clients and one master process. Following the rules in the CEC 2011 competition [50], each experiment with a specific algorithm and problem was repeated for 25 independent trials, each trial continued for a maximum of 150,000 fitness evaluations, and results were collated to conform to the CEC 2011 performance comparison report format [235]. As a first view of our results, Table 6-1 shows the mean fitness value on each CEC 2011 problem for each of the nine algorithms explored in this thesis, and also shows the corresponding mean fitness value for the CEC 2011 competition winner \textit{GA-MPC}. Note that, in every case, the problem is one of minimization.

From Table 6-1 we can notice that the distributed algorithms presented in this thesis show very strong comparative performance on the CEC 2011 functions, with most of the nine algorithms achieving a better mean fitness than GA-MPC on every real-world problem. For example, the mean fitness for problem $P_4$ achieved by \textit{GA-MPC} was 13.8154, while all nine of the distributed models in this thesis achieved a mean best fitness of 0.316 on this problem. Also, the adaptive models from chapter IV, \textit{AdGAT1}, \textit{AdGAT2} and \textit{AdGAT3}, seem to be the best in performance (from the viewpoint of mean fitness value).
Table 6-1: The mean fitness value for 22 test functions optimized by the studied models and GA-MPC

<table>
<thead>
<tr>
<th>Algo/Prob</th>
<th>P₁</th>
<th>P₂</th>
<th>P₃</th>
<th>P₄</th>
<th>P₅</th>
<th>P₆</th>
<th>P₇</th>
</tr>
</thead>
<tbody>
<tr>
<td>GA-MPC</td>
<td>0</td>
<td>-27.701</td>
<td>1.2E-05</td>
<td>13.8154</td>
<td>-35.039</td>
<td>-27.488</td>
<td>0.74841</td>
</tr>
<tr>
<td>dGAT1</td>
<td>0</td>
<td>-28.88</td>
<td>-0.001</td>
<td>0.316</td>
<td>-37.34</td>
<td>-28.08</td>
<td>0.49</td>
</tr>
<tr>
<td>dGAT2</td>
<td>0</td>
<td>-29.18</td>
<td>-0.001</td>
<td>0.316</td>
<td>-38.96</td>
<td>-28.29</td>
<td>0.49</td>
</tr>
<tr>
<td>dGAT3</td>
<td>0</td>
<td>-28.88</td>
<td>-0.001</td>
<td>0.316</td>
<td>-36.12</td>
<td>-28.87</td>
<td>0.49</td>
</tr>
<tr>
<td>AdGAT1</td>
<td>0</td>
<td>-29.49</td>
<td>-0.009</td>
<td>0.316</td>
<td>-37.92</td>
<td>-28.42</td>
<td>0.48</td>
</tr>
<tr>
<td>AdGAT2</td>
<td>0</td>
<td>-30.46</td>
<td>-0.004</td>
<td>0.316</td>
<td>-38.91</td>
<td>-29.166</td>
<td>0.41</td>
</tr>
<tr>
<td>AdGAT3</td>
<td>0</td>
<td>-30.32</td>
<td>-0.004</td>
<td>0.316</td>
<td>-38.15</td>
<td>-29.016</td>
<td>0.45</td>
</tr>
<tr>
<td>DGdGAT1</td>
<td>0</td>
<td>-29.01</td>
<td>-0.0061</td>
<td>0.316</td>
<td>-37.92</td>
<td>-28.17</td>
<td>0.49</td>
</tr>
<tr>
<td>DGdGAT2</td>
<td>0</td>
<td>-29.98</td>
<td>-0.005</td>
<td>0.316</td>
<td>-38.12</td>
<td>-28.87</td>
<td>0.45</td>
</tr>
<tr>
<td>DGdGAT3</td>
<td>0</td>
<td>-29.41</td>
<td>-0.006</td>
<td>0.316</td>
<td>-37.96</td>
<td>-28.68</td>
<td>0.46</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Algo/Prob</th>
<th>P₈</th>
<th>P₉</th>
<th>P₁₀</th>
<th>P₁₁</th>
<th>P₁₂</th>
<th>P₁₃</th>
<th>P₁₄</th>
</tr>
</thead>
<tbody>
<tr>
<td>GA-MPC</td>
<td>220</td>
<td>1220.59</td>
<td>-21.702</td>
<td>52054.6</td>
<td>1073382</td>
<td>15444.2</td>
<td>18261</td>
</tr>
<tr>
<td>dGAT1</td>
<td>220</td>
<td>1126.62</td>
<td>-21.21</td>
<td>52053.3</td>
<td>1046916</td>
<td>14980.7</td>
<td>17569.4</td>
</tr>
<tr>
<td>dGAT2</td>
<td>220</td>
<td>1126.62</td>
<td>-21.65</td>
<td>52022.4</td>
<td>1025551</td>
<td>14980.7</td>
<td>17550.1</td>
</tr>
<tr>
<td>dGAT3</td>
<td>220</td>
<td>1115.24</td>
<td>-21.01</td>
<td>52027.9</td>
<td>1025551</td>
<td>14980.1</td>
<td>17592.3</td>
</tr>
<tr>
<td>AdGAT1</td>
<td>220</td>
<td>1113.86</td>
<td>-21.21</td>
<td>52022.4</td>
<td>1025551</td>
<td>14826.2</td>
<td>17592.3</td>
</tr>
<tr>
<td>AdGAT2</td>
<td>220</td>
<td>1103.13</td>
<td>-21.87</td>
<td>51457.3</td>
<td>1020282</td>
<td>14644.3</td>
<td>17373.4</td>
</tr>
<tr>
<td>AdGAT3</td>
<td>220</td>
<td>1118.21</td>
<td>-21.87</td>
<td>52017.6</td>
<td>1025282</td>
<td>14644.6</td>
<td>17373.4</td>
</tr>
<tr>
<td>DGdGAT1</td>
<td>220</td>
<td>1138.46</td>
<td>-21.01</td>
<td>52057.6</td>
<td>1068282</td>
<td>14844.5</td>
<td>17592.3</td>
</tr>
<tr>
<td>DGdGAT2</td>
<td>220</td>
<td>1122.48</td>
<td>-21.77</td>
<td>52017.9</td>
<td>1025551</td>
<td>14735.7</td>
<td>17379.4</td>
</tr>
</tbody>
</table>
The problem \( P_6 \) was optimized by \textit{GA-MPC} with fitness value of -27.488 while it has been optimized with the studied models with fitness value in the range between -28.08 by \textit{dGAT1} and -29.166 by \textit{AdGAT2}. Also, the problem \( P_{21} \) was optimized by \textit{GA-MPC} with fitness value of 12.81 while it has been optimized with the studied models with fitness value in the range between 7.01 by \textit{DGdGAT1} and 6.01 by \textit{AdGAT2} and \textit{AdGAT3}. Thus, we can see that the studied models successfully optimized all problems with better fitness values than those obtained by the CEC11 competition winner the algorithm \textit{GA-MPC}.

Figure 6-1 shows the convergence/time behaviour of the nine distributed algorithms on a selected problem (\( P_1 \)), again compared with the same for \textit{GA-MPC}. This example is representative of similar convergence patterns that we observed for each of the 22 CEC 2011 functions, showing that the adaptive models from Chapter IV,
and in particular \textit{AdGAT2}, tended to converge the most efficiently, in terms of improvements in fitness per fitness evaluation. Appendix B provides further details that show the number of fitness evaluations to converge to the best-found solution for each of the experiments in this chapter. From the computational time record, it is interesting to report here that the average computation times required by adaptive models for most problems are approximately 30% to 90% lower than the same of the other models.

Our results on the 22 test problems from the CEC 2011 real-world function suite show clear evidence that the adaptive parallel genetic algorithms \textit{AdGAT}_{1,2,3} are effective optimization algorithms, both in terms of the quality of solutions reached and in terms of time efficiency. Table 6-2 shows some problems’ speedup values in comparison between the computation time of \textit{AdGAT}_{1,2,3} with d\textit{GAT}_{1,2,3} and \textit{AdGAT}_{1,2,3} with DG\textit{dGAT}_{1,2,3} in addition to DGd\text{GAT}_{1,2,3} with d\textit{GAT}_{1,2,3}. These problems in Table 6-2 are carefully selected from our benchmark test functions to cover different types and complexity levels with the aim of showing how the studied algorithms and topologies behave with different problems. Appendix B shows the full list of speed up values for all problems.

<table>
<thead>
<tr>
<th>Method / Problem</th>
<th>P1</th>
<th>P9</th>
<th>P12</th>
<th>P15</th>
<th>P17</th>
<th>P20</th>
<th>Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>AdGAT1-dGAT1</td>
<td>1.512</td>
<td>1.69</td>
<td>2.224</td>
<td>1.303</td>
<td>1.159</td>
<td>1.045</td>
<td>1.489</td>
</tr>
<tr>
<td>AdGAT2-dGAT2</td>
<td>2.217</td>
<td>2.81</td>
<td>3.208</td>
<td>1.303</td>
<td>1.159</td>
<td>1.045</td>
<td>1.957</td>
</tr>
<tr>
<td>AdGAT3-dGAT3</td>
<td>2.029</td>
<td>2.361</td>
<td>3.346</td>
<td>1.303</td>
<td>1.159</td>
<td>1.045</td>
<td>1.874</td>
</tr>
<tr>
<td>AdGAT1-DGdGAT1</td>
<td>1.421</td>
<td>1.289</td>
<td>1.748</td>
<td>1.303</td>
<td>1.159</td>
<td>1.045</td>
<td>1.327</td>
</tr>
<tr>
<td>AdGAT2-DGdGAT2</td>
<td>1.766</td>
<td>1.485</td>
<td>1.638</td>
<td>1.303</td>
<td>1.159</td>
<td>1.045</td>
<td>1.399</td>
</tr>
<tr>
<td>AdGAT3-DGdGAT3</td>
<td>1.565</td>
<td>1.932</td>
<td>1.704</td>
<td>1.303</td>
<td>1.159</td>
<td>1.045</td>
<td>1.451</td>
</tr>
<tr>
<td>DGdGAT1-dGAT1</td>
<td>1.064</td>
<td>1.311</td>
<td>1.71</td>
<td>1.303</td>
<td>1.159</td>
<td>1.045</td>
<td>1.265</td>
</tr>
<tr>
<td>DGdGAT2-dGAT2</td>
<td>1.255</td>
<td>1.892</td>
<td>1.959</td>
<td>1.303</td>
<td>1.159</td>
<td>1.045</td>
<td>1.435</td>
</tr>
<tr>
<td>DGdGAT3-dGAT3</td>
<td>1.296</td>
<td>1.222</td>
<td>1.963</td>
<td>1.303</td>
<td>1.159</td>
<td>1.045</td>
<td>1.331</td>
</tr>
</tbody>
</table>
Figure 6-1: Convergence plots for problem P1, a) GA-MPC and \( dGAT_{1,2,3} \)

b) \( DGdGAT_{1,2,3} \) c) \( AdGAT_{1,2,3} \)
From Table 6-2 we can see that the speedup values are spread within the range [1.045, 3.346], indicating that the relative performances among the models studied in this thesis are somewhat problem-dependent. It is of course no surprise that the performance of any algorithm is problem-dependent, however strong relative performance over a wide range of problems provides evidence that an algorithm will perform reasonably well on 'arbitrary' new optimisation problems, and the CEC 2011 real-world problem suite provides a good set of problems to help make such a judgement. In the case of the algorithms explored in this thesis, from Table 6-1 and Figure 6-1 it seems clear that \textit{AdGAT1}, \textit{AdGAT2} and \textit{AdGAT3} are the most promising for future study.

The convergence behaviour visualised in Figure 6-1 indicates that these particular adaptive migration methods converge faster than the other methods explored here, and find better overall solutions, also finding better solutions than those found by \textit{GA-MPC}.

### 6.3 Conclusion

Continuing to explore and test our new distributed EA models, we have tested the nine approaches developed in this thesis on all of the 22 real world problems of the CEC 2011 competition, and compared the results with competition winner algorithm \textit{GA-MPC}. This provided a 'validation' of our algorithms in the context of real-world problems, as well as enabled us to explore their performance on a wider range of different problem types, with dimensionality ranging from 1 to 216.

Following the CEC2011 rules [50], 25 trials were run for each problem, and a variety of result indicators were recorded, Table 6-1 summarises the results for all of these problems. In each case, we see the mean of 25 trials, reported for each of algorithms \textit{dGAT}_{1,2,3}, \textit{DGdGAT}_{1,2,3} and \textit{AdGAT}_{1,2,3}. For the comparative algorithm \textit{GA-MPC}, we take the mean results directly from the cited publications. If we observe the results summarized in Table 6-1 and Figure 6-1 we can conclude that the proposed algorithms \textit{dGAT}_{1,2,3}, \textit{DGdGAT}_{1,2,3} and \textit{AdGAT}_{1,2,3} show good performance in optimizing all benchmark test functions, in terms of both the final solution quality and computation time. These findings add to the evidence from earlier chapters that suggests each of the following: (i) in the context of distributed EAs, adaptive migration schemes
are more effective than non-adaptive migration schemes; (ii) our adaptive approach based on progress within a subpopulation is more effective than our adaptive approach based on diversity within a subpopulation; (iii) interconnection topology $T_2$ is a more effective substrate for the distributed algorithms than the simpler topology $T_1$ and the more sophisticated 'cube'-based topology $T_3$.

Thus, we found increasing evidence in this chapter that $AdGAT_{1,2,3}$ and $AdGAT_2$ in particular, are promising methods for further research. A further important finding was that the relative performance advantage of $AdGAT_2$ seems to improve as problem dimensionality increases; so, $AdGAT_2$ may be particularly recommended for high-dimensional problems.
CHAPTER VII

CONCLUSION

“Such a work is never actually finished – one must simply declare it to be finished once one has done as much as possible given the time and circumstances.”

Johann Wolfgang von Goethe (1749 – 1832)

7.1 Summary

The aim of this research was partly to study the performance of Distributed Evolutionary Algorithms (dEAs) on different parallel topologies, and mainly to study the behaviour of different schemes for adaptation of the migration process in this context. Researchers and practitioners are interested in dEAs because many real-world problems take a very long time to solve on serial hardware. In this thesis, we began with a review of the literature, and briefly introduced the reader to the history and development of EAs from early ideas in population genetics to modern theoretical models of EAs. This was done to prepare the way for the introduction of several functions that were proven or demonstrated to be difficult or outright deceptive for the genetic operators. Evolutionary algorithms work with populations of independent solution which make it naturally easy to distribute the computational load among several processes. EAs are “embarrassingly parallel” which make it very easy to be efficiently implemented on parallel computers.

There has been considerable work on the benefits of parallelism in EAs. However there has been very little work on adaptive migration schemes. Most other work on dEAs so far seems to focus on the very wide range of other issues relevant to dEAs, but use a fixed migration rate. In particular, the design of distributed EAs involves choices such as using single or multiple populations; in both cases, the size of population must be decided carefully, and for multiple population it has to decided how many to be used. There are also three main types of distributed EAs: global single-population master-slave, single-population fine-grained and multiple-population coarse-grained dEAs. Each of these broad approaches has an extensive amount of research effort devoted to it. However, in this thesis we focussed on only one of these three: multiple-population coarse-grained dEAs. Since such dEAs comprise distinct subpopulations, they are
naturally the relevant models under which to study migration between subpopulations. In such dEAs, the size of an individual subpopulation (called a 'deme') tends to be smaller than the population size used by a serial EA; this goes along with the expectation that, all else being the same, the demes of a distributed EA would converge faster with the risk of poor quality solutions. The important characteristics of multiple-deme coarse-grained dEAs are the use of a few relatively large subpopulations and migration, which makes this type of algorithms very complex and its behaviour is affected by many parameters.

In the research literature, one can notice different levels of heterogeneity in dEAs, with regard to search strategies (e.g. these may be different in different subpopulations), migration schemes, and the relationship maintained among the subpopulations. This means that the search occurs at multiple exploration and exploitation levels. Our parallel models (as described in Chapter III) are implemented as physical parallelization of a heterogeneous dEA which is sometime referred in the literature as gradual distributed real-coded GA with control of individuals migration among subpopulations; in most of the physical implementations we explored, sixteen populations were run concurrently using one of three different parallel topologies, and one of three basic schemes for migration of individuals between demes. In Chapter III we explored a specific basic dEA model, realised over three distinct topologies, T1, T2 and T3. These three models, \textit{dGAT1}, \textit{dGAT2} and \textit{dGAT3}, each showed very clear advantages in both speed and solution quality when compared to a serial GA (SGA) which was otherwise the same - that is, running the same basic GA that operated in each deme of the distributed models.

The advantage of \textit{dGAT1} is that it can speed up the optimization procedure and save evaluations by using a good (albeit simple) migration control scheme, which promotes exploitation of good individuals at the same time as preventing stagnation in demes that might otherwise stay close to poor solutions. However, the apparent disadvantage of \textit{dGAT1} is the load on the master process - all migration is controlled by the master, and the consequent communication load may lead to slowdown of the whole process. Meanwhile, \textit{dGAT3} was based on topology T3, which was drawn from the recent literature. The model \textit{dGAT3} shares the advantage of \textit{dGAT1} that comes simply from having a migration scheme at all. However T3's 'cube'-based topology involves migration occurring locally among neighbours among the vertices of a virtual cube, while the main job of the master process is to supervise this activity. This distributed the overall communication load, and led to better overall performance for \textit{dGAT3} when
compared with dGAT1. Nevertheless, the potential disadvantage of this model is that many of the migrations that are done between subpopulations are wasteful in the sense that the best individuals in the local region are being propagated, ignoring perhaps better individuals elsewhere in the virtual cube. Arguably this may be more damaging when the fixed migration rate is high, although in general the dynamics of such activity are very difficult to predict. Topology T2 was also designed to reduce the communication load for the master process, and distribute some of the communication requirements evenly over the clients, but in a simpler way than T3. In T2, subpopulations are grouped in pairs, and only one from each pair is directly connected to the master process. However, similar to the potential disadvantage of T3, the master process is not immediately aware of good new chromosomes that may appear in half of the overall population. The results detailed in Chapter III showed that the basic parallel models dGAT1, dGAT2 and dGAT3 were able to achieve super-linear speed up over the serial genetic algorithm for the same test problems, and also showed a slight but consistent overall advantage in performance for dGAT2. It is appropriate to assume that the improvements we see in success rates, when comparing the distributed with the serial models, are mainly due to the migration activity, since this is the primary element that makes the algorithmic structure of the dEA different from the serial EA. In Chapter III, however, and in common with most work in this field, the migration process was non-adaptive - that is, migration of a chromosome between subpopulations simply happened at a priori fixed intervals.

Following the work in Chapter III, we therefore started in Chapter IV to investigate more sophisticated migration schemes. This was done in Chapter IV by using a migration scheme in which the rate of migration was governed by performance within a deme; when a deme was progressing well, migrations from that deme were more frequent. Further, to provide a strong test of this scheme, additional versions of the test problems were used, and we compared these new adaptive models, AdGAT1, AdGAT2 and AdGAT3, with the Chapter III approaches on 30, 50 and 100 dimension variants of the test problems. The results of these tests confirmed that the new methods AdGAT1, AdGAT2 and AdGAT3 have the capability to outperform the non-adaptive approaches, and achieve near-linear speedup over dGAT1, dGAT2 and dGAT3 in the case of most of the test problems (and therefore advancing further beyond the super-linear speedup over SGA). Reflecting analogous findings in Chapter III, statistical tests also confirmed that AdGAT2 has superior performance to the two other topologies, and that there is no significant difference between AdGAT1 and AdGAT3 for most test
functions. It was also seen that the advantage of the adaptive models over the non-adaptive models increased with the dimensionality of the test problem.

We went forward in Chapter V to investigate an alternative idea for adapting the migration process in dEAs. This time, migration of new chromosomes into a subpopulation was partly based on the genetic diversity level within that subpopulation. Where diversity was within a given interval, new artificially generated chromosomes (in part constructed from recent global best-so-far chromosomes) were injected into the population. Further, when diversity was below the lower valued threshold of this interval, a deme was forced to reset its population. The new models $DGdGAT1$, $DGdGAT2$ and $DGdGAT3$, successfully used the population diversity control regime to optimize our benchmark test problems with 30, 50 and 100 dimensions. The experiments in Chapter V were repeated for several different settings of the diversity threshold interval, comprising the lower and upper diversity levels between which demes were injected with artificial chromosomes.

One of the clear findings was that there was a region of diversity threshold settings that had generally good performance over all problems. It was also found, as before, that these new adaptive models were again superior to the non-adaptive models of Chapter III, and also that the topology T2 version of the new models was superior to the T1 and T3 versions. However, in general it seemed clear that the performance-based adaptation schemes of Chapter IV were more successful than the diversity-guided schemes of Chapter V. Finally, in chapter 6 we performed tests of all of the Chapters III, IV and V methods on a suite of real-world problems with varying dimensionality and nature. This was designed to validate and test the previous 'benchmark function' observations, and provide evidence about the potential performance of our methods on unseen real problems. The results of chapter 6 confirmed all of the main general observations in the previous chapters, and added further weight to the observation that migration control has a significant effect in dEAs. Reflecting the performance on test functions in Chapters IV and V, we found that the adaptive migration algorithms demonstrated clearly better performance than the non-adaptive dEAs on the real-world problem suite. Also, reflecting observations in earlier chapters, we found that the best performance was obtained from the Chapter IV models, $AdGAT_{1,2,3}$. In terms of execution time towards solving the problems, the performance of $AdGAT_{1,2,3}$ was generally 30% to 90% faster than $dGAT_{1,2,3}$ and $DGdGAT_{1,2,3}$ for the same problems.
7.2 Future Work

The empirical investigations undertaken in this thesis, and our findings, lead to several suggestions for fruitful future work, aimed at achieving further improvements in performance and generally better understanding of performance in dEA models, particularly with regard to the migration schemes:

- First, there are of course many parameters and design choices that affect the performance of dEAs, and in this thesis we have investigated only a small selection. The very fact that there are so many configuration possibilities makes it very difficult to understand how performance is related to groups of parameter settings. For example, although we may consider it unlikely, it could be the case that the performances achieved in this thesis are sensitive to our choice of crossover operator, and a different choice (or no crossover operator) may have revealed different observations in the differential performance of the different adaptive methods. We therefore think there is a need for very large scale analysis of dEAs, in which perhaps thousands of separate parameter configurations are contrasted in a large study, with appropriate statistical analyses done to identify any robust relationships between configuration and performance.

- Meanwhile, we think it is clear, and further demonstrated in this thesis, that adaptive migration schemes are very effective in the context of dEAs. There is a very large space of possibilities for such schemes. In this thesis we have only investigated a small region of that space of possibilities. Further work on alternative adaptive migration schemes is thus recommended, and the evidence from this thesis suggests that such work could be directed more successfully along the lines of adaptation on the basis of performance rather than diversity. Further, in this thesis we performed limited investigation of different interconnection topologies. However even this limited investigation showed that the effect of the basic interconnection topology is significant in the EA context.

- Obviously, the interconnection topology defines which demes are 'neighbours' and therefore also interacts closely with the migration scheme. Nevertheless, we did find that topology T2 maintained an advantage over the other topologies studied in this
thesis, which seemed independent of the migration scheme. It therefore seems that more study would be beneficial which focussed on a wide range of simple topologies based on T2, but keeping other aspects constant.
A.1 Synthesis of Evolutionary Computation

Scholars, at the end of 1980s and beginning of 1990s, started meeting irregularly at some conferences to deliver and present their researches and communicate ideas. Such conferences may be mentioned as an example of that, but not limited to, are the International Conference on Genetic Algorithms (ICGA) [103] and Parallel Problem Solving from Nature (PPSN) [218]. These two conferences were established in 1985 and 1990 correspondingly as biannual conferences. Though, the IEEE Conference on Evolutionary Computation (ICEC) was started in 1994 as an annual one. However, at the end of 1999s, the two conferences ICGA and ICEC were combined with GP and EP conferences respectively generating the annual GECCO [26] and CEC [1] conferences.

The term of Evolutionary Computation was intended at the beggning of PPSN conference to include all EA research dialects. In addition, the term Evolutionary Algorithm was used as the generic form of the group of algorithms. The following definition gives an example of them main processes in the Evolutionary Algorithm. This formulation can be fit on the general Evolutionary Algorithms which have not been integrated with other algorithms.

Evolutionary Algorithm:

1. INITIALIZE population with random candidate solutions.
2. EVALUATE each candidate.
3. REPEAT UNTIL TERMINATION CONDITION:
   a. SELECT parent candidates. (Reproductive selection)
   b. RECOMBINE AND/OR MUTATE parent candidates into child candidates.
   c. EVALUATE child candidates.
   d. SELECT candidates for next generation. (Survival selection)
A.2 Population Size

In literature there are also lots of empirical works on population sizing. Arabas and his co-researcher [12] introduced “Genetic Algorithm with Variable Population Size (GAVaPS)” which considers the size of population as an obvious parameter through adding age and highest level of lifetime attribute for each individual in any given population. By that and upon the individual’s birth, the maximum level of lifetimes is assigned to it depending on its newborn fitness, although the age set to zero at birth. The individual’s age is increased by one for each generation, and when individual’s age reaches its limit of predefined level of lifetime it is marked as expired and then removed from the current population. However, this mechanism works towards making population size recognizable with dynamic adjusting and not just a parameter, as well as it makes survivor selection strategy unnecessary. In contrast of that, Bäck et al. [21] introduced “The Adaptive Population size Genetic Algorithm (APGA)” which uses a steady-state GA. The APGA works in order to maintain the best individual’s lifetime invariable through processing time when population individuals become older.

Harik and Lobo [113] and Lobo [87, 152] pointed a “parameter-less GA (PLGA)” which simultaneously develops a list of different sizes of populations. In this instance, PLGA allow more function evaluations for smaller populations than the bigger ones, for example, population \( P_i \) is evolved and create generations four times more than the population \( P_{i+1} \). However, the algorithm will drop any smaller population once it converges. Costa et al [44] introduced “Random Variation of the Population Size GA (RVPS)”. Through this algorithm, they compared GAs with fixed and varying population sizes, for a given N they aimed to make population size varied every N fitness evaluations. They found out, the GA’s performance with changeable population size, in the worst case, is doing the same effort of the simple GA, regarding less the population size is changed randomly throughout the process.

Hinterding, et al. [121] illustrated an adaptive GA technique. They mainly used three groups of populations along with different population size each. The idea behind this technique was to make the size of these populations to be adapted at regular periods (epochs) on the basis of the current state of the search, leading algorithm to make the performance of the group in the maximum with the mid-most size. However, fitness diversity was used for this algorithm as criterion to varying the population sizes.

Schlierkamp-Voosen and Mühlenbein [213] work used a competition scheme between sub-populations. They aimed to modify all sub populations’ size along with
changing the size of total population. There is a quality and gain criteria attached to each group, which control the change’s level in the size of groups. However, the algorithm mechanism was built on the way to allow only the size of the fittest group to increase. Another empirical study based on Goldberg’s research [91] was introduced by Robert Smith [227] which uses a dynamic mechanism to adapt the size of population under controlling of the selection error’s probability. He concluded that by automatically adapting population size, Genetic Algorithms could control their search process and would effectively adjust and contract to control variations in problem complexity and in available computational resources.

A.3 Selection Operators

Unlike linear ranking selection, the exponential ranking selection schema [169] assigns probabilities of the ranked individuals through exponential weighting. Parameter \( c \) is used as the base of the exponent in which \( c \in [0, 1] \) of the selection method. Thus, the probability of the each individual can be illustrated as follow:

\[
P_i = \frac{c^{\mu - i}}{\sum_{j=1}^{\mu} c^{\mu - j}} ; \quad i \in \{1, \ldots , \mu\}
\]

The sum \( \sum_{i=1}^{\mu} c^{\mu - i} \) normalizes the probabilities to make sure that \( \sum_{i=1}^{\mu} P_i = 1 \)

Tournament selection as it was stated by Bäck [14] and Blickle [29], may be considered as the second most well-known reproductive selection strategy. A group of size \( t \in [2, \mu] \) of individuals from the population are randomly chosen. These individuals according to their fitness are dynamically ranked and then the top two chosen as parents for the next generation individuals which latter will participate within the genetic operations like mutation and crossover. This process will be repeated until producing new population of the next generation.

Boltzmann schemes is an important selection technique in EA as it has theoretical properties which help in theoretical analysis, it’s derived from Simulated Annealing (SA) [144]. SA is employing such search scheme so called generate-and-test which derived from a physical analogy, instead of a biological one. Theoretically it’s formed
on a similarity of annealing techniques employed to reduce the internal stresses in the material sample. Annealing mechanism achieved by first increasing material temperature to raise the size of its crystals and release the stresses, and then the temperature slowly decreased with control to allow the stresses to become stable throughout the material.

Boltzmann selection schemes have many commonalities to the temperature control of simulated annealing, in which the selection pressure can be managed analogously to temperature. This gives the advantage instead of using constant selection pressure, it can be controlled. In the initial set of generations, the selection pressure can be slowly decreased (like SA temperature), which will initiate a more explorative mode. Then, after a while of the process and a better area of the search space have been allocated, the selection pressure may be increased towards highly-fit individuals which is similar to temperature dropping in Simulated Annealing in order to reach zero degree for more rapidly towards the optimum. For more details about Boltzmann selection see good text of Mahfoud & Goldberg [160] and de la Maza & Tidor [56].

A.4 Mutation Operator

Hesser and Männer [119] derived theoretically optimal schedules for deterministically changing \( P_m \) for the counting-ones function. They suggest

\[
P_m = \sqrt{\alpha \beta} \cdot \frac{\exp \left( \frac{-\gamma T}{Z} \right)}{\lambda \sqrt{T}}
\]

Where \( \alpha, \beta, \gamma \) are constant, and \( t \) (time) is the generation number. In contrast of that, another research of Bäck [15] illustrated an optimal mutation rate decrease method of \( P_m \) in opposed to a function of time as a function of distance, like:

\[
P_m \left( f(x') \right) \approx \frac{1}{2(f(x') + 1) - l}
\]

While the function to control decrease schedule of \( P_m \) illustrated by Bäck and Schütz [23] used time \( (t) \) instead of distance which constrained \( P_m(t) \) so that the initial
mutation rate value $P_m(t) = 0$ and for later stages $P_m(T) = \frac{1}{t}$, where $T$ is the maximum evaluation number, which can be clearly noticed from following formula:

$$P_m(t) = (2 + \frac{l - 2}{T}.t)^{-1}, \quad 0 \leq t \leq T$$

Janikow and Michalewicz [130] through their experimental study tested a non-uniform mutation (NUM) where for a selected individual $S^t = \{v_1, v_2, ..., v_k, ..., v_l\}$ of current generation $(t)$ and $v_k \in [l_k, u_k]$ where $l_k$ and $u_k$ are the lower and upper bounds of $v_k$ respectively, be the element to be mutated for the next generation $(t+1)$, the resulting individual $S^{t+1}$ which can be represented as $S^{t+1} = \{v_1, v_2, ..., v'_k, ..., v_l\}$ with $k \in [1, l]$, where:

$$v'_k = \begin{cases} v_k + \Delta(t, u_k - v_k), & \text{if random digit is 0} \\ v_k - \Delta(t, v_k - l_k), & \text{if random digit is 1} \end{cases}$$

The function $\Delta(t, y)$ returns a value in the range $[0, y]$, such that the probability of $\Delta(t, y)$ being close to 0 increases as $t$ increases (i.e. generation number). This property causes this operator to search the space uniformly initially (when is small), and very locally at later stages. In the experiments reported in [130], the following function $\Delta(t, y)$ was used:

$$\Delta(t, y) = y \cdot r \cdot (1 - \frac{t}{T})^b$$

Where $(r)$ is a random number from the range $[0, 1]$, $(T)$ is the maximal generation number and $(b)$ is a system parameter determining the non-uniformity degree.

Michalewicz [167] proposed random (uniform) mutation. In this type of mutation, a selected gene $v_k$ is substituted with a random value $r_k$ between its lower and upper bounds $r_k \in [l_k, u_k]$. A special type of uniform mutation is “boundary mutation” [168] in which a gene $v_k$ with equal probability is substituted by either its lower bound $l_k$ or
The boundary mutation usefully applied to type of problems in which their results on the boundary or very close to the boundary.

\[ v_k' = \begin{cases} l_k & \text{if random digit is 0} \\ u_k & \text{if random digit is 1} \end{cases} \]

Mäkinen with his co-researcher [161] first suggested a mutation strategy named “Exponential Mutation” to solve some multidisciplinary shape optimization problems using GA in aerodynamics and electromagnetics. Meittinen and his co-researchers [165] later used this strategy to solve the large set of constrained optimization problem. While on 2007, Deep and Thakur through their work used this mutation and named it Mäkinen, Periaux and Toivanen Mutation (MPTM). In this mutation if the element \( v_k \) of a selected individual \( S = \{v_1, v_2, ..., v_k, ..., v_l\} \) for the current generation with \( v_k \in [l_k, u_k] \) the resulting individual for the next generation will be a vector \( S' = \{v_1, v_2, ..., v_k', ..., v_l\} \) with \( k \in [1, l] \) and \( v_k' = (1 - \hat{t})l_k + \hat{t}u_k \) where:

\[ \hat{t} = \begin{cases} t - t(\frac{t - r}{t})^b & \text{if } r < t \\ t & \text{if } r = t \\ t + (1 - t)(\frac{r - t}{1 - t})^b & \text{if } r > t \end{cases} \]

Where \( t = \frac{v - l_k}{u_k - v} \) and \( r \) be a uniformly distributed random number between 0 and 1.

Munteanu and Lazarescu [179] suggested PCA-mutation. This mutation strategy is based on principal component analysis (PCA) which is familiar statistical technique that has been widely used in compression and data analysis. PCA-mutation is observed to achieve a higher level of population diversity than classical ones like uniform mutation and non-uniform mutation on IIR Filter design problem by Deczky Method [185] that requires a global search strategy.

Deb and Agrawal [58] introduced polynomial mutation (PM) operator which is one of the most widely used mutation operator that successfully applied in solving single and multi-objective optimization problems [57]. Polynomial mutation is based on
polynomial probability distribution instead of normal distribution to produce new offspring closer to their parents and therefore enhance the real-coded genetic algorithm. In polynomial mutation if the element $v_k$ from individual $S = \{v_1, v_2, ..., v_k, ..., v_l\}$ has been selected from current population to be mutated using PM introduces the element $v'_k$ of the individual $S' = \{v_1, v_2, ..., v'_k, ..., v_l\}$ for the population of the next generation where:

$$v'_k = v_k + (u_k - l_k) \cdot \delta_k$$

the parameter $\delta_k$ can be obtained by the polynomial probability distribution:

$$P(\delta) = 0.5 \left( \eta_m + 1 \right) \left( 1 - \delta^{\eta_m} \right)$$

where $\eta_m$ is the polynomial distribution index (non-negative real number), and $r_k$ is a random number in $[0, 1]$.

However, the shape of probability distribution is controlled by the parameter $\eta_m$ and distribution is not dynamically changed with generations.

Wavelet mutation was first introduced by Ling and Leung [150] which is mainly based on wavelet theory. Wavelet mutation ($W$) uses Morlet wavelet as the mother wavelet [51]. The following equation can be used as an example of mother wavelet, and figure2.4-1 shows the Morlet wavelet:

$$\Psi(x) = e^{-x^2/2} \cos(5x)$$
The wavelet mutation operation shows a fine-tuning ability therefore the GA performance used WM is illustrated better than some other real coded GAs performance on a group of benchmark test functions as well as economic load dispatch and tuning an associative-memory neural network. In wavelet mutation, every element of population individuals will have a chance to mutate which is controlled by a mutation probability $P_m \in [0, 1]$ which is defined by the user. For current iterations $(t)$ each element $v_k$ from individual $S = \{v_1, v_2, ..., v_k, ..., v_l\}$ where $1 \leq k \leq l$, a random number $r \in [0, 1]$ will be generated such that if $r \leq P_m$ a mutation operation will take place on that $v_k$ as follow:

$$v'_k = \begin{cases} v_k + \sigma \cdot (u_k - v_k) & \text{if } \sigma > 0 \\ v_k + \sigma \cdot (v_k - l_k) & \text{if } \sigma \leq 0 \end{cases}$$

As been mentioned before $l_k$ and $u_k$ are the lower and upper bounds of $v_k$ respectively and if Morlet wavelet has been selected as the mother wavelet, then:

$$\sigma = \frac{1}{\sqrt{a}} e^{-\frac{(\varphi)^2}{2}} \cos(5 \frac{\varphi}{a})$$

the parameter $\varphi$ is a random number in domain $[-2.5a, 2.5a]$, while parameter $(a)$ can be calculated using the following equation:
\[ a = e^{-\ln(g) \cdot \left(1 - \frac{t}{t_{\text{max}}}\right)^{\xi_{\text{wm}} + 1} / g} \]

where \( \xi_{\text{wm}} \) is the shape parameter of the monotonic increasing function, and \( g \) is the upper limit of the parameter \( a \), \( t_{\text{max}} \) is the total number of iterations.

### A.5 Crossover Operator

The main idea of crossover is to allow that the genetic strings to be mutual from at least two parents in some way to simulate the reproduction action in natural populations. The following is the main and common types of crossover operator.

#### A.5.1 One-Point Crossover

Holland [122] was the first one who suggested the standard and original formulation of recombination which is so called one-point crossover which was previously analysed by DeJong [52]. The one-point crossover mechanism can be described as follow:

1. Two parents \( A, B \) are selected from population.
2. A random position \( P \in [0, l - 1] \) is chosen where \( l \) is individual's length.
3. The genes string of parent \( A \) is copied from location 0 to \( P \) to the offspring genome.
4. The offspring genome is completed with the values from genes string of parent \( B \) from location \( P+1 \) to \( l-1 \).

The original form by Holland was to generate two offspring where the rest of parent \( A \) and \( B \) will form the second child so that by reversing the previous scheme. Figure 2.4-2 illustrates a default form of one-point crossover.
A.5.2 Two-Points Crossover

There is advantage to have more points of crossover operator which is make the problem searching space probably be searched more deeply. As alternative form of one-point crossover, the two-point crossover schema can be described as follows (see figure2.4-3a):

1. Two parents A, B are selected from population.
2. Two locations $P_1$ and $P_2$ are chosen randomly from range $[0, l - 1]$ with each number equally likely.
3. Two new children $C_1, C_2$ are formed from exchanging the alleles from each parent to another, where child $C_1$ will has alleles from parent A and B so that $C_1 = \{A_0^{P_1}, B_{P_1+1}^{P_2}, A_{P_1+1}^{l-1}\}$, while child $C_2$ will be formed from parent A and B by taking the rest of alleles so that $C_2 = \{B_0^{P_1}, A_{P_1+1}^{P_2}, B_{P_1+1}^{l-1}\}$

In fact, some studies represent the two-point crossover schema in different way using ring shape. DeJong [52] first represents each population member is imagined as ring rather than linear strings, it is build up by joining the individual's ends together. Then, a two cut points $P_1, P_2$ will be chosen to make a segment of alleles to be exchanged between parents forming two offspring. Figure2.4-3b illustrates this process. Here, the two-point crossover can be seen as one-point crossover with one of the cut points is considered at the start of the individual string, with the intention that the two-point crossover can make the same performance of one-point crossover. This way of representing the individuals (ring format) in order to apply the two-point crossover may produce extra variety of building blocks (segments) since they can be able to "wrap
around" at the end of the string. Generally, the two-point crossover is classified better than one-point crossover.

![Diagram of two-point crossover](image)

**Figure A-3: Two-Point Crossover; a) Normal representing, b) Ring representing**

### A.5.3 Multi-Points Crossover

The Multi-Point (M-point) Crossover uses (at least) two parents as input producing two offspring. This type of crossover used more than two cut points (m-points) in its schema to generate offspring. The crossover mechanism to produce children and after determining the number of cutting points, can be described as follow:

1. Select the crossover (cutting) points' positions \( P_i : i = 1 \ldots m \) and \( m \in [0, l - 1] \).
2. The first child gets the even segments from the first parent, with the odd segments from the second parent.
3. The second child gets the odd segments from the first parent, along with the even segments from the second parent.

The m-point crossover can be seen as two-point crossover if number of cutting points equal to two i.e. m=2. Such that, the first and third segments from parent A with second segment from parent B will used to form child C1, while the first and third segments from parent B with second segment from parent A will used to form child C2. Figure2.4-3 shows an example for m=4.
A.5.4 Uniform Crossover:

A number of empirical studies have projected the advantage of increasing the crossover points [228]. The uniform crossover can be understood as a logical extension of the multi-point crossover is pointed. The formulation of uniform crossover drastically differs to the other types of crossovers schema. Uniform crossover employees a pre-defined template (crossover mask) to process its form instead of randomly selecting the cut points for crossover. The uniform crossover mask (bit-mask) consists of a series of zeros and ones with length equal to problem dimension $l$. However, this bit-mask is usually built up through an exact 50/50 probability of choosing each single bit to be zero or one. Each allele value in offspring is generated by copying the corresponding allele from one or the other parent chosen according to the generated crossover bit-mask. The crossover formulation to produce new offspring can be described as follow:

1. Two parents $A$, $B$ are chosen from population.
2. For each index of the bit-mask, if the bit value is one then the offspring will get allele value from parent $A$.
3. If the bit value is zero, allele value will be copied from parent $B$ to the offspring.

In fact, it is not so important to generate such a bit-mask in real coding. The simple way is to use a `coin flip' at each location can do the job. However, to understand and analyse the uniform crossover behaviour in theory, the bit-mask explanation is
necessary. Figure 2.4-5 illustrates uniform crossover. This type of crossover allows using recombination operator with more than two parents. The defined schema is vary where a simple method uses bit-mask to state the selection between parent A against B, this process is extended to make selection between a group parents.

A.5.5 Adaptive Crossover:

A different type of crossover it can be found in literature is Adaptive crossover. This crossover in evolutionary algorithm used, as it runs, to enhance process performance by making a decision which form of crossover operator or operators will be the optimal to be used for current generation. This concept can be applied by two different ways, either locally or globally regarding to level of applying crossover operator.

Locally: adaptive crossover operator can be processed after selecting two parents A, B from population then generating one or two children to be added to population for next generation. This child is build up by looking at the parents' allele value for each locus. If the allele values on both parents are equal to one then uniform crossover is used. If they are zero on both side then two-point crossover is chosen to be applied. However, in case of each parent has different allele value, a random crossover then will be chosen to be applied.

Globally: adaptive crossover operator can be globally applied by looking at the entire population of current generation. That will be done through a statistical calculating the average of ones and zeros in all individuals. In case of population
members have more zeros then the two-point crossover operator will be used on the entire population. While in the contrary, the uniform crossover operator will be used for the entire population.

In fact, all genetic operators including crossover and mutation have the capability to control these generated members in global and local manner [228].

A.5.6 Guided Crossover

Guided Crossover (GC) [197] is a different form of the normal crossover with some advantages. The guided crossover is an enhanced type of operator used in genetic algorithms to replace some standard crossover and mutation operation in the direction of improving and guiding the convergence headed to the end of optimization. It is generally designed to enhance the result's value so that the genetic algorithm has a better chance to become closer to the global optima. The guided crossover as originally defined by Rasheed [197] can be described as follow:

1. Select two individuals A, B as parents from the population.
2. The first member A is selected from current population via the normal selection schema.
3. The second member B is selected from current population using the following way: for each individual I in the current population rather than A, the quantity function Mutual_fitness(A, I) is calculated such that:

   \[ \text{Mutual\_Fitness}(A, I) = \frac{(\text{fitness}(A) - \text{fitness}(I))^2}{\text{Euclidean distance}(A, I)^2} \]

   Where B will be considered as follow:

   \[ B = \max(\text{Mutual\_Fitness}(A, I_i)) : i \in [0, \mu] \]

4. After selecting two parents, the nominated members A and B may be swapped in order to make A with the highest fitness among A and B.
5. The guided crossover then will randomly generate the offspring \( C \) taken from the line joining the two candidates from the small region closest the highest fitness parent \( A \) such that:

\[
C = L \times A + (1 - L) \times B
\]

where \( L \) is a uniformly distributed random number taken in the range of \([1 - 0.2x, 1 + x]\) and \( x \) is a function of the number of elapsed iterations \( I_e \) and the total allowed number of iterations \( I_t \) such that:

\[
x = 0.75 \frac{(I_t - I_e)}{I_e} + 0.25
\]

However, this crossover operator can simply be described as it observes all points in searching space that can be generated by joining the randomly selected one parent with all others in the current population. For more details about the guided crossover operator please see [197] and [136].

A.5.7 Gene Pool Crossover

The gene-pool crossover is alternative type of recombination operator was examined in EA literature. Mühlenbein and Voigt [178] introduced a new recombination operator and named it Gene-Pool crossover (GPC). In this formulation, the gene value of the single offspring for a particular location is formed from the allele (multinomial) distribution of the whole population at the same location rather than from two selected parents.

In fact, there are some extra studied recombination operators in evolutionary algorithm literature can be found such that "Ordered Crossover (OC) [93], Partially-Mapped Crossover (PMC) [263], Uniform Partially-Mapped Crossover (UPMC) [43], Cycle Crossover (CC) [183] and Non-Wrapping Ordered Crossover (NWOX) [92].
A.5.8 Fuzzy connective-based crossover operators

This crossover allows establishing different exploration and exploitation degrees and Offspring Selecting Method (OSM) inducing different diversity levels in the population. In order to do so, it needs to develop the following steps.

1. Define genes combination functions.
2. Use these functions to define crossover operators between two chromosomes.
3. Apply the crossover operators to the individuals in the population, establishing the number and type of operators along with the selection mechanism to be used.

**Genes Combination Functions:**

Consider \( c^1_i, c^2_i \in [a_i, b_i] \) two genes to be combined and \( \alpha_i = \min\{c^1_i, c^2_i\} \) and \( \beta_i = \max\{c^1_i, c^2_i\} \). The action interval \([a_i, b_i]\) of these genes can be divided into three intervals \([a_i, \alpha_i]\), \([\alpha_i, \beta_i]\) and \([\beta_i, b_i]\). These intervals bound three regions to which the resultant genes of some combination of the former may belong. Moreover, considering a region \([\alpha'_i, \beta'_i]\) where \( \alpha'_i \leq \alpha_i \) and \( \beta'_i \geq \beta_i \) would seem reasonable. Figure A-6 illustrates it graphically as follow:

![Action interval for a gene](image)

The intervals described above could be classified as exploration or exploitation zones. The interval with both genes being the extreme is an exploitation zone, the two intervals that remain on both sides are exploration zones and the region with extremes \( \alpha'_i \) and \( \beta'_i \) could be considered as a relaxed exploitation zone. The following table shows these considerations which are related to Figure A-6.
Table A-1: Interval Properties

<table>
<thead>
<tr>
<th>Interval</th>
<th>Property</th>
</tr>
</thead>
<tbody>
<tr>
<td>[a, α_i]</td>
<td>exploration</td>
</tr>
<tr>
<td>[α_i, β_i]</td>
<td>exploitation</td>
</tr>
<tr>
<td>[β_i, b_i]</td>
<td>exploration</td>
</tr>
<tr>
<td>[α'_i, β'_i]</td>
<td>relaxed exploitation</td>
</tr>
</tbody>
</table>

With regards to these intervals, it proposes four functions: F, S, M and L defined from [a, b] × [a, b] in [a, b], a, b ∈ R, and which fulfil the following properties:

\[
P_1. \forall \ c, c' \in [a, b] \text{ then } F(c, c') \leq \min\{c, c'\}
\]

\[
P_2. \forall \ c, c' \in [a, b] \text{ then } S(c, c') \geq \max\{c, c'\}
\]

\[
P_3. \forall \ c, c' \in [a, b] \text{ then } \min\{c, c'\} \leq M(c, c') \leq \max\{c, c'\}
\]

\[
P_4. \forall \ c, c' \in [a, b] \text{ then } F(c, c') \leq L(c, c') \leq S(c, c')
\]

P5. F, S, M, and L are monotone and non-decreasing.

Each one of these functions allows combining two genes giving results belonging to each one of the aforementioned intervals. This is shown in the following Figure. Where \( \alpha'_i = F(\alpha_i, \beta_i) \) and \( \beta'_i = S(\alpha_i, \beta_i) \)

![Figure A-7: Genes generated using F, S, M and L](image)

These functions will have different exploration or exploitation properties depending on their range. Although the only zone that seems natural to be considered for obtaining offspring is the exploitation zone, the other two zones may be shown to be
appropriate for introducing diversity in the population. The four functions presented allow all these zones to be covered.

**F, S, M and L Functions Using Fuzzy Connectives**

Using t-norms, t-conorms, averaging functions and generalized compensation operators; by associating a t-norm to \( F \), a t-conorm to \( S \), an averaging operator to \( M \) and a generalized compensation operator to \( L \). In order to do so, it needs to transform the genes, that will be combined, wherefrom the interval \([a, b]\) into \([0, 1]\) and later, the results into \([a, b]\). Complying with a set of fuzzy connectives, \{T, G, P, Q\} set of functions \{F, S, M, L\}, associated with it, is built as described below:

If \( c, c' \in [a, b] \) then

\[
F(c, c') = a + (b-a).T(s, s') \\
S(c, c') = a + (b-a).G(s, s') \\
M(c, c') = a + (b-a).P(s, s') \\
L(c, c') = a + (b-a).Q(s, s')
\]

Where \( s = \frac{c-a}{b-a} \) and \( s' = \frac{c'-a}{b-a} \)

These operators have the properties of being continuous and non-decreasing, and satisfy the respective properties (P1 to P5). Table A-2 details families of fuzzy connectives as follow:

<table>
<thead>
<tr>
<th>Family</th>
<th>t-norms</th>
<th>t-conorms</th>
<th>Avg. functions</th>
<th>Gen. compensation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logical</td>
<td>( T_l(x, y) = \min {x, y} )</td>
<td>( G_l(x, y) = \max {x, y} )</td>
<td>( P_l(x, y) = (1-\bar{\lambda})x + \bar{\lambda}y )</td>
<td>( Q_l = T_l^{1-\lambda} \cdot G_l^\lambda )</td>
</tr>
<tr>
<td>Hamacher</td>
<td>( T_h(x, y) = \frac{xy}{1+(1-x)(1-y)} )</td>
<td>( G_h(x, y) = \frac{x+y-2xy}{1-xy} )</td>
<td>( P_h(x, y) = \frac{1}{y+\lambda(\bar{x}+\bar{x})+\lambda^2x} )</td>
<td>( Q_h(x, y) = P_h(T_h, G_h) )</td>
</tr>
<tr>
<td>Algebraic</td>
<td>( T_a(x, y) = xy )</td>
<td>( G_a(x, y) = x + y - xy )</td>
<td>( P_a(x, y) = x^{\lambda-\lambda} \cdot y^{\lambda} )</td>
<td>( Q_a(x, y) = P_a(T_a, G_a) )</td>
</tr>
</tbody>
</table>
**FCB-crossovers: F-crossover, S-crossover, M-crossover and L-crossover:**

Let us assume that $Q \in \{F, S, M, L\}$, $C_1 = (c_1^1, ..., c_n^1)$ and $C_2 = (c_1^2, ..., c_n^2)$ are two chromosomes that have been selected to apply the FCB crossover operator to them. We can generate the chromosome $H = (h_1, ..., h_n)$ as:

$$h_i = Q(c_i^1, c_i^2), \quad i = 1, ..., n \quad (6)$$

This operator applies the same F, S, M or L function for all the genes in the chromosomes to crossover. For this reason, they will be called F-crossover, S-crossover, M-crossover and L-crossover when the F, S, M and L functions are applied respectively. It should be emphasized how these crossover operators have different properties: the F-crossover and S-crossover operators show exploration, the M-crossover operators show exploitation and the L-crossover operator shows relaxed exploitation.

Using the families of fuzzy connectives in the Table A-3 we can build four families of crossover operators. Each one of them shall be called the same as the related fuzzy connective family:

<table>
<thead>
<tr>
<th>Table A-3: Set of crossover operators</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interval</td>
</tr>
<tr>
<td>$F_1, S_1, M_1, L_4$</td>
</tr>
<tr>
<td>$F_2, S_2, M_2, L_2$</td>
</tr>
<tr>
<td>$F_3, S_3, M_3, L_3$</td>
</tr>
<tr>
<td>$F_4, S_4, M_4, L_4$</td>
</tr>
</tbody>
</table>
In fact, the implementation of genetic algorithm more optimum crossovers can be found such as leading crossover [249], order crossover [158], annular crossover [186], DPS [240] and DSO crossover [72].

A.6 Crossover vs. Mutation

In 1990, Fogel and his college Atmar [76] studied EA and used it in solving problems of linear equations. They reached a thought that crossover operator can be considered as a small value in importance, while mutation operator alone has a big chance in finding good solutions. Latter in 1993, Eshelman and Schafer [71] by the use of schema processing methods disagreed this idea, that crossover has a possibility to be valuable and effective to solve some problems.

A.6.1 Mutation Only EAs Challenge

The following discussion will underline some researches on fitness functions which can be recognised hard to optimize with mutation only, while by adding crossover operator will increase the optimization performance towards finding solutions. The first problem Royal Road (RR) were projected by Mitchell et al. [170], a well-studied problem as set of fitness functions. This set was designed to express that the crossover scheme is very critical in genetic algorithms GA to find optimized solution. Latter in 1994, Jones [137] proposed a full definition of the Royal Roads problem with its elements.

Mitchell with his co-researchers mentioned that for a typical GA without taking hitchhiking effects, the optimization complexity time is expected as \( O(2^k \log \left( \frac{n}{k} \right)) \). Forrest and Mitchell [79, 80], throughout their experimental studies show, somewhat surprisingly, that some random mutation only hill-climber algorithms performance are come over other GAs with crossover. This result encouraged them to propose an open problem in [171]; "Finding a set of function optimization and verify that the genetic algorithms GAs are initially outperform the evolutionary algorithms EAs without using crossover operator”.

Prügel-Bennett [195] tied to a good answer to this theory by proposing a fitness gradient with a series of barriers. His study showed with asymptotic approximations and
experiments that a hill-climber performance is a slightly better by using GAs with no applying of crossover, while it much better when using full GA with crossover operator. The expected optimization time found by Jansen and Wegener [134] of the well-studied problem (1+1) EA on the classic Royal Road function is $O \left( 2^k \left( \frac{n}{k} \right) \log \left( \frac{n}{k} \right) \right)$ where $n$ is the individual size and $k$ is the sub elements string size, with mutation rate equal to $1/n$.

Deb and Goldberg research [59] proposed the concatenated trap functions. This type of fitness function formed from a series of minor fully deceptive functions. The proposed fitness functions make all members in the search space, excluding the optima ones, showing local guidance to make process follows a wrong direction in search space towards the optimal solutions. The authors used a function build up from a number of connected non-overlapping deceptive functions.

However, in reasonable time GAs with mutation only were unable to find optimized solution of the compound functions, while using the crossover operator allows generating sequences with short-order forming high fitness individuals and used these sequences combined to successfully lead process to find optimize solution of the function. This building block scheme [59] is historically fundamental of claiming crossover with highly performances on some fitness functions. Note in case of the trap size is fixed or equal to $n^\varepsilon$ where $\varepsilon \in (0,1)$ then it has been reached a suggestion that the mentioned conclusions are inapplicable.

Although the Royal Road and the concatenated trap functions were both proposed to show and prove the crossover operator is necessary for GAs, the successive proof of theory has failed for the Royal Road functions resulted in experiments used to explain that the RR problems were separable and non-deceptive. These particular issues endorsed the use of EAs with non-crossover and RMHCs to locate optimized solution of RR. Focusing on this concern, Watson and Pollack [250, 251] generated a fitness function with hierarchical scheme and named it HIFF (hierarchical if and only if). The authors build this function in way making all sub-blocks are interrelated, non-separable and being deceptive to the mutation process. EAs with mutation only most likely need exponential time to find an optimized solution for HIFF. Later Dietzfelbinger et al. [61] asymptotically analyzed a recombinitive hill-climber on the HIFF function and demonstrated an expected complexity time is $O(n \log(n))$. 
Jansen and Wegener study [133] proposed JUMP_m,n a fitness function utilizing a mutation with special structure. The main attribute is to make the fitness canyon sorting out the global optimal solutions from the fake optimal ones using m-bit gap of low fitness (m < n). The steady state GA (employing uniform crossover) complexity time to locate optimized solution of JUMP_m,n fitness function is expected as O(n^2 log(n)). However, to achieve a good optimization, a hill-climber steady state GA which allows no fitness decreases, such that (µ+1)-EA, is required to apply at same time m bits mutilation to pass the canyon. The expected complexity time to for such process to find optimized solution is estimated as O(n^m), which lead to a clear difference in the time of optimization.

In [134] Jansen and Wegener in their study proposed that the (1+1)-EA which is well known evolutionary algorithm, can successfully optimize the classic Royal Road problem with expected optimization time O\left( 2^k \left( \frac{n}{k} \right) \log\left( \frac{n}{k} \right) \right), where n represents the string size, k represents the sub-elements string size with mutation rate equal to 1/n. This contrasts with the order n/k speed up of the optimized GA's expected time of O\left( 2^k \log\left( \frac{n}{k} \right) \right). As been previously mentioned, experimental results also showed that different random mutation hill climber algorithms (RMHCs) along with the (1+1) EA, can perform better than the normal genetic algorithms GAs.

Furthermore, in [134] Jansen and Wegener proposed the Real Royal Road (RRR) fitness functions by utilizing two types of crossover operator which are one-point crossover as well as uniform crossover. In the work of RRR, two methods were analysed the steady state GA along with equivalent mutation only EA, in intention to find accurate bounds of expected complexity time. The authors' results showed that the steady state GA successfully optimized the Real Royal Road functions in polynomial complexity time, whereas the mutation only EAs take to find optimum solution exponential expected time. Their work employed a various population size S_n which gives a chance to be different at a rate rather than the constant individual length n.

Other researchers latter, Storch and Wegener [233], stated a new form of Real Royal Roads fitness function using both one-point and uniform crossover but for fixed population size. The population has been used of size two that in order to allow a crossover operator performs better with small population. They reach a result that expected complexity time to optimize of the steady state (2 + 1) GA was polynomial time, whereas the (2 + 1) EA expected optimization time was exponential.
The Real Royal Road functions which are formed from a set of artificial functions are fall in a valid critique as they were constructed to work on a narrow challenge. More researches have been made on other natural fitness functions. Fischer and Wegener [73] stated that an advanced GA can perform much better for one-point and uniform crossover than the EA. However, the experimental results are mixed for a standard GA. If \( \lambda \) is carefully chosen, the \((1 + \lambda)\) EA can perform better than the typical GAs. In [234], Sudholt practical results on the Ising problem shows for GA the expected complexity time was polynomial while it expected exponential time for the EA.

Horn et al. [125] suggested a Hamming Path with one-bit mutation operator, which latter named as the Long Path. Horn et al. [125] and latter Rudolph [208] experimental results showed that the expected time to optimize the well known hill-climbing genetic algorithms which searching a one-bit neighbourhood using the long path of length \( n \) is exponential. Afterwards, Rudolph confirmed that the expected waiting time of \((1 + 1)\) EA which changes multiple bits values in one search step to optimize long paths is \( O(n^3) \) time. In [85] Garnier and Kallel statistical study showed that the first hitting of Rudolph's method expected times are differ in the long path of size \( k \). The expected waiting time of simple non-crossover \((1 + 1)\) EA for any chosen \( k \) that is various relative to the size of bit string \( n \) is exponential.

In [206], Rosenblueth with his co-researcher Stephens have explained an analysis scheme to help in justifying the effectiveness of recombination operator for a certain fitness function. This scheme mainly depends on two types of methods the Walsh basis [248] methods and the building block basis (BBB) methods of Stephen [231, 232]. The authors in [206] have tested a pair of functions, the first one is counting-ones (CO) with no epistasis while the second one is the needle-in-a-haystack (NEEDLE) with maximal epistasis. Their work explained that the crossover operator can help to optimize the CO function, while it was not for the NEEDLE function. In fact, any comparison work in terms of genotype distribution will reach the result that it very difficult to notice benefit or harm of the crossover operator. However, the distinction illuminates clearly when using the methodology of building block basis or the Walsh one. Latter, Burjorjee [33] proposed a new type of problems (the biological kind!) which can be significant to computational genetics. He explained the genetic algorithm efficiency on these problems and demonstrated an alternative view of describing the GAs performance and efficiency on them.
\hspace{1cm}

\textit{A.6.2 Crossover Only EAs Challenge}

The problems which are hard to be optimized using crossover operator may be optimized with mutation only algorithms. However they are not widely common in literature of evolutionary and genetic algorithms, they are not totally new. In [193] the authors designed a new fitness function called OneMix (see figure 2.5-1) and they experimentally proved that it is deceptive to crossover operator. The function landscape has a very special saw-tooth pattern accompanied with a distinctive level of frequency oscillation in its shape making it alternately moves from high to low fitness, on the contrary, on first side of the unitation landscape to reach the global optimum. While the second side of the searching space involves a smooth hill climbing towards a local optimum. The saw-tooth side is similar to a long 2 path function and needs two-bits mutation flipping to switch to the low fitness side on the landscape. Following is the formal definition of OneMix function:

\[
\text{OneMix}(x) = \begin{cases} 
(1 + a) \left( \frac{l}{2} - x \right) + \frac{l}{2} & \text{if } x \text{ is even and } x < \frac{l}{2} \\
 x & \text{otherwise}
\end{cases}
\]

Poli et al. in [193] proposed a new GAs theory, the low-pass filter, and discussed that the crossover only has no chance to distinguish the fitness changing with high frequency. Consequently they defined that the crossover operator over a large number of populations works like a low-pass filter. The low-pass filter has the capability to distinguish the diversities between low fitness area and high fitness one of the Poli et al. pattern, therefore GA deception through distinguishing a slope with a extra lower gradient than the gradient formed just from the high-fitness long 2 path.

The authors studied the infinite population model function and evaluate this model through run-time tests. The results showed that the effects of low-pass filter for all genetic operators are also existed with crossover dominating of most effects. In fact, Poli with his co-researchers in [193] did no attempt to use any run-time analysis methods to show any clear boundaries between the GA and EA on the OneMix fitness function.

The author in [203] state that the OneMix results detailed in Poli et al. [193] have a great chance to be truly proven by means of run-time analysis method. Moreover he considers that the fitness space of the saw-tooth pattern will not successfully deceives a
GA with small population size, for instance the (2+1) GA. Additionally he clarifies that depending Poli et al. believes that if a trivial diversity methods were added with guaranteeing that all population's members in the fitness landscape have a good distance between each other, then a small population GA will optimize the OneMix fitness function in polynomial expected time with an exponentially low failure rate.

Figure A-8: Poli et al. OneMix fitness function with low pass filter, \( l=100, a=0.6 \)

A.7 Operators & Attributes Control

A main key issue for researchers is how to tune and control a range of EA's parameters and operators. Some basic ways suggest to previously setting all parameters, while the majority of researchers prefer to carry out a set of runs to statically attempt to setup good settings of the tuned parameters. A modern way more willingly than those have projected and employed adaptive scheme or parameter-less EAs.

A simple and common example of using such a mechanism is mutation adaptation. A basic technique is to adjust the mutation rate regarding to a deterministic schedule. A further group of techniques applying adaptation on more than one parameter a time such the process mutation, population size, and crossover rates by mining algorithm progress or heuristic of quality. The studies are widely existed and a reasonable set of schemes are assessed [65, 66, 226]. However, some practitioners
employing a different technique using a fuzzy logic rule systems, this technique is also reviewed via Richter and Peak study in [204].

One more common studied method in literature is diversity control of the population size. These schemes maintained through two attributes, explicit and implicit. These methods try to utilize reproduction wisdom along with genetic knowledge on the meaning of healthy populations with importance of genetic diversity. The Implicit methods covering some schemes like incest prohibition [45] and some other prohibited mating techniques.

A further formulation in literature is to employ sub-populations to mate evolution process within an isolated population with controlled migration scheme of population members. This will work towards making population diversity included through delaying any probable population dominance via a set of good individuals and allow using a semi-independent evolution direction. The cellular genetic algorithms by Whitley [257] can be projected as a typical example of combining a restrictive mating method together with sub-populations. In such case, all members are distributed on a wide net of virtual points isolated from the search space. These members will only be able to mate among neighbouring solutions. This process will replace one parent in the net by new offspring solutions. Once more, desired target is to retard the highly fitted members spreading and as a result implicitly guaranteeing diversity.

The diversity manipulating schemes aims straight to manage population diversity in active way. Basic mechanisms contain phenotype and genotype duplicate inhibition. Genotype duplicate inhibition is applying a rule on population preventing it from holding any multiple copy of any individual. This restriction is very close to some kinds of bit-string of the genomes similarity behaviour. From other side, the phenotype behaviour is very similar to restriction used through a similarity metric across the fitness function value or any other solution member.

However, fitness sharing and deterministic crowding [159] can be consider more complex than diversity manipulating scheme. The author stated that, a genotype similarity measuring function is utilized to assign the reproductive fitness of new individuals by reducing individual fitness according to similarity assessment with other members. The philosophy behind this scheme is to reduce the chance of individuals being recognised genetically similar of being selected for reproduction, whilst increase it for distinct member of similar fitness. The deterministic crowding scheme as described in [159] suggested a kind of competition (the fitter win) between parents and
their offspring. If the offspring is fitter than its parent, it will replace that parent in the next population.

However, the above summarized successful measuring schemes may be poor in measuring a-priori for a certain fitness landscape. They may be generally distinct in such landscapes which manipulating several dispersed sub-optima or others suffering from various 'barriers and basins' straight the way through out global optima. While some are intuitive to a certain extent which make no possibility to firmly say that they must be used all the time. Yet, as a matter of course, a various forms of algorithm's improvements and developments the researchers should attempt to always work towards enhance the quality of solution. From other side, they must be very careful to make a big chunk of developing or modifications of any EA key preferences, where the changing of problem nature being solved may conquer the assumptions applied by the improvements and optional operators.

In fact, several studies have been done trying to find appropriate selection of EA operators and their associated parameters which will process across a variety type of problems. DeJong [52] introduce the early key study which characterized a collection of test functions and projected a group of parameters which it was wished in order to work well over a wide range of problem types. Though, further researches like Grefenstette [102] used a “meta-ga” in trying to identify a typical values or using comprehensive testing such as Schaffer et al. [212] found dissimilar conclusions. Meanwhile, the theoretical analysis of Goldberg [94] work on finding the optimal population sizes established to formalise the clear point of conclusion that the value of population sizes could be dependably determined relies on the size of the used search space.

However, letter studies introduced a collection of new operators proposed, some of them, like Uniform Crossover by Syswerda [238], used the Schema Theory revaluation and led to the concentrating on two considerable concepts. The first one is Crossover Bias introduced by Eshelman [70], which illustrates the variety ways in that the p.d.f. can arise from different crossover operators upholding hyper planes of high-level of estimated fitness throughout recombination processes, as a determination of their order and length. This reflects the performance and usability of the p.d.f. on the landscape caused by the recombination operator which’s generated by the problem encoding. DeJong and Spears [54, 55, 230] researches help to approve these findings by
using extra proper analysis on the relative properties of different recombination mechanisms.

The second concept, the Safety Ratios (ratio of probability), was introduced by Schaffer and Eshelman [211] in which that the use of reproductive operators produce a new point that would be healthier than its parent(s) to the probability that it is worse. Practically, the Safety Ratios were shown to be changed for the different reproductive operators, and changed over time as well. Once more these can demonstrate harmonise between the p.d.f.s caused by reproductive operators on the current population to the searching landscape’s fitness outlines.

This can be illustrated by employing a simple case like the OneMax function as a thought experiment:

$$f(a) = \sum_{i=0}^{1} a_i$$

In the first initial step and after population has randomly generated, a proportion of fifty percent of the genes will be with the non-optimal fitness value, and so on almost fifty percent of all mutations will be doing well. Though, during the algorithm search progresses, the population fitness average becomes better and the amount of genes with the optimal fitness value will be increased, and that will make the chance of a randomly distributed mutation be beneficially decreased. Bäck work [16, 17] validated this issue, in where for a mutation probability $P_m$, a correct form of the Safety Ratio was produced. Unfortunately, this could not be solved analytically, but it was optimised numerically and noticed to be fitted by a curve of the following form:

$$P_{opt} \approx \frac{1}{2 (f(a) - 1) - 1}$$

This can be obviously seen as a time-dependant process with the existence of any fitness dependent selection pressure.

In interest to combine interactions between other Evolutionary Algorithm societies that were previously working on adaptive operators, such as Evolutionary Strategies of Rechenberg (1+1) [198], and Schwefel ($\lambda \mu$) [216, 217], gave a better
awareness in the opportunity of developing algorithms which can make one or more of their parameters or operators be able to adapt in order to equate the p.d.f. caused by the algorithm to the searching landscape.

In order to formalise what was mentioned above, this describes the possibility of allowing time variation of the GA functions: M, R and U, the parameter set d, and the variables m, and l. It is very necessary in this case to use a function or updating form to state the rule of transformation $X_t \rightarrow X_{t+1}$ where X is the algorithm’s main aspect that being adapted.

Once algorithm prepared to be adaptive, it will face an increased range of jobs, in which include searching the problem space as well as the space of all available variants of the essential algorithm. In account of the range of the possible changes and the learning algorithm nature, this searching job may be restricted, as it may takes variant forms. It may differ from the simple time dependant decrease in the value of a single parameter according to some fixed rule (e.g Fogarty [75]) to a complex path which potentially occupies any position in the hyper space defined by variations in R, M and $\delta$, and which is wholly governed by self-adaptation and the updating process (e.g. Smith and Fogarty [225]).

In case of knowing the accurate nature of the searching landscape, it can be probable to track the optimal route through algorithmic process for a given problem. Unfortunately this may not normally to be the accurate situation, since practically it is essential to supply form of learning technique to steer the algorithm trajectory. Certainly it will not be a trouble free, even it aimed to be though, but the search process becomes more powerful than another process maintaining a certain group of operators and parameters.

Wolpert and Macready [258] stated that almost of all problem types, the entire non-revisiting algorithms will show identical effort. Regarding for certain set of parameter & operator, there can be some problems for which they are optimal, but of course there must be other types of problems for which the performance is so weak. The purpose of the exploiting of adaptive operators and parameters within genetic algorithms is to develop a range of algorithms which demonstrate high performance so that they will be trustable to be used as optimisers on other problems.

A.8 Metaheuristic Algorithms & EAs
A metaheuristic can be defined as a group of concepts designed to allow utilizing a set of heuristic methods to tackle a large set of complex problems. This is particularly applied for the several real-world problems that are naturally combinatorial where other optimization schemes show a failure to be either efficient or effective. In other words, metaheuristic algorithms have come to be classified as a general optimization scheme to be used on different types of deterministic combinatorial problems for optimization, taking into account the few amendments and adaptations to be applicable for a different class of problems [11, 81, 182]. The metaheuristic algorithms are mainly applicable to locate good heuristic solutions for difficult optimization problems which holding several local optima and weak inherent structure to direct the search process. The scheme approach to find good solution of such problems is done by initially obtaining one solution or a group of solutions, and subsequently working towards enhancing the search lighted by certain principle or a set of principles.

The evolutionary algorithms EAs and genetic algorithms GAs can be classified as certain type of metaheuristic optimization algorithms. The metaheuristic algorithms family contains type of algorithms such that, Simulated Annealing SA, Ant Colony algorithms ACA, Metropolis-Hastings algorithm MHA, differential evolution DE, Tabu Search TS and other invented algorithms. An excellent introduction and general reference provided by Glover and Kochenberger [89] and Luke [155] to many of the most popular metaheuristic algorithms.

A.9 Simulated Annealing and Metropolis Algorithms

In [64] Droste et al. demonstrated that the algorithm (1 + 1) EA has the capability be amended to look like Metropolis algorithms and showed deferent convergence and complexity results. Following is the (1 + 1) EA combined with selection scheme derived from Metropolis algorithm. The selection scheme is represented by the function $\alpha (t) : \mathbb{N} \rightarrow [1, \infty)$. In case function $\alpha (t)$ is chosen to be constant, the (1 + 1) EA algorithm becomes exactly same to the Metropolis one. However, once this function selected to be non-constant then the (1 + 1) EA algorithm becomes identical to the Simulated Annealing one. Note that the (1 + 1) EA utilizes a Hamming distance in its mutation operator which flips only one bit. The algorithm:

1. Assign $t = 1$,
2. Select individual $x \in \{0, 1\}^n$ uniformly at random.
3. Generate offspring $y$ by flipping one bit of $x$ randomly.
4. Use probability $P_m = \min\{1, \alpha(t)^{f(y) - f(x)}\}$ to assign $x = y$.
5. Increase $t$ by 1.
6. Repeat from line 2.

The study and analysis was limited to symmetric fitness functions and a Markov chain with $n + 1$ status. The symmetric functions which some time named “unitation functions” can be defined as a group of functions which deal only with the number of ones ($\|x\|_1$) on the individuals’ bits string. This representation help in reducing the different state space of the function. In [64], The function Valley is defined as an instance of symmetric fitness function where Valley($x$) $\in \{0, 1\}^n \Rightarrow \mathbb{R}$ : $n$ is even, the function graph comes as a result of previous algorithm.

$$\text{Valley}(x) = \begin{cases} \frac{n}{2} - \|x\|_1 & \text{if } \|x\|_1 \leq \frac{n}{2} \\ 7n^2 \ln(n) - \frac{n}{2} + \|x\|_1 & \text{if } \|x\|_1 > \frac{n}{2} \end{cases}$$

The author in [203] has studied this function and found that the expected time of previous algorithm to optimize Valley fitness function for the first time with constant $\alpha(t)$ is given by:

$$\Omega \left( \left( \frac{n}{4} \right)^n + \left( \frac{1}{\alpha} + 1 \right)^n \right) = \Omega(1.179)^n \text{ where } \exists \alpha \in [1, \infty)$$

Furthermore, he stated that with probability of $P = 1 - O(n^{-n})$ with selection schedule $\alpha(t) = 1 + \frac{1}{s(n)}$ the expected number of steps required to make the algorithms find the global optimum (maximum) of Valley function for the first time is $1 - O(n s(n))$ for any polynomial $s$ with $s(n) \geq 2en^4 \log n$. 

186
Droste with his co-researchers [64] continued in presenting another copy of the (1 + 1) EA algorithm. In this instance they used a cyclic mutation rate scheme then they success to verify that the algorithm can be able to search and locate the optimized solution for a particular function’s type in expected polynomial complexity time. In addition, they managed to demonstrate that the evolutionary algorithm with standard version may take highly polynomial time to find global optimum, and they illustrated in general the best worst-case behaviour for selected problems. Following is (1 + 1) EA algorithm using Dynamic-Cyclic Mutation operator.

1. Select individual $x \in \{0, 1\}^n$ uniformly at random.
2. Set mutation rate $P_m = 1/n$.
3. Generate offspring $y$ by independently mutate each bit string of $x$ with $P_m$.
4. If $f(y) \geq f(x)$ assign $x = y$.
5. Set $P_m = 2P_m$, if $P_m > 0.5$ assign $P_m = 0.5$
6. Repeat from number 3.

In [339] Jansen and Wegener study, demonstrated another vision the (1 + 1) EA algorithm with employing a dynamic mutation scheme. In order to ease the algorithm analysis, the authors introduced a reasonably hard fitness function with jumping fitness barrier and utilizing a particular Hamming path in the binary space. They proved that the expected time to optimize the standard algorithm with an optimal mutation rate is $O(n^{2.361})$, while the standard mutation rate of $P_m = 1/n$ expected to be a super polynomial step. The algorithm with dynamic mutation rate requires $O(n^2 \ln n)$ time. This study and its approach toward designing fitness functions encourages for more
researches and papers for constructing new functions to examine the changes may occur in evolutionary algorithm with crossover only and mutation only.

A further related work, Jansen and Wegener [131] examined Metropolis (MA) and simulated annealing (SA) along with the (1+1) EA algorithm. The authors demonstrated that the expected time for SA and MA to optimize a type of fitness function that affected with the number of ones in bit string representation is $O(n^2 \ln n)$ steps. In addition, they stated that the SA and EA expected time to optimize fitness functions may differ exponentially and constructed formulations on the cooling methods for different time barriers. They gave some example in their paper [131] showing that the EA and SA/MA performance is vary regarding to the outcome of the constraint that allowing to flip one bit only of string for any generation. In case the fitness function deals with more than one bit string will make the SA/MA algorithms expect an aggravation status towards making progress traverse the barrier. However, EA has ability to flip more than one bit a time and consequently traverse the barrier earlier.

In [132] Jansen and Wegener they treat the question of fitness highlands, they noted that the classic version of (1+1)EA algorithm will only deal with offspring of fitness value are better than or, in the worst case, equal to their parents. What of the slight changes requiring a strict fitness superiority? This will easily show that it may cause a fitness functions problem which named fitness plateaus. These are Hamming neighbour region that at certain fitness values the algorithm be required to navigate to locate a new area with better fitness value. The authors considered the different direction and tack and characterized a new type of fitness functions and named SPTn. in this family of fitness function, the EA with its classic version expected an exponential time to find the global optimum, but with algorithm does not allowing neutral moves can be optimized in expected $O(n^3 \ln n)$ time. Once more, these fitness functions include path to control and guide the algorithms and involve traps which make the classic version fall in. The reached result was developed to deal with more than one path and trap among the same differentiation holding.

Wegener and Witt [253] and through using of quadratic pseudo-boolean functions, they demonstrated a new analysis of the (1+1)EA with multistart variants. These functions are polynomials of degree 2, which can be defined as a type of fitness functions which include NP-hard problems. A multistart variant is a scheme can be defined as populations of isolated members where every single member generates offspring can be only replaced with this offspring. The authors in [253] presented some
instances of this functions which give a good overview of failure and success of the two algorithms. A subclass of quadratic functions which only using positive operators, the (1+1) EA can find the global optimum of them in expected polynomial $O(n^3)$ time, while it will takes expected exponential time to optimize the functions with the negative operators. On the other hand, the multi-start variant algorithm was be able to locate the global optimum of this subclass with expected time of $O(P(n)^n \ln n)$ where $P(n) = \omega(n \ln n)$. Another type of fitness function was presented as a part of quadratic group was the TRAPn function. The reached result showed that the (1+1)EA and multistart variant will require exponential time to optimize TRAPn fitness function or exponential restarts to achieve the global optimum. As a final result of the authors stated that the (1+1) EA will not require exponential time to successfully optimize if the separable quadratic function which can be defined as a function which can be formed as a summation of quadratic functions on smaller domains. The (1+1) EA algorithm requires a population with the basic manner which make some researchers consider it belongs to hill-climber algorithms than of the GA, Note that EA not including a population $p > 1$ will not be able employ any form of the crossover operator.
Here we demonstrate some practical results of the projected models to optimise the test functions including the CEC’2011 benchmark test problems.

Table B-1: Benchmark test function with normal version

<table>
<thead>
<tr>
<th>Function</th>
<th>Formulation</th>
</tr>
</thead>
</table>
| Sphere   | $f_{Sph}(x) = \sum_{i=1}^{n} x_i^2$  
-5.12 ≤ $x_i$ ≤ 5.12, $i = 1, ..., n$  
$\min(f_{Sph}(x)) = 0$; $x_i = 0$ |
| Rosenbrock| $f_{Ros}(x) = \sum_{i=1}^{n-1} [100(x_{i+1} - x_i^2)^2 + (1 - x_i)^2]$  
-2.048 ≤ $x_i$ ≤ 2.048; $i = 1, ..., n$  
$\min(f_{Ros}(x)) = 0$; $x_i = 1$ |
| Schwefel | $f_{Sch}(x) = \sum_{i=1}^{n} -x_i \sin(\sqrt{|x_i|})$  
-500 ≤ $x_i$ ≤ 500; $i = 1, ..., n$  
$\min(f_{Sch}(x)) = -418.9829n$; $x_i = 420.9687$ |
| Rastrigin | $f_{Ras}(x) = 10n + \sum_{i=1}^{n} [x_i^2 - 10 \cos(2\pi x_i)]$  
-5.12 ≤ $x_i$ ≤ 5.12; $i = 1, ..., n$  
$\min(f_{Ras}(x)) = 0$; $x_i = 0$ |
| Griewangk | $f_{Gri}(x) = \frac{1}{4000} \sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} \cos(\frac{x_i}{\sqrt{i}}) + 1$  
-600 ≤ $x_i$ ≤ 600; $i = 1, ..., n$ |
<table>
<thead>
<tr>
<th>Function</th>
<th>Formulation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>S-Sphere</strong>&lt;br&gt;$f_{S-Sph}$</td>
<td>$f_{Sph}(x) = \sum_{i=1}^{n} x_i^2 + f_{bise}$, $z = (x - o)$&lt;br&gt;$x = [x_1, x_2, \ldots, x_n]$, $o = [\alpha_1, \alpha_2, \ldots, \alpha_n]$&lt;br&gt;$\text{min}(f_{Sph}(x)) = -450$; $x_i = a_i$</td>
</tr>
<tr>
<td><strong>S-Rosenbrock</strong>&lt;br&gt;$f_{S-Ros}$</td>
<td>$f_{Ros}(x) = \sum_{i=1}^{n-1} [100(z_{i+1} - z_i^2)^2 + (1 - z_i)^2] + f_{bise}$, $z = (x - o)$&lt;br&gt;$x = [x_1, x_2, \ldots, x_n]$, $o = [\alpha_1, \alpha_2, \ldots, \alpha_n]$, $f_{bise} = -390$&lt;br&gt;$\text{min}(f_{Ros}(x)) = -390$; $x_i = a_i$</td>
</tr>
<tr>
<td><strong>S-Schwefel</strong>&lt;br&gt;$f_{S-Sch}$</td>
<td>$f_{Sch}(x) = \sum_{i=1}^{n} -z_i \sin \left(\sqrt{</td>
</tr>
<tr>
<td>Function</td>
<td>Formula</td>
</tr>
<tr>
<td>----------</td>
<td>---------</td>
</tr>
<tr>
<td>S-Rastrigin</td>
<td>$f_{S-Ras}(x) = 10n + \sum_{i=1}^{n} [x_i^2 - 10 \cos(2\pi x_i)] + f_{bise}$, $z = (x - o)$</td>
</tr>
<tr>
<td>S-Griewangk</td>
<td>$f_{S-Gri}(x) = \frac{1}{4000} \sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} \cos \left( \frac{x_i}{\sqrt{i}} \right) + 1 + f_{bise}$, $z = (x - o)$</td>
</tr>
<tr>
<td>S-Ackley</td>
<td>$f_{S-Ack}(x) = -20 \exp \left( -0.2 \sqrt{\frac{1}{n} \sum_{i=1}^{n} x_i^2} \right) - \exp \left( \frac{1}{n} \sum_{i=1}^{n} \cos(2\pi x_i) \right) + 20 + b$</td>
</tr>
</tbody>
</table>
Table B-3: The 16 clients’ average execution (ms) time of 20 runs to optimise the normal version of 30D benchmark test problems

<table>
<thead>
<tr>
<th></th>
<th>SGA</th>
<th>$dGAT1$</th>
<th>$dGAT2$</th>
<th>$dGAT3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f_{Sph}$</td>
<td>72515</td>
<td>828</td>
<td>778</td>
<td>808</td>
</tr>
<tr>
<td>$f_{Ros}$</td>
<td>375703</td>
<td>40036</td>
<td>37325</td>
<td>41643</td>
</tr>
<tr>
<td>$f_{Ras}$</td>
<td>35197</td>
<td>730</td>
<td>688</td>
<td>735</td>
</tr>
<tr>
<td>$f_{Sch}$</td>
<td>1963761</td>
<td>12680</td>
<td>10664</td>
<td>13132</td>
</tr>
<tr>
<td>$f_{Gri}$</td>
<td>77949</td>
<td>859</td>
<td>702</td>
<td>756</td>
</tr>
<tr>
<td>$f_{Ack}$</td>
<td>75139</td>
<td>809</td>
<td>677</td>
<td>803</td>
</tr>
</tbody>
</table>

Table B-4: The 16 clients’ average execution time (ms) of 20 runs to optimise the shifted version of 30D benchmark test problems

<table>
<thead>
<tr>
<th></th>
<th>SGA</th>
<th>$dGAT1$</th>
<th>$dGAT2$</th>
<th>$dGAT3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f_{Sph}$</td>
<td>5142212</td>
<td>51219</td>
<td>46363</td>
<td>49833</td>
</tr>
<tr>
<td>$f_{Ros}$</td>
<td>10119573</td>
<td>750837</td>
<td>551186</td>
<td>718053</td>
</tr>
<tr>
<td>$f_{Ras}$</td>
<td>1435685</td>
<td>56009</td>
<td>43877</td>
<td>55459</td>
</tr>
<tr>
<td>$f_{Sch}$</td>
<td>1350589</td>
<td>48797</td>
<td>43906</td>
<td>50891</td>
</tr>
<tr>
<td>$f_{Gri}$</td>
<td>1350589</td>
<td>40909</td>
<td>29812</td>
<td>58575</td>
</tr>
<tr>
<td>$f_{Ack}$</td>
<td>1296606</td>
<td>45273</td>
<td>30890</td>
<td>51809</td>
</tr>
</tbody>
</table>
Table B-5: The 8 clients’ average execution time (ms) of 20 runs to optimise the normal version of 30D benchmark test problems

<table>
<thead>
<tr>
<th></th>
<th>SGA</th>
<th>dGAT1</th>
<th>dGAT2</th>
<th>dGAT3</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f_{Sph}$</td>
<td>72515</td>
<td>2261</td>
<td>1565</td>
<td>1802</td>
</tr>
<tr>
<td>$f_{Ros}$</td>
<td>375703</td>
<td>246484</td>
<td>193983</td>
<td>313005</td>
</tr>
<tr>
<td>$f_{Ras}$</td>
<td>35197</td>
<td>1562</td>
<td>1562</td>
<td>2172</td>
</tr>
<tr>
<td>$f_{Sch}$</td>
<td>1963761</td>
<td>34059</td>
<td>25316</td>
<td>49369</td>
</tr>
<tr>
<td>$f_{Gri}$</td>
<td>77949</td>
<td>2233</td>
<td>1542</td>
<td>1856</td>
</tr>
<tr>
<td>$f_{Ack}$</td>
<td>75139</td>
<td>2031</td>
<td>2000</td>
<td>2359</td>
</tr>
</tbody>
</table>

Table B-6: The 8 clients’ average execution time (ms) of 20 runs to optimise the shifted version of 30D benchmark test problems

<table>
<thead>
<tr>
<th></th>
<th>SGA</th>
<th>dGAT1</th>
<th>dGAT2</th>
<th>dGAT3</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f_{S-Sph}$</td>
<td>5142212</td>
<td>138503</td>
<td>112712</td>
<td>137969</td>
</tr>
<tr>
<td>$f_{S-Ros}$</td>
<td>10119573</td>
<td>1762936</td>
<td>1396264</td>
<td>1795062</td>
</tr>
<tr>
<td>$f_{S-Ras}$</td>
<td>1435685</td>
<td>153054</td>
<td>114120</td>
<td>132423</td>
</tr>
<tr>
<td>$f_{S-Sch}$</td>
<td>1350589</td>
<td>528545</td>
<td>426094</td>
<td>713984</td>
</tr>
<tr>
<td>$f_{S-Gri}$</td>
<td>1350589</td>
<td>165531</td>
<td>143793</td>
<td>163831</td>
</tr>
<tr>
<td>$f_{S-Ack}$</td>
<td>1296606</td>
<td>141634</td>
<td>132812</td>
<td>146461</td>
</tr>
</tbody>
</table>
Table B-7: The 16 clients’ average execution time (ms) of 20 runs to optimise the normal version of 50D benchmark test problems

<table>
<thead>
<tr>
<th></th>
<th>SGA</th>
<th>AdGAT1</th>
<th>AdGAT2</th>
<th>AdGAT3</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f_{Sph}$</td>
<td>72515</td>
<td>1201</td>
<td>1123</td>
<td>1159</td>
</tr>
<tr>
<td>$f_{Ros}$</td>
<td>375703</td>
<td>125764</td>
<td>112764</td>
<td>124429</td>
</tr>
<tr>
<td>$f_{Ras}$</td>
<td>35197</td>
<td>1849</td>
<td>1754</td>
<td>1804</td>
</tr>
<tr>
<td>$f_{Sch}$</td>
<td>1963761</td>
<td>21218</td>
<td>19760</td>
<td>20223</td>
</tr>
<tr>
<td>$f_{Gri}$</td>
<td>77949</td>
<td>2013</td>
<td>1864</td>
<td>1912</td>
</tr>
<tr>
<td>$f_{Ack}$</td>
<td>75139</td>
<td>1956</td>
<td>1872</td>
<td>1920</td>
</tr>
</tbody>
</table>

Table B-8: The 16 clients’ average execution time (ms) of 20 runs to optimise the shifted version of 50D benchmark test problems

<table>
<thead>
<tr>
<th></th>
<th>SGA</th>
<th>AdGAT1</th>
<th>AdGAT2</th>
<th>AdGAT3</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f_{S-Sph}$</td>
<td>5142212</td>
<td>132755</td>
<td>130088</td>
<td>137739</td>
</tr>
<tr>
<td>$f_{S-Ros}$</td>
<td>10119573</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$f_{S-Ras}$</td>
<td>1435685</td>
<td>145299</td>
<td>135951</td>
<td>148800</td>
</tr>
<tr>
<td>$f_{S-Sch}$</td>
<td>1350589</td>
<td>244302</td>
<td>232311</td>
<td>254307</td>
</tr>
<tr>
<td>$f_{S-Gri}$</td>
<td>1350589</td>
<td>152499</td>
<td>156228</td>
<td>146218</td>
</tr>
<tr>
<td>$f_{S-Ack}$</td>
<td>1296606</td>
<td>139086</td>
<td>127955</td>
<td>129098</td>
</tr>
</tbody>
</table>
Table B-9: The 16 clients’ average execution time (ms) of 20 runs to optimise the normal version of 100D benchmark test problems

<table>
<thead>
<tr>
<th></th>
<th>SGA</th>
<th>AdGAT1</th>
<th>AdGAT2</th>
<th>AdGAT3</th>
</tr>
</thead>
<tbody>
<tr>
<td>f_{Sph}</td>
<td>72515</td>
<td>2880</td>
<td>2604</td>
<td>2720</td>
</tr>
<tr>
<td>f_{Ros}</td>
<td>375703</td>
<td>119764</td>
<td>111970</td>
<td>117614</td>
</tr>
<tr>
<td>f_{Ras}</td>
<td>35197</td>
<td>1850</td>
<td>1705</td>
<td>1760</td>
</tr>
<tr>
<td>f_{Sch}</td>
<td>1963761</td>
<td>120920</td>
<td>109906</td>
<td>126214</td>
</tr>
<tr>
<td>f_{Gri}</td>
<td>77949</td>
<td>1510</td>
<td>1326</td>
<td>1400</td>
</tr>
<tr>
<td>f_{Ack}</td>
<td>75139</td>
<td>2280</td>
<td>2094</td>
<td>2230</td>
</tr>
</tbody>
</table>

Table B-10: The 16 clients’ average execution time (ms) of 20 runs to optimise the shifted version of 100D benchmark test problems

<table>
<thead>
<tr>
<th></th>
<th>SGA</th>
<th>AdGAT1</th>
<th>AdGAT2</th>
<th>AdGAT3</th>
</tr>
</thead>
<tbody>
<tr>
<td>f_{Sph}</td>
<td>5142212</td>
<td>451120</td>
<td>435473</td>
<td>490365</td>
</tr>
<tr>
<td>f_{Ros}</td>
<td>10119573</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>f_{Ras}</td>
<td>1435685</td>
<td>445201</td>
<td>423556</td>
<td>461460</td>
</tr>
<tr>
<td>f_{Sch}</td>
<td>1350589</td>
<td>445152</td>
<td>413827</td>
<td>459251</td>
</tr>
<tr>
<td>f_{Gri}</td>
<td>1350589</td>
<td>430215</td>
<td>401206</td>
<td>440951</td>
</tr>
<tr>
<td>f_{Ack}</td>
<td>1296606</td>
<td>475630</td>
<td>450547</td>
<td>498541</td>
</tr>
</tbody>
</table>
Table B-11: The Summary of IEEE-CEC’11 [69] benchmark problems

<table>
<thead>
<tr>
<th>Problem</th>
<th>No. of Dimensions</th>
<th>Constraints</th>
<th>Bounds</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>Parameter Estimation for Frequency-Modulated (FM) Sound Waves</td>
<td>6</td>
<td>Bound constrained</td>
</tr>
<tr>
<td>P2</td>
<td>Lennard-Jones Potential Problem</td>
<td>$3 \times 10 = 30$</td>
<td>Bound constrained</td>
</tr>
<tr>
<td>P3</td>
<td>The Bi functional Catalyst Blend Optimal Control Problem</td>
<td>1</td>
<td>Bound constrained</td>
</tr>
<tr>
<td>P4</td>
<td>Optimal Control of a Non-Linear Stirred Tank Reactor</td>
<td>1</td>
<td>Unconstrained</td>
</tr>
<tr>
<td>P5</td>
<td>Tersoff Potential for model Si (B)</td>
<td>$3 \times 10 = 30$</td>
<td>Bound constrained</td>
</tr>
</tbody>
</table>

$x_i \in \left[ -4 - \frac{1}{4} \left( \frac{i - 4}{3} \right), 4 + \frac{1}{4} \left( \frac{i - 4}{3} \right) \right]$
<table>
<thead>
<tr>
<th>Page</th>
<th>Problem Description</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>P6</td>
<td>Tersoff Potential for model Si (C)</td>
<td>Bound constraints: $x_1, x_2 \in [0,4]$ $x_3 \in [0, \pi]$ $x_i \in \left[-4 - \frac{1}{4} \left\lfloor \frac{i-4}{3} \right\rfloor, 4 + \frac{1}{4} \left\lfloor \frac{i-4}{3} \right\rfloor\right]$</td>
</tr>
<tr>
<td>P7</td>
<td>Spread Spectrum Radar Polly phase Code Design</td>
<td>Bound constraints: All dimensions bound between $[0, 2\pi]$</td>
</tr>
<tr>
<td>P8</td>
<td>Transmission Network Expansion Planning (TNEP)</td>
<td>Equality and inequality constraints: All variables are bounded in the interval $[0, 15]$</td>
</tr>
<tr>
<td>P9</td>
<td>Large Scale Transmission Pricing Problem</td>
<td>Linear Equality Constraints: ${G_{ij}} = \min{P_{gi} - BT_{ij}, P_{dj} - BT_{ij}}$ $\min{G_{ij}} = 0$</td>
</tr>
<tr>
<td>P10</td>
<td>Circular Antenna Array Design Problem</td>
<td>Bound constraints: First six dimensions in $[0.2, 1]$ and next six dimensions $[-180, 180]$</td>
</tr>
<tr>
<td>P11</td>
<td>DED instance 1</td>
<td>Inequality constraints: $P_{\min} = [10, 20, 30, 40, 50]$ $P_{\max} = [75, 125, 175, 250, 300]$</td>
</tr>
<tr>
<td>P12</td>
<td>DED instance 2</td>
<td>Inequality constraints: $P_{\min} = [150, 135, 73, 60, 73, 57, 20, 47, 20]$ $P_{\max} = [470, 460, 340, 300, 243, 160, 130, 120, 80]$</td>
</tr>
<tr>
<td>P13</td>
<td>ELD Instance 1</td>
<td>Inequality constraints: $[100, 500; 50, 200; 80, 300; 50, 150; 50, 200; 10, 120]$</td>
</tr>
<tr>
<td>P14</td>
<td>ELD Instance 2</td>
<td>Inequality constraints: $[0, 680; 0, 360; 0, 360; 60, 180; 60, 180; 180; 60, 180; 60, 180; 60, 180; 40, 120; 40, 120, 55, 120; 55, 120]$</td>
</tr>
<tr>
<td>Page</td>
<td>Instance</td>
<td>Type</td>
</tr>
<tr>
<td>------</td>
<td>--------------</td>
<td>--------</td>
</tr>
<tr>
<td>P15</td>
<td>ELD Instance 3</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P16</td>
<td>ELD Instance 4</td>
<td>40</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P17</td>
<td>ELD Instance 5</td>
<td>140</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P18</td>
<td>Hydrothermal Scheduling Instance 1</td>
<td>96</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P19</td>
<td>Hydrothermal Scheduling Instance 2</td>
<td>96</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P20</td>
<td>Hydrothermal Scheduling Instance 3</td>
<td>96</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P21</td>
<td>Messenger: Spacecraft Trajectory Optimization Problem</td>
<td>26</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P22</td>
<td>Cassini 2: Spacecraft Trajectory Optimization Problem</td>
<td>22</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Table B-12: Model T1 Speedup values of CEC 2011 benchmark test problems

<table>
<thead>
<tr>
<th>Problem</th>
<th>Execution Time</th>
<th>Speed Up</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$dEA$</td>
<td>$DGdEA$</td>
<td>$AdEA$</td>
</tr>
<tr>
<td>P01</td>
<td>1754</td>
<td>1648</td>
<td>1160</td>
</tr>
<tr>
<td>P02</td>
<td>1358</td>
<td>1167</td>
<td>994</td>
</tr>
<tr>
<td>P03</td>
<td>2324</td>
<td>2116</td>
<td>2179</td>
</tr>
<tr>
<td>P04</td>
<td>1916</td>
<td>1813</td>
<td>1301</td>
</tr>
<tr>
<td>P05</td>
<td>26885</td>
<td>23364</td>
<td>15038</td>
</tr>
<tr>
<td>P06</td>
<td>21776</td>
<td>19786</td>
<td>14137</td>
</tr>
<tr>
<td>P07</td>
<td>4590</td>
<td>3866</td>
<td>3256</td>
</tr>
<tr>
<td>P08</td>
<td>1332</td>
<td>1170</td>
<td>1032</td>
</tr>
<tr>
<td>P09</td>
<td>40260</td>
<td>30720</td>
<td>23824</td>
</tr>
<tr>
<td>P10</td>
<td>30763</td>
<td>25560</td>
<td>20053</td>
</tr>
<tr>
<td>P11.1</td>
<td>39894</td>
<td>19910</td>
<td>11518</td>
</tr>
<tr>
<td>P11.2</td>
<td>55846</td>
<td>38650</td>
<td>22108</td>
</tr>
<tr>
<td>P11.3</td>
<td>11990</td>
<td>9932</td>
<td>8402</td>
</tr>
<tr>
<td>P11.4</td>
<td>11893</td>
<td>10242</td>
<td>7646</td>
</tr>
<tr>
<td>P11.5</td>
<td>10784</td>
<td>10230</td>
<td>9682</td>
</tr>
<tr>
<td>P11.6</td>
<td>23989</td>
<td>18904</td>
<td>13144</td>
</tr>
<tr>
<td>P11.7</td>
<td>65922</td>
<td>39192</td>
<td>22275</td>
</tr>
<tr>
<td>P11.8</td>
<td>61132</td>
<td>45960</td>
<td>29957</td>
</tr>
<tr>
<td>P11.9</td>
<td>56149</td>
<td>29587</td>
<td>19421</td>
</tr>
<tr>
<td>P11.10</td>
<td>48125</td>
<td>22520</td>
<td>17269</td>
</tr>
<tr>
<td>P12</td>
<td>26647</td>
<td>22776</td>
<td>19790</td>
</tr>
<tr>
<td>P13</td>
<td>28557</td>
<td>27925</td>
<td>27735</td>
</tr>
<tr>
<td>Average</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Table B-13: Model T2 Speedup values of CEC 2011 benchmark test problems

<table>
<thead>
<tr>
<th>Problem</th>
<th>$dEA$</th>
<th>$DGdEA$</th>
<th>$AdEA$</th>
<th>$AdEA/dEA$</th>
<th>$AdEA/DGdEA$</th>
<th>$DGdEA/dEA$</th>
</tr>
</thead>
<tbody>
<tr>
<td>P01</td>
<td>1244</td>
<td>991</td>
<td>561</td>
<td>2.2175</td>
<td>1.7665</td>
<td>1.2553</td>
</tr>
<tr>
<td>P02</td>
<td>885</td>
<td>626</td>
<td>569</td>
<td>1.5554</td>
<td>1.1002</td>
<td>1.4137</td>
</tr>
<tr>
<td>P03</td>
<td>662</td>
<td>530</td>
<td>503</td>
<td>1.3161</td>
<td>1.0537</td>
<td>1.2491</td>
</tr>
<tr>
<td>P04</td>
<td>704</td>
<td>610</td>
<td>480</td>
<td>1.4667</td>
<td>1.2708</td>
<td>1.1541</td>
</tr>
<tr>
<td>P05</td>
<td>18967</td>
<td>14684</td>
<td>12371</td>
<td>1.5332</td>
<td>1.1870</td>
<td>1.2917</td>
</tr>
<tr>
<td>P06</td>
<td>18216</td>
<td>14025</td>
<td>10587</td>
<td>1.7206</td>
<td>1.3247</td>
<td>1.2988</td>
</tr>
<tr>
<td>P08</td>
<td>791</td>
<td>631</td>
<td>501</td>
<td>1.5788</td>
<td>1.2595</td>
<td>1.2536</td>
</tr>
<tr>
<td>P09</td>
<td>25998</td>
<td>13738</td>
<td>9253</td>
<td>2.8097</td>
<td>1.4847</td>
<td>1.8924</td>
</tr>
<tr>
<td>P10</td>
<td>16684</td>
<td>13069</td>
<td>10150</td>
<td>1.6437</td>
<td>1.2876</td>
<td>1.2766</td>
</tr>
<tr>
<td>P11.1</td>
<td>33114</td>
<td>10775</td>
<td>9506</td>
<td>3.4835</td>
<td>1.1335</td>
<td>3.0732</td>
</tr>
<tr>
<td>P11.2</td>
<td>44082</td>
<td>22508</td>
<td>13741</td>
<td>3.2081</td>
<td>1.6380</td>
<td>1.9585</td>
</tr>
<tr>
<td>P11.3</td>
<td>8403</td>
<td>8176</td>
<td>6794</td>
<td>1.2368</td>
<td>1.2034</td>
<td>1.0278</td>
</tr>
<tr>
<td>P11.4</td>
<td>7926</td>
<td>6630</td>
<td>5204</td>
<td>1.5231</td>
<td>1.2740</td>
<td>1.1955</td>
</tr>
<tr>
<td>P11.5</td>
<td>9039</td>
<td>8127</td>
<td>7004</td>
<td>1.2905</td>
<td>1.1603</td>
<td>1.1122</td>
</tr>
<tr>
<td>P11.6</td>
<td>18139</td>
<td>13174</td>
<td>9063</td>
<td>2.0014</td>
<td>1.4536</td>
<td>1.3769</td>
</tr>
<tr>
<td>P11.7</td>
<td>44362</td>
<td>15796</td>
<td>9657</td>
<td>4.5938</td>
<td>1.6357</td>
<td>2.8084</td>
</tr>
<tr>
<td>P11.8</td>
<td>38828</td>
<td>32044</td>
<td>20331</td>
<td>1.9098</td>
<td>1.5761</td>
<td>1.2117</td>
</tr>
<tr>
<td>P11.9</td>
<td>29008</td>
<td>17088</td>
<td>13025</td>
<td>2.2271</td>
<td>1.3119</td>
<td>1.6976</td>
</tr>
<tr>
<td>P11.10</td>
<td>29169</td>
<td>22057</td>
<td>11409</td>
<td>2.5567</td>
<td>1.9333</td>
<td>1.3224</td>
</tr>
<tr>
<td>P12</td>
<td>17300</td>
<td>15344</td>
<td>15132</td>
<td>1.1433</td>
<td>1.0140</td>
<td>1.1275</td>
</tr>
<tr>
<td>P13</td>
<td>20057</td>
<td>17427</td>
<td>14010</td>
<td>1.4316</td>
<td>1.2439</td>
<td>1.1509</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td></td>
<td></td>
<td></td>
<td><strong>1.9913</strong></td>
<td><strong>1.3385</strong></td>
<td><strong>1.4703</strong></td>
</tr>
</tbody>
</table>
Table B-14: Model T3 Speedup values of CEC 2011 benchmark test problems

<table>
<thead>
<tr>
<th>Problem</th>
<th>dEA</th>
<th>DGdEA</th>
<th>AdEA</th>
<th>AdEA/dEA</th>
<th>AdEA/DGdEA</th>
<th>DGdEA/dEA</th>
</tr>
</thead>
<tbody>
<tr>
<td>P01</td>
<td>1544</td>
<td>1191</td>
<td>761</td>
<td>2.0289</td>
<td>1.5650</td>
<td>1.2964</td>
</tr>
<tr>
<td>P02</td>
<td>1290</td>
<td>840</td>
<td>671</td>
<td>1.9225</td>
<td>1.2519</td>
<td>1.5357</td>
</tr>
<tr>
<td>P03</td>
<td>1371</td>
<td>884</td>
<td>877</td>
<td>1.5633</td>
<td>1.0080</td>
<td>1.5509</td>
</tr>
<tr>
<td>P04</td>
<td>1005</td>
<td>961</td>
<td>528</td>
<td>1.9034</td>
<td>1.8201</td>
<td>1.0458</td>
</tr>
<tr>
<td>P05</td>
<td>24973</td>
<td>21674</td>
<td>13190</td>
<td>1.8933</td>
<td>1.6432</td>
<td>1.1522</td>
</tr>
<tr>
<td>P06</td>
<td>19343</td>
<td>14162</td>
<td>12375</td>
<td>1.5631</td>
<td>1.1444</td>
<td>1.3658</td>
</tr>
<tr>
<td>P07</td>
<td>4077</td>
<td>3282</td>
<td>2512</td>
<td>1.6230</td>
<td>1.3065</td>
<td>1.2422</td>
</tr>
<tr>
<td>P08</td>
<td>1301</td>
<td>813</td>
<td>686</td>
<td>1.8965</td>
<td>1.1851</td>
<td>1.6002</td>
</tr>
<tr>
<td>P09</td>
<td>33046</td>
<td>27037</td>
<td>13996</td>
<td>2.3611</td>
<td>1.9318</td>
<td>1.2223</td>
</tr>
<tr>
<td>P10</td>
<td>26051</td>
<td>19614</td>
<td>13542</td>
<td>1.9237</td>
<td>1.4484</td>
<td>1.3282</td>
</tr>
<tr>
<td>P11.1</td>
<td>36144</td>
<td>11068</td>
<td>9891</td>
<td>3.6542</td>
<td>1.1190</td>
<td>3.2656</td>
</tr>
<tr>
<td>P11.2</td>
<td>48742</td>
<td>24827</td>
<td>14569</td>
<td>3.3456</td>
<td>1.7041</td>
<td>1.9633</td>
</tr>
<tr>
<td>P11.3</td>
<td>10323</td>
<td>9342</td>
<td>7698</td>
<td>1.3410</td>
<td>1.2136</td>
<td>1.1050</td>
</tr>
<tr>
<td>P11.4</td>
<td>8655</td>
<td>8150</td>
<td>7604</td>
<td>1.1382</td>
<td>1.0718</td>
<td>1.0620</td>
</tr>
<tr>
<td>P11.5</td>
<td>10361</td>
<td>9951</td>
<td>8955</td>
<td>1.1570</td>
<td>1.1112</td>
<td>1.0412</td>
</tr>
<tr>
<td>P11.6</td>
<td>18656</td>
<td>15104</td>
<td>10526</td>
<td>1.7724</td>
<td>1.4349</td>
<td>1.2352</td>
</tr>
<tr>
<td>P11.7</td>
<td>59540</td>
<td>36139</td>
<td>21236</td>
<td>2.8037</td>
<td>1.7018</td>
<td>1.6475</td>
</tr>
<tr>
<td>P11.8</td>
<td>44358</td>
<td>39351</td>
<td>27997</td>
<td>1.5844</td>
<td>1.4055</td>
<td>1.1272</td>
</tr>
<tr>
<td>P11.9</td>
<td>43268</td>
<td>23878</td>
<td>16230</td>
<td>2.6659</td>
<td>1.4712</td>
<td>1.8120</td>
</tr>
<tr>
<td>P11.10</td>
<td>42968</td>
<td>20495</td>
<td>16850</td>
<td>2.5500</td>
<td>1.2163</td>
<td>2.0965</td>
</tr>
<tr>
<td>P12</td>
<td>21034</td>
<td>20706</td>
<td>18753</td>
<td>1.1216</td>
<td>1.1041</td>
<td>1.0158</td>
</tr>
<tr>
<td>P13</td>
<td>25775</td>
<td>23234</td>
<td>18211</td>
<td>1.4154</td>
<td>1.2758</td>
<td>1.1094</td>
</tr>
<tr>
<td>Average</td>
<td></td>
<td></td>
<td></td>
<td>1.9649</td>
<td>1.3697</td>
<td>1.4464</td>
</tr>
</tbody>
</table>
BIBLIOGRAPHY


[235] Suganthan P. N., "Testing Evolutionary Algorithms on Real-World Numerical Optimization Problems", Nanyang Technological University, School of Electrical and Electronics Engineering, Singapore,


[252] Weerayuth N. and Chaiyaratana N., "Closed-loop time-optimal path planning using a multi-objective diversity control oriented genetic algorithm", in


