An Adaptive Framework for Internet-based Distributed Genetic Algorithms

A dissertation submitted in partial fulfilment of the requirement for the degree of Doctor of Philosophy

by

Johan Berntsson

M. Sc. Linköping University, Sweden

Supervisor: Dr. Maolin Tang

Associate Supervisors: Dr. Wayne Kelly, Associate Professor Dr. Paul Roe

School of Software Engineering and Data Communications
Faculty of Information Technology
Queensland University of Technology
Brisbane, Australia

Keywords

genetic algorithms, distributed genetic algorithms, Internet computing, floorplanning, adaptation, VLSI
Abstract

Genetic Algorithms (GAs) are search algorithms inspired by genetics and natural selection, and have been used to solve difficult problems in many disciplines, including modelling, control systems and automation. GAs are generally able to find good solutions in reasonable time, however as they are applied to larger and harder problems they are very demanding in terms of computation time and memory. The Internet is the most powerful parallel and distributed computation environment in the world, and the idle cycles and memories of computers on the Internet have been increasingly recognized as a huge untapped source of computation power. By combining Internet computing and GAs, this dissertation provides a framework for Internet-based parallel and distributed GAs that gives scientists and engineers an easy and affordable way to solve hard real world problems.

Developing parallel computation applications on the Internet is quite unlike developing applications in traditional parallel computation environments, such as multiprocessor systems and clusters. This is because the Internet is different in many respects, such as communication overhead, heterogeneity and volatility. To develop an Internet-based GA, we need to understand the implication of these differences. For this purpose, a convergence model for heterogenous and volatile networks is presented and used in experiments that study GA performance and robustness in Internet-like scenarios.

The main outcome of this research is an Internet-based distributed GA framework called G2DGA. G2DGA is an island model distributed GA, which can provide support for big populations needed to solve many real world problems. G2DGA uses a novel hybrid peer-to-peer (P2P) design with island node activity coordinated by supervisor nodes that offer a global overview of the GA search state. Compared to client/server approaches, the P2P architecture improves scalability and fault tolerance by allowing direct communication between the
islands and avoiding single-point-of-failure situations.

One of the defining characteristics of Internet computing is the dynamics and volatility of the environment, and a parallel and distributed GA that does not adapt to its environment cannot use the available resources efficiently. Two novel adaptive methods are investigated. The first method is migration topology adaptation, which uses clustering on elite individuals from each island to rebuild the migration topology. Experiments with the migration topology adapter show that it gives G2DGA better performance than a GA with static migration topology of a similar or larger connectivity level. The second method is population size adaptation, which automatically finds the number of islands and island population sizes needed to solve a given problem efficiently. Experiments on the population size adapter show that it is robust, and compares favourably with the traditional trial-and-error approach in terms of computational effort and solution quality.

The scalability and robustness of G2DGA has been extensively tested in network scenarios of varying volatility and heterogeneity. Experiments with up to 60 computers were conducted in computer laboratories, while more complex network scenarios have been studied in an Internet simulator. In the experiments, G2DGA consistently performs as well as, and usually significantly better than, static distributed GAs and the difference grows larger with increased network instability. The results show that G2DGA, by continuously adjusting the migration policy and the population size, can detect and make efficient use of idle cycles donated over volatile Internet connections.

To demonstrate that G2DGA can be used to implement and solve real world problems, a challenging application in VLSI design was developed and used in the testing of the framework. The application is a multi-layer floorplanner, which uses a novel GA representation and operators based on a slicing structure approach. Its packing quality compares favourably with other multi-layer floorplanners found in the literature.

Internet-based distributed GA research is exciting and important since it enables GAs to be applied to problem areas where resource limitations make traditional approaches unworkable. G2DGA provides a scalable and robust Internet-based distributed GA framework that can serve as a foundation for future work in the field.
Authorship

The work contained in this thesis has not been previously submitted for a degree or diploma at this or any other higher education institution. To the best of my knowledge and belief, the thesis contains no material previously published or written by any other person except where due reference is made.

Signed: ___________________________ Date: 23 July 2006
For Erika and Mayumi.
Contents

Keywords ................................................................. III
Abstract ............................................................... V
Authorship ............................................................. VII
Table of Contents ..................................................... XVI
List of Tables .......................................................... XVIII
List of Figures .......................................................... XXI
List of Algorithms ..................................................... XXIII
List of Abbreviations ................................................ XXV
List of Publications ................................................... XXVII
Acknowledgements .................................................... XXIX

1 Introduction ......................................................... 1
   1.1 Motivation .................................................... 1
   1.2 Research Problem .......................................... 3
   1.3 Major Contributions ....................................... 5
   1.4 Thesis Outline .............................................. 6

2 Overview of Genetic Algorithms ................................. 9
   2.1 Introduction ................................................ 9
   2.2 Simple GAs ............................................... 10
       2.2.1 Theoretical Foundation ............................ 11
   2.3 Advanced GAs ............................................. 13
       2.3.1 Linkage Learning GAs ............................ 13
       2.3.2 Multi-objective GAs ............................ 13
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.4</td>
<td>Parallel GAs</td>
<td>15</td>
</tr>
<tr>
<td>2.4.1</td>
<td>Global Parallel GA</td>
<td>16</td>
</tr>
<tr>
<td>2.4.2</td>
<td>Island Parallel GA</td>
<td>16</td>
</tr>
<tr>
<td>2.4.3</td>
<td>Cellular Parallel GA</td>
<td>17</td>
</tr>
<tr>
<td>2.4.4</td>
<td>Hybrid Parallel GAs</td>
<td>18</td>
</tr>
<tr>
<td>2.5</td>
<td>Summary</td>
<td>19</td>
</tr>
<tr>
<td>3</td>
<td>Internet-based Distributed Genetic Algorithm Review</td>
<td>21</td>
</tr>
<tr>
<td>3.1</td>
<td>Frameworks</td>
<td>21</td>
</tr>
<tr>
<td>3.1.1</td>
<td>Global Model</td>
<td>22</td>
</tr>
<tr>
<td>3.1.2</td>
<td>Island Model</td>
<td>23</td>
</tr>
<tr>
<td>3.1.3</td>
<td>Open Research Problems</td>
<td>27</td>
</tr>
<tr>
<td>3.2</td>
<td>Adaptation</td>
<td>30</td>
</tr>
<tr>
<td>3.2.1</td>
<td>Migration Topology Adaptation</td>
<td>31</td>
</tr>
<tr>
<td>3.2.2</td>
<td>Population Size Adaptation</td>
<td>31</td>
</tr>
<tr>
<td>3.3</td>
<td>Summary</td>
<td>33</td>
</tr>
<tr>
<td>4</td>
<td>Asynchronous Parallel Genetic Algorithm Modelling</td>
<td>35</td>
</tr>
<tr>
<td>4.1</td>
<td>Introduction</td>
<td>35</td>
</tr>
<tr>
<td>4.2</td>
<td>Convergence Model</td>
<td>36</td>
</tr>
<tr>
<td>4.2.1</td>
<td>Sequential GAs</td>
<td>37</td>
</tr>
<tr>
<td>4.2.2</td>
<td>Assumptions and Limitations</td>
<td>38</td>
</tr>
<tr>
<td>4.2.3</td>
<td>Parallel GAs</td>
<td>38</td>
</tr>
<tr>
<td>4.2.4</td>
<td>Emigrants</td>
<td>39</td>
</tr>
<tr>
<td>4.2.5</td>
<td>Immigrants</td>
<td>41</td>
</tr>
<tr>
<td>4.2.6</td>
<td>Selection Differential after Migration</td>
<td>42</td>
</tr>
<tr>
<td>4.3</td>
<td>Failure Model</td>
<td>43</td>
</tr>
<tr>
<td>4.3.1</td>
<td>Synchronous Model Failure</td>
<td>44</td>
</tr>
<tr>
<td>4.3.2</td>
<td>Asynchronous Model Failure</td>
<td>46</td>
</tr>
<tr>
<td>4.4</td>
<td>Verification</td>
<td>47</td>
</tr>
<tr>
<td>4.5</td>
<td>Experiments</td>
<td>48</td>
</tr>
<tr>
<td>4.5.1</td>
<td>Migration Rate</td>
<td>50</td>
</tr>
</tbody>
</table>
4.5.2 Migration Interval ................................................. 51
4.5.3 Fault Tolerance .................................................. 56
4.6 Summary ............................................................... 57

5 G2DGA Framework Design and Implementation 59
5.1 Introduction .......................................................... 59
5.2 G2P2P ................................................................. 61
5.3 Framework Design .................................................. 63
  5.3.1 Console ......................................................... 65
  5.3.2 Island .......................................................... 66
  5.3.3 Supervisor ...................................................... 67
5.4 Framework Implementation and Discussion ...................... 68
  5.4.1 Transparent Exchange of Data Types ....................... 68
  5.4.2 Remote Execution and Tracing ............................... 69
  5.4.3 Security Issues Relating Access ............................ 69
  5.4.4 Process Migration ............................................. 70
  5.4.5 Adaptation to Dynamic Behaviour .......................... 70
  5.4.6 Fault Tolerance .............................................. 70
5.5 Simulator ............................................................ 70
  5.5.1 Overview ....................................................... 71
  5.5.2 Related work .................................................. 72
  5.5.3 Design ........................................................ 72
  5.5.4 Network Configuration File ................................. 73
  5.5.5 Message Passing .............................................. 73
  5.5.6 Background Load ............................................. 75
  5.5.7 Updating the Emulator ..................................... 75
  5.5.8 Simulator Verification ...................................... 76
5.6 Analyser ............................................................. 78
5.7 dotGALib ............................................................. 79
  5.7.1 Programming Model ........................................ 79
  5.7.2 GA .............................................................. 81
  5.7.3 Genome ......................................................... 82

XIII
6 Adaptation of Migration Topologies

6.1 Introduction ................................................. 87
6.2 Migration Topology Adaptation ................................. 88
  6.2.1 Proposed Method ........................................ 89
  6.2.2 Clustering Algorithm ................................. 90
6.3 Experiments .................................................. 92
  6.3.1 Test Problem ........................................ 93
  6.3.2 Connectivity ......................................... 94
  6.3.3 Static Topologies .................................. 95
  6.3.4 Dynamic Topologies ................................ 97
  6.3.5 Other Experiments .................................. 102
  6.3.6 Comparison with Random Topology ................. 103
6.4 Summary ..................................................... 103

7 Adaptation of the Population Size and the Number of Islands 107

7.1 Introduction .................................................. 107
7.2 Proposed Method ............................................. 108
  7.2.1 Competitive Evaluation ................................ 110
  7.2.2 Collaborative Restart ................................ 112
  7.2.3 Termination .......................................... 112
  7.2.4 Putting It All Together ................................ 113
7.3 Experiments .................................................. 113
  7.3.1 Test Problems ........................................ 114
  7.3.2 Manual Population Sizing ............................... 117
  7.3.3 Population Adapter .................................. 118
  7.3.4 Combined Population and Migration Topology Adaptation .... 120
  7.3.5 Dynamic Network Environments ...................... 122
7.4 Summary ..................................................... 123
8 G2DGA Application: VLSI Floorplanning

8.1 Introduction ............................................. 127
8.2 Background ............................................. 128
8.3 Related Work ........................................... 129
8.4 Problem Formulation .................................... 131
  8.4.1 Area Optimisation ................................. 131
  8.4.2 Wirelength Optimisation ......................... 132
8.5 Representation .......................................... 133
  8.5.1 GA Encoding ....................................... 136
  8.5.2 Genome Decoder .................................. 137
  8.5.3 G2DGA Implementation ......................... 138
8.6 Area Minimisation ................................... 139
  8.6.1 Experimental Design ............................. 139
  8.6.2 Experimental Results ............................ 140
8.7 Multi-objective Optimisation ........................ 141
  8.7.1 Combative Accretion Model ...................... 141
  8.7.2 Experimental Design ............................. 142
  8.7.3 Experimental Results ............................ 143
8.8 Summary ................................................ 145

9 Investigation of Scalability and Robustness ........ 147

9.1 Introduction ............................................. 147
9.2 Experimental Design .................................. 148
  9.2.1 Test Problem ...................................... 148
  9.2.2 Speedup Measurements ............................ 149
  9.2.3 Network Environments ............................ 149
  9.2.4 Statistical Methods .............................. 151
  9.2.5 Baseline DGA .................................... 152
9.3 Scalability Study ...................................... 154
  9.3.1 Resource Scalability ............................. 154
  9.3.2 Problem Scalability .............................. 155
9.4 Robustness Study ...................................... 157
List of Tables

3.1 Overview of Internet-based DGA related research ................................... 30
4.1 Definition of neighbourhood for different migration topologies .................. 39
4.2 Selection differential for the four possible selection events. ...................... 43
5.1 Comparison of message sending time for simulator and real networks .......... 76
6.1 GA Parameters ....................................................................................... 94
6.2 Static topologies performance ................................................................... 95
6.3 Single topology adaptation performance .................................................. 98
6.4 Continuous topology adaptation performance ......................................... 98
6.5 Comparison of dynamic topology adapters .............................................. 101
6.6 Summary of benchmark testing on the migration topology adapter .......... 102
7.1 Fitness calculation parameters for the Royal Road function ...................... 114
7.2 Royal Road GA parameters ....................................................................... 115
7.3 F101 GA parameters ................................................................................ 116
7.4 Manual sizing with the F101 benchmark, without migration topology adap-
  tation. ........................................................................................................ 118
7.5 Manual sizing with the F101 benchmark using migration topology adaptation. 118
7.6 Manual sizing with the Royal Road problem, without migration topology adap-
  tation. ........................................................................................................ 119
7.7 Manual sizing with the Royal Road problem using migration topology adap-
  tation. ........................................................................................................ 119
7.8 Population adapter with the F101 benchmark. ............................................ 119
7.9 Population adapter with the Royal Road problem. ................. 119
7.10 Population adapter with the F101 benchmark, combined with migration topology adaptation. ......................... 120
7.11 Population adapter with the Royal Road problem, combined with migration topology adaptation. ......................... 120
7.12 Population adapter comparison using Wilcoxon rank sum test .... 121
7.13 Dynamic network performance ........................................ 124
7.14 Dynamic network performance ........................................ 124
8.1 Area minimisation results with MCNC benchmarks. ............. 140
8.2 Area and wirelength comparison among 3D floorplanners (4 layers). .... 144
9.1 VLSI Floorplanner GA Parameters ................................. 148
9.2 Manual sizing of the ami33 problem on a LAN cluster .......... 153
9.3 Test of statistical significance in manual population sizing of ami33 floorplanner application ................................. 153
9.4 Run-time and performance comparison of ami33 floorplanner application ... 154
9.5 Test of statistical significance in performance comparison of ami33 floorplanner application using uni-ring, fully connected, and adaptive migration topologies ................................. 154
9.6 Comparison of autonomous G2DGA and static DGA on HOM, HET, and VOL network scenarios ............................. 161
9.7 Performance comparison of autonomous G2DGA and static DGAs using Wilcoxon rank sum test ............................. 161

XVIII
List of Figures

2.1 The Simple Genetic Algorithm ........................................... 11
2.2 Parallel GA types ......................................................... 16

3.1 Model/architecture comparison .......................................... 22

4.1 Asynchronous migration example ....................................... 39
4.2 Convergence time with varying migration rates ...................... 45
4.3 Parallel GA survival probabilities with varying fault rates ......... 46
4.4 Verification, fully connected ........................................... 48
4.5 Verification, uni-ring ..................................................... 49
4.6 Verification, fully connected, island 1 fails ......................... 49
4.7 Fully connected, uni- and bi-directional ring topologies .......... 50
4.8 Convergence time for mixed speeds in fully connected topology . 51
4.9 Convergence time for varying migration rates with bi-directional ring topology 52
4.10 Convergence time for varying migration rates with uni-directional ring topology 52
4.11 Convergence time for varying migration rates with fully connected topology . 53
4.12 Convergence time for varying migration intervals with bi-directional ring topology .................................................. 53
4.13 Convergence time for varying migration intervals with uni-directional ring topology .................................................. 54
4.14 Convergence time for varying migration intervals with fully connected topology 54
4.15 Convergence time for bi-directional ring topology when island 1 fails . . . . 55
4.16 Convergence time for uni-directional ring topology when island 1 fails . . . . 55
4.17 The impact of island failure on convergence ........................ 56

XIX
9.1 Resource scalability with *ami33* ................................................. 156
9.2 Speedup for a 64 island *ami33* floorplanner application with varying number
    of computers. .......................................................... 157
9.3 Resource scalability with *ami49* ................................................. 158
9.4 Speedup for a 64 island *ami49* floorplanner application with varying number
    of computers. .......................................................... 159
9.5 Fitness degradation in unstable network environments .......................... 162
## List of Algorithms

1. Convergence model outline .................................................. 40
2. GA Update procedure .......................................................... 67
3. The F8 benchmark problem implemented with dotGALib. .......... 80
4. Laumann’s MOGA test function ................................................. 82
5. Migration topology adapter pseudo-code ............................... 90
6. Pseudo-code for the population adapter ................................. 113
7. Pseudo-code for the splicer algorithm .................................... 135
List of Abbreviations

CORBA Common Object Request Broker Architecture
DCOM Distributed Component Object Model
DGA Distributed Genetic Algorithm
G2 Garden point 2, a distributed cycle-stealing research programme at QUT
GA Genetic Algorithm
G2DGA G2 Distributed Genetic Algorithm
G2P2P G2 Peer to Peer
GUI Graphical User Interface
MDL Minimum Description Length
MOGA Multi-objective Genetic Algorithm
MPI Message Passing Interface, computer communications protocol
P2P Peer to Peer
PGA Parallel Genetic Algorithm
PLAS Programming Languages and Systems, a research group at QUT
RMI Remote Method Invocation, a Java interface for remote procedural calls
RPC Remote Procedure Call
SOAP originally an acronym for Simple Object Access Protocol
SGA Simple (Sequential) Genetic Algorithm
QUT Queensland University of Technology, Brisbane, Australia
VLSI Very Large Scale Integration
List of Publications

Some of the material, ideas, and figures from this dissertation have appeared previously in the following peer-reviewed publications:


Acknowledgements

First of all I want to thank my supervisor Dr. Maolin Tang for his continuous support, advice and encouragement throughout my dissertation work. I also want to thank my associate supervisors Dr. Wayne Kelly and Associate Professor Paul Roe for their comments and suggestions regarding my research.

Furthermore I want to thank Richard Mason for his excellent work on G2P2P which has been used in this project. I would also like to thank Adam Berry for insightful discussions on multi-objective optimisation techniques, as well as the anonymous reviewers of my publications, whose comments have helped me clarify and improve my work. I am also in debt to my patient proofreaders: Peter Nelson and Helen Whittle.

My biggest thanks go to my family, Mayumi and Erika, for their support, encouragement and patience over the years. It has not always been easy, and I know it.
Chapter 1

Introduction

In this dissertation I present a framework designed for Internet-based distributed genetic algorithms (DGA) which can provide scientists and engineers an easy and affordable way of getting access to the processing power needed to apply genetic algorithms (GAs) to a wide range of important real world optimisation problems. The remainder of this chapter elaborates on the motivation, research questions, and main contributions of this research.

1.1 Motivation

Charles Darwin’s scientific theory of evolution, originally introduced in *On the Origin of Species by Natural Selection* [37], starts from the premise that an organism’s traits vary in a non-deterministic way from parent to offspring. If a particular variation makes the offspring better suited to survival or to successful reproduction, that offspring is more likely to reproduce in its turn than those offspring without the variation. Therefore, certain traits are preserved due to the selective advantage they provide to their holders. Eventually, through many iterations of this process, organisms will develop increasingly complex adaptive traits.

The theory of natural selection forms the basis for GAs as well, substituting the organism with suggested solutions to a certain problem, and nature with the algorithm and the problem it is applied to. The struggle for life within GAs takes place inside the computer,
and the designer of the algorithm, not nature, supplies the conditions for survival. Furthermore, GAs apply concepts from modern genetics to represent and manipulate the solutions associated with each individual and its offspring [64, 51]. GAs encode solutions to a specific problem using a chromosome-like data structure and apply recombination operators to produce new individuals. GAs have been successfully used to solve difficult problems in many engineering disciplines, such as bioinformatics and scheduling [52], and they are generally able to find good solutions in reasonable time. However, as they are applied to larger and harder problems they are very demanding in terms of computation time and memory. An effective way of tackling this problem is parallel implementation. GAs have an inherent nature of parallelism, and therefore it is fairly easy to extend a serial GA to a parallel GA.

Most parallel GAs to date have been targeted at multiprocessor machines or cluster environments [25]. More recently, with the success of Internet-based projects such as SETI@home, the idle cycles and memories of computers on the Internet have been increasingly recognised as a huge untapped source of computation power [5], and an attractive environment in which to run GA applications [3]. However, developing parallel computation applications on the Internet is quite different from in traditional parallel computation environments, such as multiprocessor systems, because the Internet is different from those traditional parallel and distributed computation environments in many respects. Firstly, its communication latency is significantly higher and communication bandwidth is narrower than in traditional parallel computation environments. Secondly, the Internet is dynamic and volatile since the number of participating computers and their performance cannot be predicated beforehand and they may withdraw at any time. Thirdly, because of security reasons, participating computers may not be able to communicate with each other directly. Fourthly, participating computers may be heterogeneous. All these issues have to be addressed when developing Internet-based parallel and distributed GAs.

Internet-based distributed GA research is exciting and important since it enables GAs to be applied to problem areas where resource limitations make traditional approaches unworkable. The purpose of this study is to investigate DGA behaviour in an Internet computing environment, and to propose and evaluate a design for Internet-based DGAs.
1.2 Research Problem

The research problem addressed in this work is how to design a DGA that makes efficient use of donated computational resources over the Internet. I decided to limit the investigation to the island model, where the population is divided into sub-populations that run partly isolated with only a limited exchange of individuals between them, as defined by a migration policy. The island model can provide support for big populations needed to solve many big real world problems. It is also the most popular type of DGA in the literature, and the low communication overhead makes the island model fit to cope with bandwidth and latency problems of the Internet.

This research tries to answer the following questions:

- Can GA theory be extended to be used to guide the design of an Internet-based DGA?
- Can an Internet-based DGA framework be designed in such a way that it is scaleable for real world problems?
- Can adaptive methods be applied to an Internet-based DGA framework such that overall performance is improved?

During the course of the investigation these research questions were examined in the manner detailed below:

1. Internet-based DGA Modelling

The initial research question I explored was how to use theory to guide the design of the new DGA. I found that most of the existing literature on parallel GAs cannot be applied to Internet-based DGAs, since they are targeted at parallel computers or cluster computing, where communication capabilities, network topology and resource availability are known and static. I developed a simplified convergence model that can emulate computers and communication links that are volatile and heterogeneous, and used it to analyse DGA performance on various network scenarios. The model makes certain assumptions that may not hold true in real world applications, but it can be
used to compare relative DGA performance, and make some predictions on when a parameter set gives better convergence.

2. Framework Design

Secondly, the design of the framework was considered. The two main models of Internet computing are client/server and peer-to-peer (P2P). The model choice has implications for most other aspects of an Internet-based DGA. One of these aspects is the migration policy, which determines how information flows between the nodes, and has major implications for convergence speed and solution quality. Most current research on Internet-based DGAs use the client/server model, with migration through a coordinating server. This creates a potential bottleneck, with implications for scalability. I choose to use the P2P model, which scales better by using direct communication between the nodes, and improves fault tolerance by removing the single point of failure.

3. Adaptation

A major task was to responding to changes in computational environment. One of the defining characteristics of Internet computing is the dynamics and volatility of the environment, and a parallel GA which does not adapt to its environment cannot use the available resources efficiently. I have developed a migration topology adaptation method which uses clustering on elite individuals from each island to build a migration topology that gives the GA better performance than a GA with static migration topology of a similar or larger connectivity level. Reduced connectivity means less load on the network and makes the system scale better. I have also developed a population adapter method that automatically finds the number of islands and island population sizes needed to solve a given problem with a total effort that compares favourably with the traditional trial-and-error approach. These two adaptation methods were chosen since previous studies have shown that migration topology and population size have a major impact on the duration and quality of the search [113, 28].

4. Evaluation

The final problem explored was how to evaluate the performance, scalability, and robustness of the framework. As a part of this research I have used the framework to implement a multi-layer VLSI (Very Large Scale Integration) floorplanner application
which, to my knowledge, is the first floorplanner to use a true 3D slicing structure representation. I have then used this application, in addition to standard GA benchmark problems, in a series of experiments to verify that the framework scales with both problem complexity and computational resource availability. I have also investigated the framework’s response to dynamic events (e.g. computers withdrawing from the calculation).

1.3 Major Contributions

The following major original contributions to the body of knowledge are made in this thesis:

1. A convergence model for heterogeneous environments is proposed (Chapter 4).

   This model is a novel extension of existing parallel GA convergence modelling to heterogeneous distributed island models. In contrast with previous work, the model can handle scenarios with computers of varying performance, and with customisable fault rates. Another contribution is an extensive investigation of migration policy parameters in dynamic Internet networks, using the convergence model.

2. A framework for Internet-based computation is proposed (Chapter 5).

   The framework uses a novel hybrid P2P architecture where island node activity is coordinated by supervisor nodes that offer global overview and opportunities for adaptation. Another unique feature of the framework is that it can run either on a P2P network or in a simulator mode without requiring recompilation or other modifications. The simulator mode contains Internet emulation code that can handle combinations of networks, switches, and computers, each with their own latency, bandwidth, and performance parameters.

3. An adaptive method for adjusting migration topology is proposed (Chapter 6).

   The migration topology adapter uses a novel clustering approach to dynamically rebuild the migration paths between islands with the goal of reducing communication overhead and improve search performance.
4. An adaptive method for adjusting population size is proposed (Chapter 7).

   The population adapter automatically searches for the number of islands and the population size needed to solve a problem in a given run-time environment.

5. A VLSI floorplanner application is implemented on the framework are carried out (Chapter 8).

   The VLSI floorplanner application is an example of the kind of challenging real world problems that a typical GA practitioner works with. This NP-hard problem uses a novel GA slicing structure representation, and the application was implemented and tested in the DGA framework.

6. Extensive experimental studies of performance in various Internet network scenarios (Chapter 9)

   The study contains a large-scale experimental study of scalability and robustness which is more extensive than earlier research on similar approaches to Internet-based DGAs.

1.4 Thesis Outline

Chapter 2 provides an overview of the general concepts and related research, sufficient for the easy understanding of the remainder of this work. The chapter introduces the basic GA and common extensions, and provides an overview of DGAs for cluster and multiprocessor environments.

   Chapter 3 contains a literature review of Internet-based GAs, which identifies trends and contributions of previous research in the field. Furthermore, unresolved current research problems and ideas for future research are discussed.

   Chapter 4 describes a convergence model for asynchronous distributed GAs. This model makes it possible to investigate migration parameter settings in a fraction of time needed for running the same experiments on a real parallel GA. Although the model makes certain assumptions that may not be true in real world applications, it is useful for comparing the relative performance in different scenarios and configurations, and experimental results obtained with the model have been used in the design of the Internet GA framework.
Chapters 5-7 introduce the major outcome of this work: a cycle-stealing framework for Internet-based distributed GAs. Chapter 5 describes the framework and its supporting tools with special emphasis on the simulator, which uses an Internet emulation code to replace the G2P2P Internet communication and distribution layer in a way that is transparent to the GA application running on the framework. The next two chapters discuss various aspects of adaptation. One of the defining characteristics of Internet computing is the dynamics and volatility of the environment. An Internet GA which does not adapt to its environment cannot use the available resources efficiently, while a GA which can collect feedback from the environment or the search state can use this information to modify its parameters during the run of the algorithm. This issue has received little attention in parallel GA research to date, and this thesis introduces two novel adaptation methods that improve the robustness and efficiency of an Internet GA: a method for adapting the migration policy which is described in Chapter 6, and a method for population size adaptation which is described in Chapter 7.

Chapter 8 presents a GA multi-layer floorplanning application, which is an open problem in the physical design of VLSI circuits [90]. Floorplanning is the problem of placing a set of large sub-circuits (blocks) on a layout surface to meet a set of design goals and constraints. It is a generalisation of the quadratic assignment problem, which is an NP-hard problem [48], and it is often solved by simulated annealing, force-directed heuristics or aggregate methods [96]. The floorplanning application is an example of the challenging real world problems that GAs are often applied to, and it has been studied in order to evaluate the scalability and performance of the DGA framework.

Chapter 9 presents experiments that have been carried out to measure the performance of the Internet GA. Both the computer lab and simulation have then been used to measure resource scalability, problem scalability, and performance on different scenarios, including stable LAN, and WAN with different dynamic behaviours.

Chapter 10 contains a summary of the results, recommendations for future research, and the conclusions of this study.
Chapter 2

Overview of Genetic Algorithms

This chapter is an introduction to GAs. It defines some terms that will be used in the remainder of the dissertation, and describes briefly how the simple GA works. Furthermore, the different types of parallel GAs found in the literature are examined. Readers familiar with the field may skip ahead to Chapter 3, which reviews previous research on Internet-based GAs and discusses research trends and unresolved research questions.

2.1 Introduction

GA is an adaptive heuristic search method premised on the evolutionary ideas of natural selection and genetics. The basic concept of GA is to simulate processes in natural systems necessary for evolution, specifically those that follow the principles of survival of the fittest. GAs are generally used in situations where enumeration is unpractical, and the search space cannot be traversed efficiently by traditional means, such as gradient- or heuristic-based search.

GAs solve problems by using a population of solutions to cast a net over the search space. The individuals in an evolving population sample many regions in the search space simultaneously. The rate at which the GA samples different regions corresponds directly to the probability of finding a good solution in that vicinity [64]. This ability of GAs to
focus their attention on the most promising parts of a solution space is a direct consequence of their ability to combine individuals containing partial solutions. This is because, using crossover, fit individuals with desirable characteristics are mated to produce new, possibly even fitter solutions. In this way the population of solutions gradually improves. Furthermore, mutations are allowed to occur with some low probability. This provides insurance against one particular solution becoming too dominant, replacing every other variation and stopping evolution altogether. Mutation also makes it possible to introduce partial solutions that are currently missing from the population, ensuring that every part of the search space can be reached.

### 2.2 Simple GAs

For the remaining discussion, it is valuable to examine a basic GA in more detail. The Simple Genetic Algorithm (SGA) [51] performs the following steps (see also Figure 2.1):

1. Generate an initial population, randomly or heuristically.

2. Compute and save the fitness for each individual in the current population.

3. Define a selection probability for each individual so that it is proportional to its fitness.

4. Generate the next population by recombining individuals, sampled from the current generation with a bias toward fit individuals, to produce new offspring.

5. Mutate each gene in the offspring with a certain probability.

6. Repeat from step 2, until a termination criterion is matched.

As an initialisation step, the SGA generates a set of solutions to a problem (a population of genomes). Then it enters a cycle where fitness values for all solutions in the current population are calculated, individuals for the mating pool are chosen (using the operator of selection), and after performing crossover and mutation on genomes in the mating pool, offspring are inserted into a population and some old solutions are discarded. Thus a new
generation is obtained and the process begins again. The SGA stops after one of the termination criteria are met, i.e. a “good enough” solution is recognised, no more improvements can be found, or the number of generations has reached its maximum value.

2.2.1 Theoretical Foundation

Holland [64] described the working of the SGA using the schema theorem, which calculates the growth of schemata (templates that match part of the search space). Given a problem with binary genomes of length $k$, the search space is $2^k$, and a schema is a string of length $k$ constructed by the symbols 0, 1, and *. An example is “01*1” which matches the genomes “0101”, and “0111”.

The schema theorem may be written:

$$m(h, t + 1) > m(h, t)\phi(h, t)(1 - \epsilon(h, t))$$

where $\phi(h, t)$ is the reproduction ratio (by selection), $\epsilon(h, t)$ is the disruption factor (by destructive recombination operators, e.g. crossover and mutation), and $m(h, t)$ is the number
of strings with schema $h$ at generation $t$. The schema theorem shows how the frequency of schemata change in time, and schemata whose fitness remain above the average mean and are not destroyed by disruptive factors receive exponentially growing increases in the number of samples.

Over the years there has been various criticisms of the schema theorem [69]. According to Altenberg, the theorem only states the higher valued strings are more likely to reproduce under a certain selection mode [4]. For a GA to perform well there has to be some knowledge about the search space embedded in the relationship between the operators, representations, and fitness, but the schema theorem itself yields only peripheral information about the performance of the combinatorial operators. The schema theorem also assumes an infinite population. However, real GAs are based on finite populations. Vose modelled the simple GA as a Markov chain, where the Markov chain states are populations [120]. In addition to modelling finite population, the Markov chain model can make direct predictions about the population composition, distribution of fitness and other statistics. When applied to more realistic GA models the Markov chain needs to be extended with additional states, and its complexity grow exponentially. This limits the the use of Markov chains to simpler GA models with small populations.

In order to provide a better theoretical insight in efficient GA design, Goldberg extended on the schema theorem by focusing on short, low order, and highly fit schemata called building blocks (BB) [51]. He observed that they can be efficiently mixed by crossover to produce new combinations to be tested. An assumption in the schema theorem, and for the proper working of the SGA, is that the building blocks contain data that is located in the same region on the genome. If this is not the case, then the probability that the recombination operators will break good BB grows, and the efficiency of the algorithm decreases. Much research has been done on creating efficient operators and representations for specific problems, e.g. permutation crossover for Travelling Salesman Problem-like applications, but it is also possible to develop more advanced GAs that use knowledge on the problem to identify the BBs explicitly. The next section discusses such advanced GAs.
2.3 Advanced GAs

2.3.1 Linkage Learning GAs

The destructiveness of the GA depends on the representation and the operators, and operators that tend to break some building blocks can make it more difficult for the GA to find good solutions. It follows from the schema theorem that even destructive operators can be overcome by increasing the reproduction ratio. Thierens [113] has analysed this case, and showed that premature convergence can be avoided by increasing the population size. However, the population size required grows exponentially with the problem size, and there are usually practical limitations on execution time and access to computational resources. Thierens also argues that common extensions to the simple GA - elitism and niching - do not significantly address this problem.

The problem of BB disruption is often referred to as the linkage problem. The problem is due to poor linkage, where the operators destroy BBs rather than mixing them. Linkage learning techniques are extensions to GAs that try to reorder the genomes to get tighter linkage in the BBs. Initially the inversion operator was suggested by Holland [64], but more efficient methods have been proposed, starting with the messyGA [55] which first identifies the BBs and reorders the genome before running the GA. Other optimisation methods try to model the good solutions found so far and use the model to guide further search, for example PBIL [11] and BOA [91, 92]. See also “Competent GAs” [53] for an overview of recent research in this field.

2.3.2 Multi-objective GAs

Many problems have multiple goals that cannot easily be combined into one evaluation function (e.g. quality vs. cost). The set of possible solutions is a set of points in the problem space that does not violate the problem constraints. A point is dominated by another point if that point is better on at least one of the sub-goals. The set of non-dominated points, also called a Pareto optimal set, contains solutions to the problem that represent different trade-off between the sub-goals where each goal is better than the others in some aspects,
while inferior in others (unless all sub-goals happen to be optimised at the same point which means that the Pareto set only has one member). The final solution(s) will be picked from the Pareto set, usually by the human user.

Standard GAs tend to converge on a single solution when used with a finite population, and have problems finding the full Pareto set. To find a more representative set of solutions, various multi-objective extensions have been suggested [31, 38]:

- Pareto-based methods give a selection advantage to less dominated individuals, so that the population moves toward the Pareto front. This is often combined with a mechanism to avoid crowding on a single point. The best implementations give a good approximation of the Pareto front, but this comes with a significant computational cost. The number of comparisons needed to update the front is $n^2 - n$ in the worst case in a population of $n$ individuals [118] (or even more if elitism is supported). While this figure can be reduced somewhat in practice, the reduction is small in most real world problems [18]. This overhead sets a practical limit on the population size in Pareto-based optimisation.

- Non-pareto based methods do not compare individuals for dominance in order to give an explicit approximation of the Pareto front. The non-pareto based methods can be further divided into aggregating and non-aggregating methods.

  - Aggregating methods combine each goal objective with arithmetical operators and weighting constants. To get a representative set the algorithm has to be run several times with different weighting constants. While simple and cost-effective, in practice it is hard to set the weights without extensive prior knowledge of the individual goals and their relationships.

  - Non-aggregating methods try to optimise each goal separately. A famous example of this approach is VEGA [99], which used modified SGA, in which the new population was selected with one objective at a time, so that for $k$ objectives with $n$ individuals, $n/k$ individuals were selected using objective 1, $n/k$ using objective 2, and so on. Recombination was then performed on the whole population.
2.4 Parallel GAs

The basic idea of parallel programming is to divide a large problem into smaller tasks and solve them in parallel using multiple processors. A similar divide-and-conquer strategy can be applied to GAs, and many different approaches have been studied. The research on parallel implementations of GAs can be loosely divided into three main categories [121]:

- Global implementations perform selection and mating globally on the whole population, while evaluating the individuals on the available computers in parallel. Global implementations are also sometimes called master-slave implementations in the literature.

- Island implementations put a subset of the population on each computer, and perform selection and mating locally on the computers. A small population usually increases the sampling error and genetic drift, and to counter this some individuals are occasionally moved between the subsets. Island implementations are also known as coarse-grained implementations.

- Cellular implementations assign an individual on each computer. The computers only communicate with their immediate neighbours, and the selection and mating takes part in this local neighbourhood. Cellular implementations are also referred to as fine-grained implementations.

While the global implementation works in the same way as a sequential GA, the last two methods make fundamental changes in the behaviour of the algorithm. The global model works with the whole population, but in the other implementation types the selection and recombination is local to an island or the local neighbourhood, respectively. These changes can be advantageous, and the features of each implementation is reviewed in the following sections.
2.4.1 Global Parallel GA

In global parallel GAs the population is kept on a master computer, and the selection, crossover and mutation is performed globally on the population. The evaluation of fitness is distributed among all computers, possibly including the master computer.

An early example is a GA with time-consuming pole balancing simulations by Fogarty et al. [45], which used a network of transputers and obtained reasonable speedups. The global model was later applied to timetable scheduling on shared memory computers [1]. The speed-ups were initially good, but did not scale well, mainly due to increase in communication cost. Similar communication bottleneck problems have also been reported in elsewhere (e.g. [61]).

The performance of global parallel GAs has been theoretically analysed by Cantú-Paz [26], who calculated the lower bound on the potential speedups of synchronous implementations to 50%. This means that the slave computers are idle half of the time, waiting for work assignments from the master which has to wait for all slaves to finish before continuing. The results suggest that a global GA is suitable for problems where the evaluation time is relatively long compared with the communication overhead.

2.4.2 Island Parallel GA

In an island parallel GA, the population is first divided into subpopulations, and each island runs a GA with one of the subpopulations. In addition to the basic operators of the GA a migration policy is introduced, which controls the exchange of individuals between the islands. Several strategies and parameters determine how this is done in detail.
The migration topology determines the paths along which individuals can move to other islands. If the number of islands is small, then a full migration topology can be considered, where all islands are connected with all the other islands. In this case a migrating individual is present in all other islands at once. Some studies of the island model assume that partial isolation of the islands is advantageous for the optimisation process and a very fast distribution of good individuals is detrimental (e.g. [109]). The idea is that smaller populations can explore different areas in the search space in parallel, creating greater diversity which can be exploited by the migration policy.

The exchange of individuals between the islands is controlled by the migration rate and migration interval. A migration phase can be included in each iteration of the GA, or only after a certain interval. A larger migration interval is normally used in connection with a larger migration rate. An alternative is to migrate precisely when the GA has reached premature convergence on the islands [23].

In most cases only the worst or a limited sequence of the worst individuals are replaced by the newly immigrated individuals. Other strategies for migration replacements have been proposed with the goal of maintaining a large variability inside the islands, e.g. to replace those individuals which are the most similar to the immigrants [93].

Some researchers are drawing inspiration from the evolution of natural populations in their work on island models GAs. The idea of punctuated equilibrium, proposed by Eldredge and Gould [42] to explain apparent rapid evolutionary change that does not leave fossil traces, were used by Cohoon et al. [32] to study the dynamic of the island model. In their experiments there was little change in the population between migration events, but new solutions were found shortly after migration. Although the focus of most research is to design faster search algorithms, island GAs have also been used to model natural evolution itself (see for example [84]).

2.4.3 Cellular Parallel GA

The population is provided within a spatial structure, such as a grid or a hypercube. A GA is run on the whole population, with global selection replaced by local selection and reproduc-
tion. The fitness of an individual is compared only to individuals in a local neighbourhood. The neighbourhood topology determines the placement of individuals and defines the paths along which individuals can propagate through the population. It is primarily introduced to fit the GA into the structure of memory and communication of a given parallel computer. Usually only the direct neighbours of a topology belong to the local neighbourhood.

For all positions of the neighbourhood topology a pair of individuals is determined by local selection, either with both partners selected from the local neighbourhood, or one individual as mating partner for the individual sitting in the centre of the neighbourhood. Local versions of the GA selection operators can be used (fitness-proportional, rank-proportional selection or tournaments). A replacement strategy defines which individuals from the local neighbourhood are to be replaced by the offspring. Generational replacement can be used without modifications if the offspring automatically replaces the individual in the centre of a neighbourhood [10]. It has been observed that individuals in cellular models are divided by distance, and as the algorithms proceeds, local neighbourhoods will emerge which are dominated by certain fit individuals. After several generations, competition will result in fewer and larger neighbourhoods [121].

The earliest implementations of cellular parallel GAs were Robertson’s classifier system on a CM-1 massively parallel computer [94], and ASPARAGOS [56] which was a strong hybrid GA optimiser that utilised local hill-climbing. Important additional work was made by Baluja [10], who studied the impact of different neighbourhood topologies systematically and found the mesh topology to work best in most of the test problems.

2.4.4 Hybrid Parallel GAs

Some parallel GAs do not fit into any of the types that were described in the previous sections. The methods of parallelisation can be combined to give hybrid models, and two-level hybrid algorithms are sometimes used. For example, Bianchini and Brown presented an application in which the island parallel GA was used at the higher level, while each island was a global parallel GA in which fitness calculations were parallelised [20]. Other combinations are possible and a detailed description can be found in [25].
2.5 Summary

This chapter included a brief description of basic GA types and introduced concepts that will be used in the remainder of the dissertation. In addition, the chapter also outlined the different types of parallel GAs that can be found in the literature. Building on this foundation, the next chapter reviews Internet-based GAs and identifies trends and unresolved research problems.
Chapter 3

Internet-based Distributed Genetic Algorithm Review

From the previous chapter, it can be seen that there are numerous implementations of parallel GAs targeted for multi-processor and cluster environments. With the increased interest in grid computing technology, there is a growing literature on Internet-based DGAs as well. Designers of such algorithms face a number of difficulties related to the nature of Internet computing, where the environment is inherently volatile and heterogeneous. In this chapter existing approaches to Internet-based GAs are evaluated and open problems identified \(^1\). Adaptation is also reviewed, with a focus on its applicability to Internet DGA design. With this knowledge, an Internet-based DGA framework will be constructed in the following chapters.

3.1 Frameworks

Recently several Internet-based frameworks have been reported. This section gives a brief historical account of relevant work. As illustrated in Figure 3.1, the research effort is di-

\(^{1}\text{This review has earlier been presented in part at the International Conference on Computational Intelligence for Modelling, Control \\& Automation, Gold Coast, Australia, 2004 [14].}\)
vided between global and island model DGAs. The absence of cellular model DGAs is not surprising, since it is a fine-grained model in which each individual evolves in parallel, and selection and recombination is performed in a local neighbourhood. The model has a high communication overhead which makes it unsuitable for Internet computation.

3.1.1 Global Model

Global model parallel GAs keep the population on one computer, and distribute the evaluation of fitness among available volunteer computers. The global model is suitable for problems where the fitness evaluation time is relatively long compared with the communication overhead [26], and the population is small enough to be stored on a single master computer. Examples of Internet GA applications that use the global model include evaluating game strategies in backgammon by letting individuals play against each other in tournaments [36], training and evaluating artificial neural networks specified by a GA genome [58], and drug design by searching large databases to find the most similar structure to a given genome [49].

JavaGenes [49] is an early example of a global GA, which is designed to facilitate cycle-
stealing from volunteer computers in a heterogeneous network. Although it was only tested in a LAN environment, it is reported to successfully scale to thousands of jobs on more than one hundred computers, using an drug design application with a time-consuming fitness evaluation that involves searching a large data base to find the most similar structure to a given genome. The framework is one of the first written in Java to achieve platform independence, and uses Condor for job distribution. JavaGenes uses checkpointing to save the state of a job, so that it can be moved when an volunteer machine is reclaimed. However, the paper reports performance problems caused by checkpointing overhead. There is no discussion of other alternatives, such as eager scheduling.

Hamilton-Wright and Stacey [58] uses the global model to distribute fitness function evaluations to idle computers using a server which acts as a registration point and job broker. No load balancing or other adaptation is implemented. The system has been tested on training of an artificial neural network, which has a expensive objective function. The system is implemented in a cluster environment, but its low communication resource requirement makes it feasible to use a similar design in an Internet GA.

Distributed Beagle [47] by Gagné et al. is an extension to the general Beagle GA library for sequential GAs. It uses a server with two kinds of clients: an evolver which contains the population, and evaluation clients that evaluate objective functions. The server implements a load balancing scheme which identifies each volunteer and keeps track of its performance, and uses this information to submit more jobs to faster volunteers. It uses eager scheduling if there are many volunteers available. The prototype implementation uses XML for data encoding, and sockets for communication. The framework is designed for LAN or high performance WAN.

3.1.2 Island Model

Although the global model is useful for many problems, most DGA implementations have been based on the island model [25]. In an island parallel GA, the population is divided into subpopulations that run independently, with occasional migration of fit individuals between the islands. Because of the island models applicability on wide range of real world
problems and its persisting popularity in the research literature, it has been the focus of this investigation. In this section relevant work on island model GAs will be reviewed, followed by a discussion on open research problems.

The island model divides the population in subpopulations that are evolved in parallel. This allows the GA to scale to the large population sizes needed by complex real world application, such as multi-objective combinational logic circuit design [35]. In contrast with the global model, which is an embarrassingly parallel application where each fitness function is calculated in isolation, the subpopulations need to exchange individuals periodically to counter premature convergence caused by the reduced population sizes. It has been noted that the island model can perform better than a sequential implementation of GA with similar settings, even when executed on a single computer. The reason is that unlike the sequential version, where random variations in the search tend to favour certain portions of the search space while suppressing others, the semi-independent exploration of the search space on each island acts to increase diversity.

Perhaps the first distributed evolutionary algorithm designed for the Internet is Chong’s distributed genetic programming framework [30]. The framework is based on the island model, where the islands are implemented as Java applets running in browsers on volunteer computers. Migration is generational and asynchronous, with migrants being sent and received from a migration server implemented as a Java servlet. Chong observed that individuals from islands that evolve faster may have too big impact on slower islands and cause premature convergence. He used flags to prevent the same individual being repeatedly used for migration, and picks the migrants randomly from the migration pool to maintain diversity in the total population.

Tanev et al. [110] developed a genetic programming framework based on the island model with a migration server that keeps a fix-sized pool of best individuals. The framework uses the DCOM protocol for object naming, deployment and communication. Using DCOM (or its successor Microsoft .NET) enables the framework to be used over LAN or the Internet but, being a closed protocol, restricts the network to computers using Windows. The islands

\footnote{An application is said to be embarrassingly parallel if it can be decomposed into independent pieces that can be computed on the nodes asynchronously and completely independent from work being done by the other nodes [46].}
register at the server to volunteer, and migration is performed asynchronously after each generational update. The emigrants are inserted in a migration pool, which is sorted on fitness and culled to size. Experiments show close to linear speedup on small experiments with eight workstations.

Paladin-DEC [107] is a similar project for Internet DGAs. The framework is based on the island model, and uses a central server to register volunteers, issue jobs, and handle a migration pool. The server also monitors performance and emails the final result to the client. The framework is implemented in Java to be platform independent, and uses the Java Remote Method Invocation (RMI) protocol for communication. The migration between islands is synchronous. An interesting aspect of the system is load balancing, which is implemented as an on-line adaptive method. At regular intervals the server compares the performance of the best and worst islands, measured in a number of generations since the last adaptation event, and adjusts the population size down and up, respectively, if their difference in fitness is greater than 10%.

Kojima et al. [70] is an experimental study aiming at reducing communication overhead in island GAs. Their suggested design uses a migration server which keeps a pool of elites, defined as the best solutions found in each island. The server is only accessed when an island has found a new best solution, or when it has run for many generations without finding new elites. In the latter case it is assumed that the island has converged prematurely, and the island will request elites from the server. The migrants are inserted at random in the population. The migration is dynamic both in intervals and topology, since it depends on the search state on each island. The prototype is implemented on a single parallel computer, but the authors have designed the framework with Internet computing in mind and the paper contains discussions on future implementations in Java with PVM (Parallel Virtual Machine) extensions. Experiments show that the performance is similar to synchronous parallel GA, while communication is much reduced.

The DGAs reviewed so far are all based on a client/server architecture, where a central server coordinates the distribution of the GA. Volunteer computers (clients) connect to the server and receive jobs to execute. The client/server architecture has several advantages: global control of migration and evolution; ease of administration, code distribution and set-
up; and minimal problems with firewall and proxy restrictions. However, there are some disadvantages with central servers. A server is a single point of failure and a potential bottleneck that limits scalability. Generally the server has to communicate with all islands, maintain and update a migration pool, and handle evolution control and statistics. Both communication and processing overhead increase with the number of computers, and may exceed the capabilities of the server. This has in fact been experienced by several authors [47, 30] who report diminishing benefits on adding computers when exceeding some hardware dependent limit.

An alternative framework design is peer-to-peer. The main advantage of using P2P is to remove single points-of-failures and server performance dependencies. P2P protocols have been studied in many fields of computer science, and been verified to scale well in both empirical and theoretical studies.

The first P2P-based framework for Internet DGAs is probably the DREAM project [89]. DREAM is designed for large agent-based applications, such as parallel GAs or ALife simulations. It is implemented on top of a pure P2P framework without servers. The framework provides basic functionalities for network integration and communication through a gossip-based P2P protocol, and applications are stored as object files that are downloaded and run by on top of agent entities. To achieve platform independence, the framework is implemented in Java and Corba. The framework is reported to scale well [65], and increases robustness by automatic relocation of objects from nodes that leave the network. The project has published general overviews, theoretical studies on P2P protocols, and applications such as job scheduling problems using parallel simulated annealing. The DREAM project statement mentions parallel GAs as a potential application, but unfortunately no results of parallel GA experiments have been published, and development appears to have stalled.

Recently, Tan et al. proposed a P2P Internet GA which shares many of the features of DREAM [108]. It is based on Java, uses direct asynchronous migration between peers and automatic code distribution. However, it extends on DREAM by adding a server for P2P node registration that is also used for creating an execution plan that assigns islands to available volunteer computers based on their physical properties. No experiments on the real system is presented, but a simulation study on a deceptive benchmark problem suggests
that good speedup can be achieved when the time between migrations is large compared to communication latency.

3.1.3 Open Research Problems

The promise of Internet DGAs is to allow vast computational resources to be applied to complex and time-consuming applications. To fulfil these expectations the framework must be able to make efficient use of many computers, making scalability and robustness important design goals. Since there are many kinds of computers and operating systems that are potential volunteers, platform independence and security is also important. There are several open issues.

1. Scalability

The JavaGenes project showed that global models, where fitness evaluations are independent, can scale to hundreds of machines, and are mainly limited by server overhead and communication latency. For the island model a limiting factor to scalability is the exchange of genetic material between subpopulations by migration. The most common approach to Internet DGA is using a migration server in the client/server architecture. Scalability of such a system is limited by the server capacity, since all inter-island communication passes through it, and the overhead grows with the number of computers. Both Chong and Distributed Beagle have reported diminishing benefits on adding computers when exceeding some hardware dependent threshold.

The interest in P2P architectures have mainly been motivated by their ability to scale better than client/server alternatives. Unfortunately this is hard to verify since few experimental studies including more than a handful computers have been conducted. Tan et al. demonstrate good speedup on a deceptive GA benchmark problem with up to 15 computers. To the best of my knowledge no GA results have been published for DREAM, but experiments on parallel simulated annealing involve only five computers [6]. It should be a subject for future studies to validate the claim that P2P offers better
scalability and robustness for parallel GAs, compared to the client/server architectures.

DREAM is a pure P2P architecture where all islands are peers, communicating by passing messages between them. Although such a system can scale, it is hard to control. P2P systems in other applications, such as file sharing and communication, often use a hybrid P2P architecture, where some nodes (super-peers, super-nodes) act as servers and controllers of client-peers [76]. This concept has not been used in Internet DGA P2P solutions so far, but offers means of combining the scalability of P2P with the global vantage point of a client/server architecture. This is a promising direction for future research.

2. Platform Independence, Security and Access

Systems that are based on C++ or binary communication protocols, such as Kojima et al., Tanev et al., and Distributed Beagle (see Table 3.1) restrict the pool of potential volunteers. More recent work use virtual machines to provide a stable and safe execution environment. Communication has been provided by communication packages such as MPI or Java RMI that automatically perform conversion of platform-dependent data format. However, the usefulness of this approach is limited by their difficulty in passing through firewalls. This has prompted a trend from binary formats toward structured text message protocols based on XML which can be sent through most firewalls, and are easily parsed by computers. In the work reviewed here, only Distributed Beagle uses XML for communication. In future work, accessibility and platform independence could be improved upon by investigating and using promising new web technology, such as HTTP-based SOAP message passing.

3. Adaptation

The performance of the GA depends not only on the representation, algorithm and operators chosen for a given problem, but also on the parameter settings. The process of finding appropriate parameter values is time-consuming, and considerable effort has gone into automating the process. A natural idea is to try to modify the parameters
during the run of the algorithm, using feedback of the current state of the search. Methods that use this approach are adaptive, and have been subject to much research. Surveys of the field can be found in [40, 62].

An Internet DGA cannot predict in advance what resources it will run on, and an Internet parallel GA using adaptive methods could change its algorithm so that it works more efficiently on available resources. Since adaptation is a continuous process, it also allows the Internet DGA to respond to dynamic events, such as volunteer withdrawal or network load changes.

There has been some efforts of developing adaptive methods for Internet-based DGAs. For instance, load balancing adaptation has been used in global model Internet DGAs to utilise faster computers more efficiently by assigning more function evaluations to them. Darwen [36] uses a simple reactive scheme, where the available volunteer receives one job each. Once a job is returned from a volunteer, the server immediately sends a new job to it. This distributes the load evenly, but since each job is sent separately the communication overhead is big. Distributed Beagle [47] uses a more elaborate scheme, where the server keeps track of each volunteer and its past performance. This information is used to divide the total jobs into batches that are sent to each volunteer, with the faster computers receiving more jobs.

Adaptation has also been suggested for island model Internet DGAs in the PaladinDEC project [107], which uses a scheme that distributes the computational workload evenly across the volunteers. The system uses a central server which monitors the speed of the volunteer computers. If the number of generations processed by the fastest and slowest computers differ by more than 10%, then 10% of the individuals of the slowest computer are moved to the faster one. The benefit of this scheme is that all islands will stop at approximately the same time. However, the paper lacks an analysis on the effect of having a mix of different population sizes. The risk is that the slower islands will contribute little to the search since too small population sizes are prone to premature convergence and bad performance, in effect decreasing the total population size and the quality of the algorithm.

The existing adaptive methods found in Internet-based DGA literature are limited to load balancing. In future research the use of adaptation could be extended to other
### Table 3.1: Overview of research papers and parameters that are discussed in the review of Internet-based DGA related research. For each paper, the table lists architecture, parallel model, intended network environment, adaptive methods (if any), implementation details, and migration policy parameters for (if applicable).

<table>
<thead>
<tr>
<th>Paper</th>
<th>Arch.</th>
<th>Model</th>
<th>Env.</th>
<th>Adaptive</th>
<th>Implementation</th>
<th>Migration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chong [30]</td>
<td>C/S</td>
<td>Island</td>
<td>WAN</td>
<td>-</td>
<td>Java servlets</td>
<td>generation</td>
</tr>
<tr>
<td>Tanev et al. [110]</td>
<td>C/S</td>
<td>Island</td>
<td>LAN, WAN</td>
<td>-</td>
<td>C++ DCOM</td>
<td>generation</td>
</tr>
<tr>
<td>Paladin-DEC [107]</td>
<td>C/S</td>
<td>Island</td>
<td>WAN</td>
<td>load balancing</td>
<td>Java RMI, JMS</td>
<td>generation</td>
</tr>
<tr>
<td>Kojima et al. [70]</td>
<td>C/S</td>
<td>Island</td>
<td>LAN, WAN</td>
<td>-</td>
<td>C MPI</td>
<td>convergence</td>
</tr>
<tr>
<td>DREAM [89]</td>
<td>P2P</td>
<td>Island</td>
<td>WAN</td>
<td>-</td>
<td>Java RMI</td>
<td>generation</td>
</tr>
<tr>
<td>Tan et al. [108]</td>
<td>P2P</td>
<td>Island</td>
<td>WAN</td>
<td>-</td>
<td>Java</td>
<td>generation</td>
</tr>
<tr>
<td>JavaGenes [49]</td>
<td>C/S</td>
<td>Global</td>
<td>LAN, WAN</td>
<td>-</td>
<td>Java Condor</td>
<td>-</td>
</tr>
<tr>
<td>Distributed Beagle [47]</td>
<td>C/S</td>
<td>Global</td>
<td>LAN, WAN</td>
<td>load balancing</td>
<td>C++ sockets</td>
<td>-</td>
</tr>
</tbody>
</table>

parameters that influence DGA performance on the Internet. Promising candidates for such research is adaptive methods of adjusting migration topology and population size, since there are indications from the literature that these parameters greatly influence the performance of a DGA, both in terms of duration and quality [113, 28]. Although these parameters have not yet been considered for adaptation on Internet-based DGAs, they have been used in adaptive methods for other types of GAs. This research is reviewed in Section 3.2.

### 3.2 Adaptation

In the discussion on open research problems in Internet-based DGAs, the existing use of adaptation was found to be limited to load balancing. The discussion identified adaptation of migration topology and population size as promising research areas. The purpose of this section is to review related adaptive research on these parameters in the literature.
3.2.1 Migration Topology Adaptation

Dynamic topologies have been investigated by Harter [60] who implemented difficult benchmark problems (F101, F102, and F103 [122]) in DGAs where everything was held constant except the topology, which was either static (uni-ring, hypercube or fully connected), or dynamic (random, or fitness-, distance- or diversity-based). In these tests, dynamic topologies performed better than static ones. The results cannot directly be applied to Internet DGAs since the system was only tested in an homogeneous set-up with low communication overhead. An interesting observation in the study was that random topologies performed better than the dynamic schemes with the same degree of connectivity. Harter argues that this is due to simplistic diversity and distance measures, which only compare the genotypic distance of the islands’ elite population.

A promising approach is the Injection-Island GA [75] which assigns different representations or objective functions to the islands, allowing them to search at different levels of resolution. Migration is only allowed from low resolution islands to higher resolution islands for more fine-grained optimisation, and in dynamic variants of the algorithm migration is only allowed if the genotypic distance between the best individuals in the two populations is greater than a threshold value. The dynamic approach improves the performance on the problems tested in the paper. No estimation of connectivity levels were made, or comparisons with random topologies of the same degree, so it is unclear if the performance improvement is due to the genotypic comparison itself, or the decreased communication in general. A problem with the Injection-Island GA is that is requires the user to provide multiple specifications to the problem to solve, and it assigns the islands in a hierarchy without regards to the capabilities of the available hardware or volatility in the run-time environment.

3.2.2 Population Size Adaptation

Smith and Smuda [104] presented a population size adapter based on the theoretical model of population sizing proposed by Goldberg, Deb, and Clark [54]. This model focuses on the correct decision between the BB and its closest competitor in a given partition, where the decision process takes place in the presence of noise that comes from the other partitions. The
probability of making a bad decision is a function of the population size, and by increasing the size it is possible to make this decision error as small as possible. This work was later extended to parallel GAs, and used by Cantú-Paz in his PhD thesis on efficient DGA design [26]. The limitation of this theoretical approach is that it assumes knowledge on problem characteristics that are hard to assess for realistic real world problems, such as the selective advantage of a building block compared to its closest competitor, and the fitness variance of the building block. Smith and Smuda [104] suggested an algorithm based on the Goldberg’s theoretical model that is easier to use, only requiring the user to specify a desired accuracy. The algorithm samples the schema fitness variance during the run of the algorithm, and adjusts the size the population so that the probability of selecting wrongly between two competing building blocks is less than the given accuracy threshold. Unfortunately, the variance estimation is very noisy, which can lead to unstable population size predictions.

Another approach relies on comparisons between parameter settings. Various population sizing schemes that are based on competing populations have been suggested for sequential GAs. In Schlierkamp-Voosen and Mühlenbein [100], the sub-populations are sized according to relative performance, with the more successful being allowed bigger populations. Eiben et. al [41] proposed a method which uses population sizing to guide the GA through exploration and exploitation stages, by increasing a population when the GA finds improvements or appears to have stalled, or otherwise decreasing it to focus on exploitation of the search space covered by the current population. The study also compared several adaptive population sizing methods on a number of test problems and found that they can outperform non-adaptive GAs by a large margin.

The Self-Adaptive Genetic Algorithm [63] runs three populations in parallel, each with different population sizes. At regular intervals, the fitness of the best individuals from each population is compared. The population size is then adjusted so that the size of the worst performing population comes closer to the best performing. There are also methods for shifting the size of all populations, e.g. increasing the size of all islands if \( f(P_1) < f(P_2) < f(P_3) \), where \( f(P_x) \) is the fitness of the best individual of population \( x \). A similar approach is used by the parameter-less GA [59], which is intended to simplify the application of GAs by automating the experimentation process. The algorithm sets crossover and selection
pressure to fixed values that satisfy the schema theorem, and uses a competition between GAs with different population sizes to find one big enough to achieve good performance. The smaller population is eliminated and new competition started with bigger population sizes if the bigger GA overtakes the smaller GA.

It would appear natural to extend competition-based population sizing methods to distributed computation by executing the populations in parallel. In fact, an extension for the parameter-less GA has been recently proposed [81]. However, the methods are still limited in that they keep the whole population on one computer. Using the island model would allow the system to scale to larger populations. The large number of computers needed to execute several DGAs in parallel need not be a problem if donated computational power from a large number of Internet-connected volunteer computers are used. This is an interesting possibility for future research, which has to address questions such as how to decide how many islands to use, and how to compare the competing DGAs.

3.3 Summary

This chapter reviewed recent Internet-based parallel and distributed GAs, and highlighted and compared the issues that affect the design and implementation of such frameworks. Open research problem in scalability, platform independence, and adaptation were identified and discussed.

The research on Internet-based parallel and distributed GAs is dominated by work using the island model on a client-server architecture. Scalability of such a system is limited by the server capacity since all inter-island communication passes through it, and the overhead grows with the number of computers. The interest in P2P architectures have mainly been motivated by their promise of better scaling. The DREAM project has shown that an agent-based P2P framework can be robust and scale well, but lacks central coordination that would allow a global overview. A hybrid P2P system was proposed as an alternative that combines the strength of both methods.

Numerous adaptive methods have been applied on sequential GAs to adjust parameters
such as crossover and mutation, population size, and genetic operators during the run on the algorithm in order to simplify the usage of GAs and improve performance. Adaptation is also a promising approach to the problem of how to design a DGA that performs well over a wide range of operating conditions, but examples in the literature have so far been limited to load balancing. An extension of adaptive methods was proposed, and two parameters identified as being crucial for good DGA performance: migration topology and population sizing.
Chapter 4

Asynchronous Parallel Genetic Algorithm Modelling

This chapter presents a novel extension to sequential convergence modelling, which is used to compare the effects of migration rates, topology and intervals on convergence in terms of selection pressure and fault tolerance in asynchronous parallel GAs\(^1\). The conclusions drawn from the experimental results have guided the design of the cycle-stealing Internet DGA framework described in Chapter 5.

4.1 Introduction

The performance of an island-model DGA depends on the available computational resources, the operators and representation of the problem, and the migration policy. The design of the sequential GA that runs on each island is not dependent on the environment, and the present theory can be used for parameters such as the estimation of population size and recombination parameters [51]. However, the impact of migration parameters on the performance depends on the capabilities of the network and the performance of the participating

\(^1\)The material in this chapter has been presented in part at the Congress on Evolutionary Computation, Canberra, Australia, 2003 [13].
computers. It would be time-consuming to determine the best migration policy settings by experimentation, and it would also be hard to make realistic scalability testing because of the limited availability of computers and networks with real world characteristics. To overcome these difficulties a convergence model has been developed to capture the main dynamics relevant to the migration policy settings.

The original contribution of the work presented in this chapter is a convergence model for asynchronous DGAs, which is used to compare the effects of migration rates, topology, and intervals on convergence in terms of selection pressure and fault tolerance. The model makes it possible to investigate migration parameters settings in a fraction of the time needed for running the same experiments on a real DGA.

The remainder is organised as follows. Section 4.2-4.3 describes the convergence model and fault rate extensions. In Section 4.4 the predictions of the model are validated by comparing them against a DGA with the same setup. Section 4.5 presents experiments that investigate aspects of running the DGA on a heterogeneous and volatile network by examining the effect of running the demes at different speed between migrations, and the effect of failure on the performance. Finally, the results are discussed and conclusions are drawn in Section 4.6.

### 4.2 Convergence Model

One factor that contributes to the performance of the GA is the selection pressure, which significantly influences the convergence rate. Too fast or slow convergence rates may cause the GA to fail to find good solutions, since very fast rates may cause the GA to converge prematurely on a suboptimal solution, while slow rates let the GA focus too much on discovery of new points in the search space to make real progress the search. The selection pressures for most common selection methods have been studied in sequential GAs (see for example [8]). It is known that migration can contribute to the selection pressure [124], and the objective of this chapter is to provide a model for asynchronous parallel GAs that can estimate the influence of different migration policies on the selection pressure.
4.2.1 Sequential GAs

The parallel GA convergence model is based on earlier research on a model for sequential GAs that uses two classes of individuals: good and bad [85]. The good individuals are those that are good enough for the problem to be considered solved, while bad individuals are any lesser solution. It is assumed that the GA is well designed so that a good solution will eventually be found, and it is of interest to investigate how fast a good solution will spread in a population. Focusing on the selection phase, for each generation the selection differential is defined as

\[ s^t = \overline{f}_s^t - \overline{f}^t \]  (4.1)

which is the difference between average fitness of the selected individuals \( f_s \) and the average fitness of the population \( f \) at generation \( t \). Assuming that the fitness of the population has a normal distribution\(^2\), the selection differential can be approximated as

\[ s^t = I \cdot \sigma_t \]  (4.2)

where \( I \) is the selection intensity, which is the expected average fitness value of the population after applying a selection method to the normalised Gaussian distribution [24]. Analytical expressions of selection intensity are known for many common selection methods [7]. \( \sigma_t \) is the standard deviation of the population at generation \( t \); this function is problem dependent. In the experiments an \( l \)-bit one-max problem is used where fitness to be maximised is the number of ones in an individual with an \( l \)-bit genome. If uniform crossover is used, the standard deviation for this problem is approximately

\[ \sigma_t = \sqrt{l \cdot P_t^t (1 - P_t^t)} \]  (4.3)

where \( P_t \) is the proportion of bits set to one in generation \( t \) [114]. The fitness of the one-max problem can be computed from \( P_t \) using

\[ f_t^t = l \cdot P_t \]  (4.4)

\(^2\)The normal distribution assumption is not generally true, especially in the beginning of the search, but it has been shown elsewhere to be a reasonable assumption for the one-max problem considered in this chapter [7].
Combining Equations 4.1-4.4 gives a recursive definition of $P_t$

$$\overline{f}_t - \overline{f} = I \sigma_t$$  \hspace{1cm} (4.5)

$$l \cdot (P_{t+1} - P_t) = I \sqrt{l \cdot P_t (1 - P_t)}$$  \hspace{1cm} (4.6)

$$P_{t+1} = P_t + I \sqrt{P_t (1 - P_t)}$$  \hspace{1cm} (4.7)

where $P_0$ is the initial proportion of bits correct, usually 0.5.

### 4.2.2 Assumptions and Limitations

The model only models selection. The effects of recombination are indirectly modelled through the standard deviation, which has been calculated for test problems with fixed operators and parameter settings. Another assumption is that the population is large enough to guarantee success, and the possibility of failure is not taken into account. However, the model is adequate for investigating how fast a good solution will spread in a population, and to make comparisons between different selection operators.

### 4.2.3 Parallel GAs

Cantú-Paz used the sequential model to investigate the selection differential in a synchronised parallel GA that performs migration after each generation [27]. Using the same framework, the model is now extended to allow arbitrary migration intervals, migration topologies, and islands with GAs that evolve asynchronously at different speeds.

The islands run the sequential model described above with the following extensions. The model is asynchronous in that the islands run at different speeds, and that the immigrants are kept in a migration pool until the island has finished its current iteration and is able to insert them into the population. Each island keeps a pool of migrants. Between generations the island checks if it is time to send emigrants, as defined by the migration interval. If so, it selects emigrants based on fitness and calculates their average fitness $f_{emg}$. The island inserts the emigrants in the migrant pool of all of its neighbours (defined by the migration
Time
Fast island
Slow island

Figure 4.1: Usage of migration buffers in asynchronous migration between islands running at different speed. The migration buffers are represented with dotted lines and the boxes represent GA calculations. The arrows are emigrant and immigrant communication.

<table>
<thead>
<tr>
<th>Topology</th>
<th>Order</th>
<th>Emigrate to island (_i)</th>
<th>Immigrate from island (_i)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fully connected</td>
<td>(r-1)</td>
<td>(\text{island}_j, j \neq i)</td>
<td>(\text{island}_j, j \neq i)</td>
</tr>
<tr>
<td>Uni-directional ring</td>
<td>1</td>
<td>(\text{island}_{i+1})</td>
<td>(\text{island}_{i-1})</td>
</tr>
<tr>
<td>Bi-directional ring</td>
<td>2</td>
<td>(\text{island}<em>{i-1}, \text{island}</em>{i+1})</td>
<td>(\text{island}<em>{i-1}, \text{island}</em>{i+1})</td>
</tr>
</tbody>
</table>

Table 4.1: Definition of neighbourhood for different migration topologies. \(r\) is the number of islands and \(\text{order}\) is the number of neighbours.

In the next sections each of these extensions are described in detail, and an outline of the model can be found in Algorithm 1.

4.2.4 Emigrants

The average fitness of the emigrants from each island needs to be calculated. The fitness of each individual \(f_i\) can be interpreted as samples of random stochastic variables \(F_i\) with a common distribution. Rearranging the \(F_i\) in increasing order gives the order statistics of \(F_i\), that is the sequence \(F_{1:n} \leq F_{2:n} \leq \ldots \leq F_{n:n}\). Let \(p\) denote the migration rate and \(n\) be the population size. If the emigrants are selected at random the expected fitness of the emigrants is the same as the mean fitness of the population. If the best \(m = p \times n\) individuals
Algorithm 1 Convergence model outline

\begin{algorithm}
\begin{algorithmic}
\State $t = 0$ // time
\State $\text{smig}(1..\text{numIslands}) = 0$ // Selection diff. after migration
\State Init $\text{Pt}(i=1..\text{numIslands})$ as in the sequential model
\While{(not converged or other stop condition)}{
\State $t = t + 1$
\State // Update GA on each island
\For{$(i=1..\text{numIslands})$}{
\If{(time to update island $i$)}{
\If{$\text{smig}(i) == 0$}{
\Comment{Sequential model}
\State update $\text{Pt}(i)$ as in sequential model
\} \Else{
\Comment{First update after migration}
\State $\text{Pt}(i) = \text{Pt}(i-1) + \text{smig}(i)$
\State $\text{smig}(i) = 0$
\}
\If{(time to migrate)}{
\State $\text{femg} =$ average fitness of emigrants
\For{$(j=$ first neighbour, \ldots, last neighbour)}{
\State add $\text{femg}$ to migrationPool in island $j$
\}
\}
\}
\Comment{Check migrantPool}
\For{$(i=1..\text{numIslands})$}{
\If{(time to update island $i$)}{
\If{(number of entries in migrantPool>0)}{
\State $\text{fimg} =$ average fitness of immigrants
\State $\text{fsur} =$ average fitness survivors
\State $\text{fnew} = (\text{numMigrants} \times \text{fimg} + \text{numSurvivors} \times \text{fsur})/\text{populationSize}(i)$
\State $\text{Pt}(i) = \text{fnew} / l$
\State $\text{smig}(i) =$ selection differential after migration
\}
\}
\}
\end{algorithmic}
\end{algorithm}

are selected, then their average fitness is

\[ \overline{f_{\text{emg}}} = \frac{1}{m} \sum_{i=n-m+1}^{n} E(F_{i:n}) \]  

(4.8)

Normalizing the random variables gives

\[ \overline{f_{\text{emg}}} = \frac{1}{m} \sum_{i=n-m+1}^{n} E(F_{i:n}) \cdot \sigma + \overline{f} = \overline{f} + \frac{\sigma}{m} \sum_{i=n-m+1}^{n} E(F_{i:n}) \]  

(4.9)

If the random variables have a normal distribution the following approximation can be used

\[ \sum_{i=n-m+1}^{n} E(Z_{i:n}) \approx n \cdot \phi(\Phi^{-1}(1 - p)) \]  

(4.10)

where \( \phi \) and \( \Phi \) are the probability density and the cumulative distribution functions of the fitness [7]. The island calculates the average fitness of emigrants using the equations and the current values for mean fitness and standard deviation (given by the sequential model). This value is then inserted in the migration pool for each of the island’s neighbours.

### 4.2.5 Immigrants

Let \( d \) denote the number of entries in the migrant pool for a given island. The mean fitness of the migrants is

\[ \overline{f_{\text{img}}} = \frac{1}{d} \sum_{i=1}^{d} \left( \frac{1}{m} \sum_{j=1}^{d} f_{i,j} \right) \]  

(4.11)

where \( f_{i,j} \) is the fitness of the \( j \):th immigrant from island \( i \). If the individuals to be replaced are selected at random the expected fitness of the survivors is the same as the mean fitness of the population. If the migrants replace the \( m \cdot d \) worst individuals in the population, the average fitness of the survivors (those not replaced) can be calculated as

\[ \overline{f_{\text{sur}}} = \frac{1}{n-d \cdot m} \sum_{i=d \cdot m+1}^{n} E(F_{i:n}) \approx \overline{f} + \frac{\sigma \cdot n}{n-d \cdot m} \phi(\Phi^{-1}(1-d \cdot p)) \]  

(4.12)
The average fitness of the new population after migration becomes

\[ f_{\text{new}} = \frac{d \cdot m \cdot f_{\text{img}} + (n - d \cdot m) \cdot f_{\text{sur}}}{n} \]  

(4.13)

4.2.6 Selection Differential after Migration

After the migrants are added the population will have a fitness distribution which is a mixture of several different normal distributions. Rank-based selection schemes, such as tournament selection, will pick individuals for the next generation from the mixed population by biased sampling that gives preference to good individuals. A series of experiments show that the population in the next generation after migration (that is, the distribution of the selected individuals) is close enough to normal for the sequential model to work satisfactorily, even in cases where migration has introduced very large fitness differences in the population. The trick is to find the mean and variance of this distribution.

The selection differential equation from the sequential case cannot be used to find the mean of the selected individuals, since it assumes that the fitness of the current population (from the individuals are selected) has a normal distribution, which is not the case after migration. Instead the selection differential must be calculated manually. Consider the case where the island has one neighbour and tournament selection with size 2 is used. The population consists of two parts, survivors and immigrants, which have different normal distributions. At each selection two individuals are selected randomly from the population. Let \( X_1 \) and \( X_2 \) denote the fitness of the samples. We have

\[ X_1 \sim N(m_1, \sigma) \]  

(4.14)

\[ X_2 \sim N(m_2, \sigma) \]  

(4.15)

\[ Y = X_1 - X_2 \sim N(m_1 - m_2, \sqrt{\sigma_1^2 + \sigma_2^2}) \]  

(4.16)

\[ Z = \frac{Y - (m_1 - m_2)}{\sqrt{\sigma_1^2 + \sigma_2^2}} \sim N(0, 1) \]  

(4.17)

The normal distribution assumption is verified in experiments that compare model and real GA performance in Section 4.4 below.
Table 4.2: Selection differential for the four possible selection events.

<table>
<thead>
<tr>
<th>Event</th>
<th>Selection differential s(event)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 migrant/migrant</td>
<td>( p \cdot p \cdot 0.7979 \cdot \sigma_{img}^{2} )</td>
</tr>
<tr>
<td>2 survivor/survivor</td>
<td>( (1 - p) \cdot (1 - p) \cdot 0.7979 \cdot \sigma_{sur}^{2} )</td>
</tr>
<tr>
<td>3 migrant/survivor</td>
<td>( p \cdot (1 - p) \cdot</td>
</tr>
<tr>
<td>4 survivor/migrant</td>
<td>( (1 - p) \cdot p \cdot</td>
</tr>
</tbody>
</table>

The selection differential (that is, the average fitness increase from one generation to the next) is half of the mean value of the fitness difference between \( X_1 \) and \( X_2 \). Since the better of two individuals is always selected, only the absolute value of the fitness difference needs to be considered. The average fitness difference of \(|X_1 - X_2|\) is equivalent to the mean of the right half of normal distribution of \( Y \) [114]. The mean of one half of a standard normal distribution is 0.7979 and the general equation for the selection differential becomes

\[
s = \frac{|m_1 - m_2| + 0.7979 \cdot \sqrt{\sigma_1^2 + \sigma_2^2}}{2}
\]  

(4.18)

The selection phase in the next generation can be divided into four events. Table 4.2 shows the selection differential when \( m_1, m_2, \sigma_1 \) and \( \sigma_2 \) have been substituted with variables relevant for each event. The total selection differential after migration becomes

\[
s = \sum_{i=1}^{4} P(event_i) \cdot s(event_i)
\]  

(4.19)

This selection differential calculation procedure can easily be extended to islands with more than one neighbour. Note that it is only necessary to use the procedure in the first generation after migration, since the next population will have a distribution that is approximately normal, and subsequent updates can be made using the sequential method.

### 4.3 Failure Model

Failure has a great impact on the convergence rate, and this section extends the previously presented convergence models with failure rates. The synchronous model is studied as a
bounding case, before turning the attention to the more realistic asynchronous model.

4.3.1 Synchronous Model Failure

In the synchronous model migration occurs after each time step, all \( r \) islands run at the same speed and the migration topology is fully connected, which implies that each island has \( d = r - 1 \) neighbours. The total selection differential is the sum of the differential caused by migration and the differential from the selection method [27].

\[
\begin{align*}
    s_{emg} &= \bar{f}_{emg} - \bar{f} = I_{emg} \cdot \sigma \Rightarrow I_{emg} = d \cdot e^{-1}(1 - p) \\
    s_{sur} &= \bar{f}_{sur} - \bar{f} = I_{sur} \cdot \sigma \Rightarrow I_{sur} = e^{-1}(1 - p) \\
    I &= I_{seq} + I_{emg} + I_{sur}
\end{align*}
\]

In a synchronised model with failure there is a certain probability that an island will fail. This research is mainly concerned with cycle-stealing where islands run on idle volunteer computers and where failure is usually due to reclamation of the computer by the owner. Let \( f \) denote the failure rate, and the stochastic variable \( X_t \) the number of islands that are alive at time \( t \). The idle time distribution has been examined on real world computer networks [21] and it is plausible to assume that the idle time has an exponential distribution, which means that \( X_t \) can be modelled as a continuous Markov chain with the transition matrix

\[
M(i, j) = \begin{cases} 
    \binom{j}{i} f^i (1 - f)^j & : 0 \leq i \leq r, 0 \leq j \leq i \\
    0 & : 0 \leq i \leq r, i \leq j \leq r
\end{cases}
\]

where \( M(i, j) \) is the probability to go from \( i \) to \( j \) islands in one time step. An island that is lost will not be replaced to preserve the assumption that all active islands have the similar fitness distributions. It is now easy to calculate the probability of the number of islands that are active at any given time as \( v_t = v_0 \cdot M^t \), where element \( i \) in vector \( v \) gives the probability that \( i \) islands are active. Initially, at time 0, all \( r \) islands are active and
Figure 4.2: Convergence time with varying migration rates for several fault rate settings (0.12, 0.1, 0.08, 0.06, 0.04, 0.02, 0.0 from top to bottom).

$v_0 = (0, 0, \ldots, 1)$. With the number of neighbours no longer constant, the Markov chain can be used to refine the calculation of $I_{emg}$. Element $i$ of the vector $v_t$ gives the probability that $i$ islands are active at time $t$. To calculate the survival probabilities for each island’s neighbours, only cases where $i > 1$ needs to be considered. Note that since the model is synchronous and fully connected all active islands are equal and the identity of the failed islands is unimportant. Treating each case separately gives a new $I_{emg}$ equation

$$I_{emg} = \phi(\Phi^{-1}(1 - p)) \cdot \sum_{i=2}^{r} (i - 1) \cdot v_{t,i}$$

(4.24)

Figure 4.2 shows the convergence time for a parallel GA with 5 islands and varying migration rates. The observed big increase in convergence time when the fault rate increases shows how it is increasingly likely that islands will become isolated as their neighbours fail, but it does not say anything of the success rate of the parallel GA itself.

Markov chains are powerful tools to study the probability for successful execution of the parallel GA for a given migration rate. Figure 4.3 presents plots of survival rates of islands (that is, the probability that at least one island is alive at the convergence time) for different
fault rates in the previously used parallel GA with 5 islands.

Higher migration rates increase the probability of successful execution at a given fault rate, since the convergence is faster. The probability drops rapidly when the failure rate exceeds a certain threshold, since it becomes increasingly unlikely that any island survives for the generations needed for convergence. If the fault rate in the network in which the parallel GA is run can be estimated, the synchronised failure model can be used to estimate the migration rate needed to achieve a high degree of success.

4.3.2 Asynchronous Model Failure

The synchronised model is a worst case scenario where no island can be restarted, the computer hardware is homogeneous, and the migration interval is minimal. To use Markov chains while allowing for the islands to converge at different speeds or using topologies of lower order than fully connected, not only the probability that a certain number of islands are alive needs to be known, but also the probability of survival for each island. It would be possible to extend the Markov matrix with additional states to represent this information,
but the size of the matrix would grow exponentially, and would still work under the rather unrealistic assumption that no island will be restarted once it fails. Instead the impact of volunteer withdrawal is added to the asynchronous convergence model by defining failure times for each island. The island will fail when the model has reached the failure time, and either be replaced with a newly initiated island, or withdraw completely from the parallel GA. This gives a greater flexibility in designing simulations, and several experiments with varying migration parameters are described in Section 4.5.

4.4 Verification

The model is verified by comparing its predictions against the average result of 20 runs of a parallel GA with the same parameter settings. Each island in the parallel GA is a generational GA running a 500-bit one-max problem with a population of 200 individuals, which is enough to ensure convergence. The GA terminates when all \( l \) bits of the one-max genome are set to one. In accordance with the selection intensity assumptions, the GA uses random initialisation, 2-tournament selection, uniform crossover with probability 1.0, and no mutation. The crossover is applied repeatedly in each generation to make the building blocks more randomly distributed across the population. The best individuals emigrate and replace the worst individuals on the receiving islands.

The speed of each island is defined as a multiple of a basic time step. An island running at speed 1 iterates every time step, while an island with speed 3 is three times slower. The migration interval is also defined in terms of the basic time step. A migration interval of 1 will give maximum migration rate, where each island sends emigrants to its neighbours after every iteration.

In each figure the model data points are connected with straight lines, while the parallel GA data points are connected with dotted lines. Figure 4.4 shows an experiment with a mix of speeds in a fully connected topology with long migration intervals. Figure 4.5 shows an experiment with uni-directional ring topology and high migration rate.

A series of island failure experiments has been conducted to verify the normal distrib-
4.5 Experiments

This section investigates parameter relations that can be used to define a migration policy for Internet-based distributed GAs. The latency and heterogeneity is simulated by using a mix of islands of different speed. Initially several configurations were tested (e.g., 1 fast island/7 slow islands, or 7 fast islands/1 slow island), and it was found that the same general
Figure 4.5: Migration rate 0.5, uni-directional ring, migration interval 4, 3 fast islands at speed 1, and one slow island at speed 5. The curves show island 4, 3, 2, 1 from top to bottom.

Figure 4.6: Migration rate 0.1, fully connected topology, 4 islands with speed 1, migration interval 4. Island 1 fails and is replaced with a new. The top curve is island 2-4 (indistinguishable), and the bottom is island 1.
behaviour and conclusions can be drawn over a wide range of configurations. In the following
presentation a standard set-up of eight islands, of which four are fast (speed 1) and four are
slow (speed 2) is used. Figure 4.7 illustrates the migration paths between the islands for
each topology used. All experiments use the 500-bit one-max problem with 200 individuals,
which is enough to ensure a high probability of convergence to the optimum. The best
individuals emigrate and replace the worst individuals on the receiving islands.

Three series of experiments are conducted to investigate the effect of the migration rate,
topology and interval on convergence. The last series of experiments also investigate the
fault tolerance. In each experiment two of the parameters are fixed, while one parameter
varies over a range of values, and the convergence times are compared.

4.5.1 Migration Rate

In this series of experiments the convergence time is calculated for each migration topology
with varying migration rates, and the results are shown in Figure 4.8 - Figure 4.10.

The migration induced selection differential is biggest in the fully connected topology,
where good results are broadcast to all islands at once. Increasing the migration rate gives
significant convergence time improvements at first, but the effect soon flattens out. Sim-
ilar effects are seen on a less dramatic scale in the ring topologies. The most striking
observation is the big improvement in convergence times between uni-directional and bi-
directional topologies, even though the increase in communication overhead is modest. The
Figure 4.8: Fully connected, the curves from top to bottom are island 1-4, 5-8. Time until convergence with four fast (speed 1) and four slow (speed 2) islands, migration interval 2, and varying migration rates.

uni-directional ring never reaches the convergence times of the other topologies, while bi-directional attains similar levels as in the fully connected case, although the migration rate needs to be roughly twice as high to maintain the same convergence time.

4.5.2 Migration Interval

This series of experiments varies the migration interval for each topology. The migration rate is set to the point after which no significant improvements were seen in Figure 4.8 - Figure 4.10. The results are shown in Figure 4.11 - Figure 4.13.

Smaller migration intervals increase the selection pressure, but the selection pressure quickly levels out. Tightly connected topologies converge slightly faster, but overall the topology does not any significant role.
Figure 4.9: Bi-directional ring, the curves from top to bottom are 2-3, 1-4, 5-8, 6-7. Time until convergence with four fast (speed 1) and four slow (speed 2) islands, migration interval 2, and varying migration rates.

Figure 4.10: Uni-directional ring, the curves from top to bottom are 4, 3, 2, 1, 5, 6, 7, 8. Time until convergence with four fast (speed 1) and four slow (speed 2) islands, migration interval 2, and varying migration rates.
Figure 4.11: Fully connected, migration rate 0.1. The curves from top to bottom are island 1-4, 5-8. Time until convergence with four fast (speed 1) and four slow (speed 2) islands, migration rate 0.1, and varying migration intervals.

Figure 4.12: Bi-directional ring, migration rate 0.2. The curves from top to bottom are 2-3, 1-4, 5-8, 6-7. Time until convergence with four fast (speed 1) and four slow (speed 2) islands, migration rate 0.1, and varying migration intervals.
Figure 4.13: Uni-directional ring, migration rate 0.2. The curves from top to bottom are 4, 3, 2, 1, 5, 6, 7, 8. Time until convergence with four fast (speed 1) and four slow (speed 2) islands, migration rate 0.1, and varying migration intervals.

Figure 4.14: Fully connected, the curves from top to bottom are island 2-4, 5-8. The impact of island failure on convergence. Migration rate 0.1, migration interval 10, island 1-4 are slow (speed 2) and 5-8 are fast (speed 1). Island 1 fails.
Figure 4.15: Bi-directional ring, the curves from top to bottom are 2, 3, 4, 6-7, 8-5. The impact of island failure on convergence. Migration rate 0.1, migration interval 10, island 1-4 are slow (speed 2) and 5-8 are fast (speed 1). Island 1 fails.

Figure 4.16: Uni-directional ring, the curves from top to bottom are 2, 3, 4, 5, 6, 7, 8. The impact of island failure on convergence. Migration rate 0.1, migration interval 10, island 1-4 are slow (speed 2) and 5-8 are fast (speed 1). Island 1 fails.
Figure 4.17: The impact of island failure on convergence. Migration rate 0.1, uni-directional ring, migration interval 4, island 1-4 are slow (speed 2) and 5-8 are fast (speed 1). Island 1 fails.

4.5.3 Fault Tolerance

The experiments displayed in Figure 4.14 - Figure 4.16 measure the effect of failure by removing an island without replacing or rebuilding the migration topology. In each experiment island 1 fails, and the convergence time is calculated for varying failure times. A parallel GA with fully connected topology is almost immune against this failure type. The uni-directional ring is most vulnerable since it only receives data from one source. If the migration policy increases the selection pressure then the number of migration events before failure is important and smaller migration intervals makes it possible for the island to have greater impact before failing. Compare Figure 4.17 with Figure 4.16 to see how shorter migration intervals make the islands converge faster.

Figure 4.18 shows how the convergence time for the worst island depends on the number of neighbours. In this experiment island, with $d$ neighbours sends its emigrants to $island_{i+1}, \cdots, island_{i+d}$. This experiment demonstrates how the closely connected islands survive failure better than topologies with few neighbours, but that the effect is most striking
Figure 4.18: The impact of island failure on varying number of neighbours (d=0,1,7 from top to bottom). Migration rate 0.1, island 1-4 are slow (speed 2) and 5-8 are fast (speed 1). Island 1 fails.

when adding islands to loosely connected topologies.

4.6 Summary

The original contributions of this chapter is a convergence model for asynchronous parallel GA, and an investigation of the impact on selection pressure of migration policy settings in Internet-like dynamic networks.

The convergence model was used in experiments that study the effect of migration policy parameters on selection pressure and fault tolerance using the one-max problem. The model has limitations in that it only models selection directly and makes certain assumptions on the fitness distribution of the problem which may not apply to real world problems. The experimental results on convergence times should therefore not be taken at face value. However, the model can be used to compare relative selection pressures in different scenarios, and make some predictions on when a parameter set gives higher selection pressure than another set. The selection pressure is important, because it may cause the DGA to converge
on good solutions significantly faster.

The experiments on the one-max problem show that dense topologies are more fault tolerant. Assuming a dense topology, a low to medium migration rate is sufficient, and the migration interval should be moderate. This conclusion is similar to the “rules of thumb” found in the literature (for example 10% of the population every 5-10 generations in [115]). However, these guidelines rest on assumptions of stable and homogeneous network with low communication overhead, that are mainly applicable to traditional DGAs designed for LAN networks.

In Chapter 3 Internet-based GA research was reviewed, and P2P architectures proposed as a promising research direction that has the potential to scale to population sizes needed by complex real world problems. Assuming a P2P architecture with many islands, dense topologies are hard to maintain in a typical Internet environment where latency and bandwidth limits acceptable communication overhead. However, the experiments in Figure 4.18 suggest that dense topologies can be reduced to more sparsely connected topologies while still maintaining good convergence in a volatile network environment. Lowering the topology has two benefits for the P2P architecture DGA. First, the communication overhead is greatly reduced from the exponential growth of the fully connected topology. Second, lowering the topology order would make it easier for the P2P transmit messages efficiently between the islands in an Internet environment where some direct node-to-node links are difficult to set up because of firewalls and other access restrictions.

The experimental results suggest that a low connectivity P2P architecture DGA can perform robustly in a volatile Internet network. In the remainder of the dissertation such a P2P DGA will be introduced and subjected to testing in Internet computing scenarios using both benchmarks and real world problems. The next chapter introduces the framework and its supporting tools.
Chapter 5

G2DGA Framework Design and Implementation

This chapter presents a cycle-stealing framework for parallel and distributed GAs \(^1\). The framework, which builds upon research done at the G2 group of Queensland University of Technology, is called G2DGA (G2 Distributed GA). G2DGA is designed to scale using a hybrid P2P architecture, which uses supervisor nodes in addition to island nodes.

5.1 Introduction

The research on Internet-based parallel and distributed GAs is dominated by work using the island model on a client/server architecture. Scalability of such a system is limited by the server capacity since all inter-island communication pass through it, and the overhead grows with the number of computers. The interest in P2P architectures has mainly been motivated by their promise of better scaling, and furthermore the convergence model experimental results discussed in Chapter 4 suggest that such a system can perform robustly in a volatile network. In addition to these theoretical predictions, the DREAM project, pre-

\(^1\)The material in this chapter has earlier been presented in part at the Genetic and Evolutionary Computation Conference Graduate Student Workshop, Washington DC, USA, 2005 [12].
Previously reviewed in Chapter 3, has demonstrated that an agent-based P2P framework can be robust and scale well. However, DREAM lacks central coordination that would allow a global view and control of the DGA. The main contribution of this chapter is a hybrid P2P architecture DGA that combines the strength of both client/server and P2P architectures.

G2DGA is a cycle-stealing framework for Internet-based parallel and distributed genetic algorithms, with a novel hybrid P2P architecture with island node activity coordinated by supervisor nodes that offer global overview and opportunities for adaptation. Another unique feature of the framework is that it can run either on a P2P network or in a simulator mode without requiring any recompilations or modifications. The simulator mode contains an Internet emulation code that can handle combinations of networks, switches, and computers, each with their own latency, bandwidth, and performance parameters.

G2DGA consists of a number of components, as illustrated in Figure 5.1. The design and basic functionality overall functions of these components will be described in the remainder, which is organised as follows.

G2DGA runs on top of a P2P architecture based on .NET Remoting called G2P2P, which is introduced in Section 5.2. The design of the framework, which distributes and coordinates the execution of GA applications, is described in Section 5.3. Section 5.4 discusses how G2DGA meets the design requirements of an Internet application. In order to facilitate testing and development G2DGA can be run in an Internet simulator instead of a P2P network. The simulator is described in Section 5.5. Applications that run on G2DGA can be either real GA implementations, or the convergence model problems earlier used in Chapter 4. GA application development is supported by a .NET GA class library called
dotGALib, which is described in Section 5.7. During the run of a GA application statistics are collected that can be stored in log files. The Analyser tool can read these log files and view their data in different ways for off-line analysis of GA performance. The Analyser is described in Section 5.6.

5.2 G2P2P

Internet computing is a form of distributed computing, where a computational task is divided into smaller sub-tasks that are apportioned out to devices across a distributed system [72]. Internet computing frameworks additionally have to be designed with the special attributes of the Internet in mind, such as access restrictions by firewalls and other security policies, the heterogeneous environment with different operating systems and available resources, the low bandwidth and high latency times, and the dynamic execution environment where computers can be expected to go on- or off-line at any time. The computers in an Internet computing environment are typically not dedicated to Internet computing, making the technique very attractive because it can utilise computational resources that would otherwise be unused, or it can make it possible to have resources for special computational purposes shared among users. It also means that high volumes of data that would otherwise require the power of supercomputers can be processed at low cost. The use of non-dedicated computers for this task is known as cycle-stealing.

Research in cycle-stealing systems range from systems dedicated to single problems (e.g. SETI@Home) to application specific frameworks. The G2 group in Queensland University of Technology has developed several cycle-stealing frameworks for grid computing [68, 78], and G2DGA builds upon the G2P2P [77], which is a P2P architecture based on .NET Remoting that supports object-based collaborative Internet computing.

The majority of cycle-stealing systems to date are based on client/server architecture. This architecture requires most communication to pass through a server creating a potential bottleneck. While some projects have begun to address this problem, they still maintain some form of coordinating server. G2P2P uses a P2P architecture to avoid such bottlenecks without requiring a complex set of hierarchical dedicated servers that other projects propose.
It also provides a more direct form of communication between volunteers.

Generally, a P2P computer network refers to any network that does not have fixed clients and servers, but a number of peer nodes that function as both clients and servers to the other nodes on the network. This model of network arrangement is contrasted with the client/server model. Any node is able to initiate or complete any supported transaction. Peer nodes may differ in local configuration, processing speed, network bandwidth, and storage quantity. P2P systems and applications have attracted a great deal of attention in computer science research. Prominent research projects include the Chord lookup service [105], APRANET, and the PAST storage utility (Pastry [95]).

G2P2P uses a physical location independent addressing scheme. This means that the object references is stable even if the host hardware changes, making it easy to implement object migration. G2P2P distributes objects across a decentralised P2P network using the distributed hash table functionality provided by Pastry. The pseudo-random properties of the hash function provides a load balancing effect, ensuring that each node receives roughly the same number of tasks. In large networks, it is impossible for each node to know the address of every other node. The routing mechanism is therefore designed to forward messages to a node that is closer to the destination address. The message passing is further optimised by storing multiple node references on each node, making it possible to use larger jumps around the address space by choosing the reference that is closest to the target node.

G2P2P handles code distribution, checkpointing and object migration transparently. It is implemented in the Microsoft .NET framework. .NET is object oriented, host platform independent, able to execute code from remote sources securely, and to contain language facilities and libraries for networking, making it suitable for Internet computing [111]. Programs written in .NET compatible languages (such as C#, Visual Basic.NET and Managed C++) are compiled into a standardised portable binary format. This binary is then interpreted at runtime by the Common Language Infrastructure (CLI) virtual machine, providing a single language-agnostic platform for the application development. .NET also has an extensive collection of library routines containing collections, GUI classes, and other utilities, which are designed to be independent of both programming languages and operating systems.
.NET is currently only fully available on Microsoft Windows platforms, whereas Java is available on many platforms. A part of the specifications of .NET has been standardised by ECMA and ISO, which may encourage standards-compliant implementations on other platforms. The Mono project [83] version 1.1.8 implements the low-level libraries needed for Linux, Windows, OSX, BSD, Solaris, and future improvements in the libraries should make it possible to run the applications with advanced graphical user interfaces (GUI) as well. dotGNU portable .NET [39] is another project trying to implement non-vendor specific implementation of .NET, with special emphasis on Windows Forms.

G2P2P communication is based on .NET Remoting, which is an RPC-like protocol under .NET that provides a rich set of classes that allow developers to ignore most of the complexities of deploying and managing remote objects. In .NET Remoting, calling methods on remote objects is nearly identical to calling local methods. When a client creates an instance of a remote object, it receives a proxy to the class instance on the server. All methods called on the proxy will automatically be forwarded to the remote class and any results will be returned to the client. From the client’s perspective, this process is no different than making a local call.

5.3 Framework Design

The G2DGA framework is designed to be a hybrid P2P with three kinds of nodes:
1. clients that initiate the DGA,
2. supervisors that perform monitoring and adaptation, and
3. islands that run a GA process.

Figure 5.2 shows these classes, and their most important methods. The supervisor creates the island objects and defines a migration policy that is sent to each of them, specifying the migration interval, rate, and a list of neighbouring islands. The islands run an instance of the GA problem.

The collaboration diagram in Figure 5.3 shows important actions during the run of the DGA. Migration between the islands is asynchronous and the migrants are sent directly between the peers (islands). The islands also collect and send feedback data from the islands, which the supervisor can use for adaptation. The supervisor also keeps the client updated of the DGA progress.

By its hybrid design G2DGA can retain a global overview of the GA which provides opportunities for adaptation that are difficult to achieve in a pure P2P architecture. Since all nodes can be hosted on any volunteer on the P2P network, bottleneck and single point of failure problems associated with traditional client/server systems can be avoided. Specifically, the hybrid design differs from client/server solutions in several ways:

- The supervisor is implemented as a G2P2P node, which is placed on a fixed computer. If the computer goes off-line or becomes busy, the load-balancer in G2P2P will automatically move it to another computer.
- The supervisor does not require special software or privileges to run, such as web server installation.
The supervisor is optional, and only used to improve the performance of the GA. The islands will continue to work without it.

By using direct communication between island nodes instead of passing migrants through a server the communication load is distributed evenly across the network.

5.3.1 Console

GA applications are started by a client object which contains methods that initiates contact with P2P network and launches the simulator which in its turn launches, monitors, and controls the islands. In G2DGA the client object is wrapped in a G2DGA console project which provides a GUI which makes interaction with the framework more user friendly.

The client reads XML scripts that define the problem and its parameters. Algorithm 5.5 shows an example XML script. The script contains at least one scenario definition, which defines the problem to solve, the dynamic behaviour of the network, migration policy, and adaptation parameters.

Figure 5.4 shows the G2DGA console. The left panel contains controls to select, execute,
and create scenario scripts, while the right panel contains displays output from the currently selected scenario. When a scenario is selected by the user, the RunScenario method in the client is called. The scenario can either be executed in the simulator or on the G2P2P network. In either case, the scenario XML file is parsed and the proper subclasses of Supervisor, Island, and Client created. After the scenario has been executed, the supervisor log data can be stored in files for later detailed study in the Analyser tool.

5.3.2 Island

An island is implemented as an Island class. It provides public methods for process control (such as start, stop, destroy, set migration policy). The island is expecting the GA problem to implement the IGAProblem or the IModelProblem interface, and to provide methods to initiate, step, get emigrants, and insert migrants. Given a problem in this form, the island class handles migration and statistics reporting automatically.

Internally the island contains an instance of a GA problem, and maintains a worker thread which updates the instance. Algorithm 2 shows the main loop of the worker thread. The island maintains a reference to its supervisor and pushes a feedback report to it periodically which contains information on number of evaluations, generations, population statistics, and best solutions found.

With an interval that is specified in number of generations, the worker thread asks the GA object for emigrants, and sends them to other islands specified by the migration
policy received from the supervisor. When an island receives migrants, they are stored in a buffer. The migration buffer allows only one set of migrants from each island, and the older migrants are replaced if new migrants arrives from the same island. This prevents faster islands from injecting a disproportionate amount of genetic material in slower islands, and preserves diversity in the total population. The buffer is checked after each update of the GA and its contents inserted in the population. The buffer is then cleared, and the new update commences.

Algorithm 2 GA Update procedure
1. Update GA
2. Check migration interval and send migrants from the GA to other islands
3. Check feedback update interval and send feedback statistics to supervisor
4. Insert migrants from migration buffer into GA
5. Clear migration buffer
6. Abort if any termination condition is true, otherwise repeat from step 1.

5.3.3 Supervisor

The supervisor class is responsible for island management, monitoring and adaptation. It provides public methods for process control (such as create island, start, stop, and destroy). The CreateIslands method takes the number of islands as an argument, and instantiates the island nodes, creates the initial migration policy and the adaptation methods. The supervisor keeps a reference of the client object which has created it, and send progress statistics back to it.

Islands send their status report to the InsertIslandFeedback method, which updates the global best solution, and maintains a record of the state of each island. When all islands report that they are done, or a good enough solution has been found, the supervisor calls the stop methods on each island and reports the final result to the client.

The average progress of the islands triggers adaptation, where the stored island statistics are used to improve the performance of the whole DGA. So far methods for adapting population size and migration topology have been developed, but the methods can easily be extended by adding new subclasses to the Adapter class.
The role of the supervisor is to coordinate and improve the performance of the DGA, but it takes no active role in the search itself. A failed supervisor will not cause failure for the DGA but the quality may suffer since without an active supervisor the DGA will revert to a standard non-adaptive DGA using the present migration policy until they reach their termination condition. Once the termination condition is reached, the islands remain idle until a new supervisor is started by the client application and contact with the islands is reinitiated. At that time the final results can be read, and the islands terminated.

5.4 Framework Implementation and Discussion

Alba and Tomassini have listed requirements of evolutionary algorithms for wide area networks [3]. These requirements are:

1. transparent exchange of data types in the Internet;
2. remote execution and tracing;
3. security issues relating access;
4. time-consuming process migration between clusters and computers;
5. adaptation to the dynamic behaviour of the links and computers involved in the execution of the EA;
6. fault tolerance.

The following sub-sections will discuss how hybrid supervisor/island design together with the use of G2P2P as the transport layer allows G2DGA to addresses these requirements.

5.4.1 Transparent Exchange of Data Types

G2DGA uses G2P2P as the transport protocol. G2P2P uses SOAP and serialisation into XML documents to pass messages between the P2P nodes. By using a text based format, binary compatibility problems are avoided.
5.4.2 Remote Execution and Tracing

G2P2P uses a physical location independent addressing scheme, which distributes objects across a decentralised P2P network using the distributed hash table functionality. The pseudo-random properties of the hash function provides a load balancing effect, ensuring that each node receives roughly the same number of tasks. G2P2P handles code distribution, checkpointing and object migration transparently. Using .NET as the implementation environment for both G2DGA and G2P2P provides a safe execution environment and platform-independent exchange of data by XML.

The routing mechanism is designed to forward messages to the known node that is closest to the destination address. The message passing is further optimised by keeping several sets of nodes, making it possible to use larger jumps around the circular identifier space, thus reducing communication overhead.

Tracing is handled by G2DGA by the collection of feedback data from the islands at the supervisor nodes, which optionally pass processed feedback data onto the client node.

5.4.3 Security Issues Relating Access

Firewalls and proxies can make direct communication between nodes difficult. G2P2P has made some effort in solving the access problems by allowing communication to be directed to port 80, which is kept open for HTML traffic on many firewalls. Additionally G2P2P improves on the standard .NET Remoting connection model, which opened a new connection in every call. The new connection protocol keeps the connection open during a whole node to node session (that is, as long as the communicating objects still reside on the same nodes), and this enables G2P2P to work in situations where firewalls limit access to one-way connections. Even if node A cannot open a connection to node B, node A is able to use the link established by node B.
5.4.4 Process Migration

Process migration is handled by G2P2P. When a computer leaves the calculation, G2P2P will serialize the objects running on the computer and pass them onto a neighbour node in the P2P network. The process is transparent for the G2DGA layer.

5.4.5 Adaptation to Dynamic Behaviour

The main focus of this research has been to develop methods that make G2DGA work efficiently and robustly in dynamic environments. These methods are adaptive in nature, and are discussed in detail in Chapter 6 and Chapter 7.

5.4.6 Fault Tolerance

G2P2P handles the basic object migration and checkpointing needed to move an island or supervisor when a computer leaves the P2P network. However, moving a node usually alters the dynamic behaviour of the system since the new computer may have different performance or access characteristics. The adaptive methods built into the supervisor will help overcoming these effects.

5.5 Simulator

Internet computing applications, such as G2DGA, are designed to work over large, dynamic, wide-area networks. To test and evaluate such applications, they must be deployed in realistic scenarios. Unfortunately, it is difficult to deploy and administer research software at more than a handful of Internet sites. Further, results obtained from such deployments on real networks are not replicable or predictive of future behaviour because wide-area network conditions change rapidly and are not subject to the researcher’s control.

To support G2DGA development and experimentation I have developed a simulator. The simulator is designed to work transparently for the application, so that the same G2DGA
code can be run in either G2P2P or the simulator without any changes.

The simulator uses an Internet emulator environment that enables unmodified G2DGA application to be deployed in a configurable Internet-like environment and subjects them to varying network conditions. The emulator routes the remoting messages through a set of virtual computers, networks, and switches, which cooperate to subject the traffic to the bandwidth, congestion constraints, latency, and the network topology profile. The G2P2P router is integrated in the simulator code in order to emulate its hash table-based message passing.

5.5.1 Overview

When a scenario script simulation is started, the InternetEmulator parses the network configuration file specified in the network argument of the Setup tag, and creates the virtual network. The simulator specific subclasses of the main G2DGA classes (Supervisor, Island, and Client) are deployed on the emulated network and all messages routed through it.

The most important classes in the simulator are:

- SimulatorClient: subclass of G2DGA Client. Reads and executes scenario scripts
- SimulatorIsland: subclass of G2DGA Island with simulator specific overrides
- SimulatorSupervisor: subclass of G2DGA Supervisor with simulator specific overrides
- InternetEmulator: maintains a list of computers on the virtual network, and emulates routing of messages on this network.

An emulated Internet consists of computers, switches, and networks. Computers connect to networks, and switches connect networks to each other. The computers and switches have packet queues with an given latency. Each network has a given bandwidth. The messages (or rather the packets that make up the message) are passed from the queues onto the network, and into the receiver’s queue after a delay specified by bandwidth and latency. Because each network has a fixed capacity, the emulator is able to emulate the effects of congestion according to application-specific communication patterns.
The SimulatorClient loops until the simulation is finished or aborted, updating the virtual time and calling the Update method in NetworkEmulator. The Update method is responsible for updating the network dynamics and execute methods on the remoting objects when the control messages arrive. This provides a mechanism for simulating heterogeneous networks with computers of different performance.

5.5.2 Related work

There are several tools that offer advanced emulation of the Internet, e.g. ns [86], but these seldom offer direct execution of unmodified software. Other research has customized the TCP-IP protocol to offer transparent emulation to unmodified software [116]. This approach requires an operating system that can be customized (e.g. Linux), and a dedicated computer that cannot be used for normal network activities (since the kernel networking code has been modified).

5.5.3 Design

The simulator targets the modern Internet, assuming Ethernet and the IPV4 protocol with 65536 bytes packet size (IPv4 allows 20-65536 bytes, but minimum required capacity of a host is 576 bytes). The header size is 20 bytes if no options are specified (the usual case), leaving 65516 bytes for the message. All computers on an Ethernet network listen and send messages on the same shared cable. If two messages are sent at the same time in Ethernet, the computers conflicting will wait for a random interval before retrying to send their packets. This behaviour is emulated in the scheduler, described below.

Figure 5.6 shows how the Internet emulator interacts with two remoting applications by using InternetEmulator, which sets up a virtual Internet with Computer, Network, and Switch nodes, as defined in configuration files. The message is routed through the virtual Internet, and once it has arrived, the real message is pushed and executed in the receiving application.
5.5.4 Network Configuration File

The emulator uses XML configuration files to set up the virtual network environment and control its dynamics. Figure 5.7 shows an example configuration.

5.5.5 Message Passing

The only message passing used in the simulator are .NET Remoting calls, which are RPC-like messages that contain the identity of the method to execute on the remote object, and the serialised function arguments.

When a message is sent, the route(s) between the sender and the receiver is calculated using the G2P2P router. Each route consists of a shortest-path ordered list of queues that need to be traversed to deliver a message from source to destination. Since G2P2P separates object addressing from the hardware layer, it is possible that a message between two computers will in fact pass through several computers along the way, each computer passing forwarding it to a computer that is closer to the logical address. By using the G2P2P router the simulated communication pattern will be the same as in the real P2P network.

Once the routing is calculated, the message is divided into packets that are stored in the sender computer’s queue. Each packet keeps track of its progress along the route,
<xml version="1.0" encoding="utf-8" ?>
<scenario>
  <configuration>
    <network name="backbone" bandwidth_KBs="1048576" />
    <network name="net1" bandwidth_KBs="10240">
      <computer name="cpu1" latency_us="1" speed="1.9"/>
      <computer name="cpu2" latency_us="1" speed="1.5"/>
    </network>
    <network name="net2" pAdd="0.01" pKill="0.01" bandwidth_KBs="10240">
      <computer name="cpu3" latency_us="1" speed="1"/>
      <addcomputers number="20" latency_us="1" speed="1-2"/>
    </network>
    <switch name="switch1" net1="net1" net2="backbone" latency_us="1" />
    <switch name="switch2" net1="net2" net2="backbone" latency_us="1" />
  </configuration>

  <!-- events time="20" -->
  <killcomputer name="cpu2"/>
  <!-- events -->

  <!-- events time="700" -->
  <addcomputers number="2" network="net2" latency_us="10" speed="3.14"/>
  <!-- events -->
</scenario>

Figure 5.7: Network configuration file example
and is updated by the scheduler. When all packets in the message have passed through
the emulated network, the InternetEmulator calls the remote object’s method with the
parameters specified by the message.

5.5.6 Background Load

Another important aspect of the Internet emulation is the ability to evaluate applications by
subjecting them to competing traffic and observing their responses to the resulting variations
in network performance. The most accurate method of doing this would be to directly
add packet generators with specified properties (packet size, and distribution over time).
However, since this generates computational overhead, the Internet emulator instead injects
background load by modifying the network’s bandwidth value dynamically as the emulation
progresses.

Background load is specified as a bandwidth demand between computer $i$ and $j$. Using
the router, the affected networks are determined, and their latency and bandwidth para-
eters adjusted. The effect is that an application competing with background load traffic will
have increased latency and lower available bandwidth.

5.5.7 Updating the Emulator

Every time the emulator’s scheduler is called, the internal timer is increased. The scheduler
then checks all networks:

- If a network is idle, all queues connected to this networks are checked for waiting pack-
  ets. A queue is randomly selected (emulating Ethernet’s sending restart algorithm),
  and the network status is changed to busy. The transmission time is determined by
  the bandwidth and the packet size;

- If the network is busy, then the transmission time for the current packet is checked. If
  the packet has arrived, then it marked to be inserted into the receiver’s queue (either
  a computer or a switch) after the latency delay specified by the receiver.
Figure 5.8: Comparison of mean fitness of simulator and lab experiments running on three computers.

<table>
<thead>
<tr>
<th>Message size</th>
<th>Same computer measured</th>
<th>Simulated</th>
<th>Different computers measured</th>
<th>Simulated</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 KB</td>
<td>1850</td>
<td>1628</td>
<td>1479</td>
<td>1093</td>
</tr>
<tr>
<td>100 KB</td>
<td>6843</td>
<td>6009</td>
<td>11441</td>
<td>10194</td>
</tr>
<tr>
<td>500 KB</td>
<td>27831</td>
<td>26844</td>
<td>48749</td>
<td>46967</td>
</tr>
<tr>
<td>1024 MB</td>
<td>54233</td>
<td>55961</td>
<td>99170</td>
<td>95142</td>
</tr>
</tbody>
</table>

Table 5.1: Comparison of message sending time for simulator and real networks. The times, given in microseconds, are averaged over 20 runs.

- If a packet has arrived at its destination it is removed from the network.

This process repeats for every queue in the path from source to destination.

5.5.8 Simulator Verification

The accuracy of the simulator was tested by comparing the time required to send .NET Remoting messages of different sizes with the time predicted by the simulator. The experiments were run in a computer lab at the Programming Languages and Systems research group at Queensland University of Technology. Each computer has Windows XP and contains a 800 MHz Intel Pentium III CPU, 512 MB of RAM memory and a Intel Pro 100 Mbps Ethernet
card, and they are connected to a LAN. Two scenarios were tested. In the first scenario both objects are on the same computer, while in the second scenario they are on different computers and must use the LAN connection to communicate. Table 5.1 shows the message time for both scenarios. Since the test program shares CPU time with the operating system in unpredictable ways, the computer measurements are averaged over 20 runs. For both scenarios the simulator correctly captures the growth in computation time as the message size increases.

The combination of G2DGA and the simulator has been verified by comparing the results of G2DGA running in simulation mode with the average result of 20 runs in the lab. These experiments fix the number of islands to 6, while varying the number of computers. The purpose of the experiment is to verify that the simulator accurately accounts for the difference in communication speed between nodes hosted on the same computer compared to nodes connected by a slower LAN connection. Furthermore, nodes will run slower if they have to share the computer with other nodes. Figure 5.8 compares mean population fitness for experiments using three computers, while Figure 5.9 shows the outcome for experiments using eight computers. The three computer experiments perform worse than the eight
computer experiments since several islands share the same computer, and the migration is uneven with a combination of slow network and fast object-object communication. The simulator generally follows the lab results well, and it accurately captures the performance gain when increasing the number of computers.

5.6 Analyser

The Analyser is used to parse and display data from the G2DGA supervisor log files, which contain data collected during the run of GA problems. The data collected include migration (with population statistics from each island), global optima updates, and adaptation events. Gnuplot [50] is used to display the data. The Analyser can also load multiple log files, to plot and compare different G2DGA runs, making it a valuable tool for development and experimentation.

Figure 5.10 shows the Analyser user interface. The log files are selected with the controls...
in the left panel, and plots are displayed in the right panel. The data scope to display can be selected on three levels: island, run, and log file. The data to be displayed is any combination of mean fitness, optimal fitness, connectivity, migration events, adaptation events, and island speed. When a higher level of scope is selected, the results on the lower levels is averaged. For example, when displaying mean fitness for a certain run, then the value shown is the average of the mean fitness of each island in that run.

The plots can be stored as bitmaps, Latex, or vector graphics for later inclusion in documents and other presentation material.

5.7 dotGALib

dotGALib is a library that contains tools that support implementation of genetic algorithms in .NET. The library is implemented in C#. The most important classes defined in the library are genomes and genetic algorithms. Each genome instance represents a single solution to the problem to be solved. The genetic algorithm object defines how the evolution should take place. The genetic algorithm uses an objective function (user defined) to determine how fit each genome is for survival. It uses the genome operators (built into the genome) and selection/replacement strategies (built into the genetic algorithm) to generate new individuals.

5.7.1 Programming Model

Algorithm 3 is an example of the programming model, where the user provides a objective function and declares instances of genome and genetic algorithm classes. Using standard C# practice, the GA parameters are set with properties.

The most important classes are GA and Genome, that are the base classes for all algorithms and genomes, respectively. These classes implement a number of methods for controlling and initialising the genetic algorithm. The library can easily be extended by overloading or adding new classes.
Algorithm 3 The F8 benchmark problem implemented with dotGALib.

```csharp
using dotGALib;

namespace GATest {
    class MainClass {
        static public double f8(Genome genome, Object data) {
            BitDecimalGenome g=(BitDecimalGenome) genome;
            double[] x=g.Decode();
            double res=1.0;
            double sum=1.0;

            for(int i=0; i<x.Length; i++)
                res+=x[i]*x[i]/4000;

            for(int i=0; i<x.Length; i++)
                sum*=Math.Cos(x[i]/Math.Sqrt(i+1));

            return res-sum;
        }

        [STAThread]
        static void Main(string[] args) {
            Genome g=new BitDecimalGenome(20, 10, -5.12, 5.11, false);

            GA ga=new SimpleGA(g, OptimizeMode.Minimize,
                                new Objective(f8), null);

            ga.Elitism=1;
            ga.CrossoverRate=0.9;
            ga.MutationRate=0.005;
            ga.PopulationSize=400;
            ga.SelectionMethod=new TournamentSelection();
            ga.TerminationMethods.Add(
                new TerminateMaxGenerations(1000));
            ga.Evolve();
            Console.WriteLine("Result: "+ga.BestGenome.Fitness);
        }
    }
}
```
5.7.2 GA

Figure 5.11 shows dotGALib’s termination class hierarchy. GA is the base class for all GA algorithms. dotGALib implements these subclasses to GA:

- **SimpleGA**

  The canonical GA, as described by Goldberg [51]. The GA uses selection to fill a pool with parents. Crossover and mutation is applied to contents of the pool, which then becomes the new population.

- **SteadyStateGA**

  Individuals are selected from the current population into a pool which is smaller than the population (default is 50%). Crossover and mutation is applied to contents of the pool, before its contents is copied back into the population, replacing the worst individuals.

- **MOGA**

  This is the base class for multi-objective GAs. The class supports genomes with multiple objectives, and contains methods for checking dominance and ranking. It
also contains methods for handling the Pareto front set, which are used to filter and present the set of final solutions. Algorithm 4 is an example of the programming model, where the user provides multiple objective functions in the GA declaration, where PredatorPreyMOGA is a subclass of MOGA which implements the Predator-Prey multi-objective optimisation approach described in [71].

Algorithm 4 Laumann’s MOGA test function [71]

```java
double moga1(Genome genome, Object data) {
    BitDecimalGenome g=(BitDecimalGenome) genome;
    return g[0]*g[0];
}

double moga2(Genome genome, Object data) {
    BitDecimalGenome g=(BitDecimalGenome) genome;
    return (1.0+g[1]*g[1])/(g[0]*g[0]);
}

void MOGATest() {
    g=new BitDecimalGenome(BitGenome.CrossoverType.TwoPoint,
        BitDecimalGenome.MutationType.BitFlip, 2, 10, false,
        Math.Sqrt(0.1), 1, 0, Math.Sqrt(5));
    ga=new PredatorPreyMOGA(g, OptimizeMode.Minimize, null,
        new Objective(moga1), new Objective(moga2), 0, 0);
    ga.PopulationSize=100;
    ga.CrossoverRate=0.9;
    ga.MutationRate=0.01;
    ga.Initialize();
    while(!ga.IsDone) ga.Step();
    ga.Stop();
}
```

5.7.3 Genome

Figure 5.12 shows dotGALib’s genome class hierarchy. The Genome class is the base class of all genome types. It contains the fitness value which is calculated by the objective function supplied to the GA class. Fitness is updated on-demand so that it does not need to be updated every time it is read. Recombination operators that modify the genome must set a flag to trigger recalculation of the fitness value by the next access. dotGALib implements these Genome classes:
• BitGenome: user defined bit genome;

• BitDecimalGenome: real value stored internally as bit genome, optionally using Gray coding which minimises the disruptive effect of mutation on bit genomes;

5.7.4 Selection

Figure 5.13 shows dotGALib’s selection class hierarchy. SelectionBase is the base class for all selection methods, and dotGALib implements these subclasses:

• TournamentSelection: implements a 2-tournament selection, which randomly picks two individuals from the current population and returns the one with the best fitness;

• RouletteSelection: in this selection method an individual’s slice of a Monte Carlo-based roulette wheel is an area proportional to its fitness. The “wheel” is spun in a simulated fashion and parents are chosen based on where the pointer stops.
5.7.5 Termination

There are two basic categories of termination conditions using the characteristics of the search; based on genotypes or phenotypes (fitness). Genotype methods checks the number of genes that have converged, defined as some percentage of the population has the same value in this gene. The search is terminated if the number of alleles exceeds a threshold value. Phenotype methods check the rate of progress in a predefined number of generations. One method is to check if a new optimal solution has been found during this period. Other methods check the mean fitness of the population instead, and terminate if no progress has been made. Yet another alternative is to look at the ratio between the best individual and the mean fitness of the population. As the population converges this should reach 1.

Figure 5.14 shows dotGALib’s termination class hierarchy. TerminationBase is the base class for all termination methods. It declares the method IsDone(), which each subclass overrides to return true if their termination criterion has been satisfied. dotGALib implements these subclasses:

- TerminateOptimaFound: terminate when the best individual has reached the desired fitness;
• TerminateGenomesConverged: Terminate if all loci contain the same allele for all individuals in the current population, or within the expected range if mutation is used;

• TerminateMeanFitnessConverged: Terminate if the mean population fitness has not improved in the last $n$ generations;

• TerminateBestFitnessConverged: terminate if the best individual's fitness has not improved in last $n$ generations;

• TerminateMaxEvaluations: terminate if the number of objective function evaluations exceed the allowed maximum value;

• TerminateMaxGenerations: terminate if the number of objective function evaluations exceed the allowed maximum value.

5.8 Summary

The major contributions of this chapter is the G2DGA framework and especially its novel hybrid P2P architecture, where the island node activity is monitored and coordinated by supervisor nodes. G2DGA also includes an Internet simulator, which is designed to run the framework and its applications transparently, without requiring modifications or recompilations. A final contribution is the dotGALib library, that supports general GA application development in .NET.

There are several advantages of basing the framework on .NET. The virtual machine provides a secure and trusted execution model which should make users willing to donate their machines for large-scale systems. The XML-based serialisation model avoids binary compatibility problems. Although .NET is less platform-independent than Java, the development of alternate .NET implementations should make this less of a problem in the future. Furthermore, the concepts described in this thesis and implemented in the current design can be re-implemented in Java without major modifications.

G2DGA is designed to scale by using a hybrid P2P design. The lack of a central server
means that the system is easier to set-up, and reduces the risk of single-point failures and bottle-neck problems. However, it is also desirable to maintain some server functionality in order to monitor and control the progress of the DGA. For this purpose, the supervisor node is used in addition to island nodes. This design differs from client/server architectures in that the supervisor node is not fixed to a specific computer, and requires no special privileges or server software to run. One major advantage of adding a supervisor is that it can collect island feedback which can be used for adaptive purposes. In the course of this research, two adaptation methods have been investigated in detail. These methods will be described in the next two chapters.
Chapter 6

Adaptation of Migration Topologies

This chapter presents a method to dynamically change the migration topology in order to adapt to the GA search state, with the goal of reducing communication overhead while maintaining good performance of the DGA ¹.

6.1 Introduction

The performance of island model DGAs is to a great extent determined by the way genetic information is exchanged between the islands. This is determined by a migration policy, which specifies the rate, amount, and topology of inter-island communication.

Migration is of special concern for Internet-based DGAs since the communication overhead (narrow bandwidth and high latency) is high, making massive data transfer and tight synchronisation difficult. The communication overhead also has implications for scalability, since the connectivity may grow exponentially with the number of islands. It is therefore desirable to minimise data transfer between the participating computers without compro-

¹This method has earlier been presented in part at the Genetic and Evolutionary Computation Conference, Washington DC, USA, 2005 [17].
mising the performance of the Internet-based DGA. Internet GAs also work in a dynamic environment where the islands are likely to make uneven progress. A migration policy cannot be expected to work efficiently in such conditions without the means to adjust the migration policy according to the search state of the DGA.

To address these problems I have developed and investigated several strategies for dynamic migration topology adaptation with the goal of reducing communication overhead. The communication load is determined by the migration policy, and the migration topology is of special interest since DGAs with dense topologies generally find the global solution using fewer function evaluations than DGAs with sparsely connected ones [28]. The focus of the investigation is therefore to find a migration topology which has low connectivity, but still maintains a good performance of the DGA.

The main contributions from the work is the use of a clustering technique on island feedback to assess the search state, and an investigation of how the clustering can be used to build a new migration topology.

The remainder is organised as follows. Section 6.2 contains a detailed description of the adaptation mechanism. Two types of experiments have been carried out to test and verify the proposed method. Section 6.3 is a comparative study of design options on benchmark test functions, that are known to be challenging for GAs, and that be easily tuned to a required level of difficulty. Finally, the results are discussed and conclusions are drawn in Section 6.4.

6.2 Migration Topology Adaptation

The communication overhead in a DGA is directly influenced by the migration topology, with sparse topologies giving less overhead. The objective of the migration topology adapter is to build a topology that minimises communication overhead while maintaining good performance.

Since the islands have different search states, it is unlikely that all migration exchanges are equally beneficial for their recipients. By removing less beneficial links it is possible to
reduce the communication overhead without compromising the quality of the search.

6.2.1 Proposed Method

The approach here uses a clustering technique based on genotype distance to determine the similarity between the search state of different islands. Since no direct competition is used, the need of fairness is relaxed and the island can run asynchronously. Furthermore, since the search state is influenced by the runtime environment, it allows the adapter to take the dynamic behaviour of the Internet into account.

Clustering is an unsupervised learning method which divides data into natural groups automatically based on similarity [44]. General clustering is an NP-hard optimisation problem with a vast literature on approximation algorithms [2]. In the current application the problem is simplified by the fact that only a small subset of individuals is evaluated. The computational expense grows with the number of islands, which is much smaller than the total population. Furthermore, it is not necessary to find optimal clusters. A heuristic method that finds useful clusters for efficient adaptation is enough.

In this work clustering is used on the feedback data received from the islands to find groups of islands that work in similar partitions of the search space, and to optimise the migration topology with the goal of reducing the connectivity while maintaining good performance.

The main steps are outlined in Algorithm 5. Clustering is first done on the whole data set, before removing all data points but the most recent from each island, and calling MakeTopology to create the new topology. Several strategies have been investigated for MakeTopology, and they are described in detail in Section 6.3.4.1. If any parameter is to be changed, UpdateIslandTopology will send the new migration policy to the affected islands.
Algorithm 5 Migration topology adapter pseudo-code

procedure Adapt()
    cluster_set=MakeClusters(data_set)
    new_topology=MakeTopology(
        RemoveNonActive(cluster_set))
    UpdateIslandTopology(new_topology)

6.2.2 Clustering Algorithm

The clustering algorithm used in this work is K-medoid [67], which can be applied to all genomes, including those that only provide nominal data, given a distance function that compares two genomes, e.g. Hamming distance for bit genomes. The data set $D$ contains the most recently received best individual genomes from each island. The stored number of individuals from each island is limited, and if needed, the oldest data point from the same island is removed to make place for a newly arrived one. The most recent data point from each island is marked as being active.

The aim of K-medoid is to partition the data set of $n$ data points into $K$ groups so as to minimise the total within-group sum of distances about $K$ representative points, or medoids, among the data points. Given a distance matrix, $(d[x_i, x_j])$, of pairwise distances between the $n$ data points to be clustered, $\{x_1, \ldots, x_n\}$, for a prespecified number of clusters $K$, the algorithm seeks to minimise the objective function $f$ defined by

$$f(\mathcal{M}) = \sum_{i=1}^{n} \min_{m \in \mathcal{M}} d[x_i, m],$$

where $\mathcal{M}$ is the set of mediods, with $\mathcal{M} \subseteq \{x_1, \ldots, x_n\}$ and $|\mathcal{M}| = K$.

The algorithm initially selects $K$ data points randomly to be centroids. It then proceeds iteratively by first constructing clusters by adding the data points to their closest centroid, and then setting the most central data point in each cluster as its new centroid. The process is repeated until the centroids are stable.

The setting of the number of clusters parameter $K$ is a non-trivial problem. To find a good $K$ a Minimum Description Length (MDL) criterion [57] from information theory is
used. The description length is a measure of how efficiently a given cluster model encodes the data set, and consists of two terms that can be interpreted as the coding of model parameters, and the coding of data given the model parameters. If the data fits the model poorly, then the latter term is large since the signal will be noise-like. To find the best $K$ the MDL of several $K$ values will be calculated, and the smallest MDL (that is, the most compact clustering model) will be used.

The MDL idea is to minimise

$$L(C) + L(D|C)$$

where $C$ denotes the cluster set, $D$ the data set, $L(C)$ the number of bits needed to encode $C$ and $L(D|C)$ the number of bits needed to encode the data set, given the clustering. As shown by Vitanyi and Li [119], MDL can be derived from Bayes’ rule:

$$P(C|D) = \frac{P(D|C) \cdot P(C)}{P(D)}$$  \hspace{1cm} (6.1)

By taking the negative logarithm on both sides of the equation, the expression becomes

$$-\log_2 P(C|D) = -\log_2 P(D|C) - \log_2 P(C) + \log_2 P(D)$$  \hspace{1cm} (6.2)

According to Shannon’s entropy formula, the minimum number of bits needed to encode $x$ is $-\log_2 x$, and by substituting the logarithms the expression becomes a minimisation problem expressed in the length of the coding

$$L(C|D) = L(D|C) + L(C) - L(D)$$  \hspace{1cm} (6.3)

$L(D)$ can be ignored since it does not depend on $C$, and the problem simplifies to the minimisation of

$$L(D|C) + L(C)$$  \hspace{1cm} (6.4)

which can also be expressed as

$$-\log_2 P(C|D) - \log_2 P(C)$$  \hspace{1cm} (6.5)
$P(C)$ can be estimated by assuming that the data are generated by a mixture of underlying probability distributions in which each component represents a different cluster. $P(C)$ is the likelihood of cluster set $C$, that is the probabilities of cluster memberships for each data point, and is calculated thus:

$$P(C) = \sum_{d \in D} \sum_{c \in C} PDF(distance(d, c.mediod), c.stddev) \times Prior(c) \quad (6.6)$$

where $PDF$ is the normal probability density function, $c.mediod$ is the representative data point of cluster $c$, and the prior probability of $c$ is

$$Prior(c) = \frac{c.Count}{D.Count} \quad (6.7)$$

where $c.Count$ is the number of data points in cluster $c$, and $D.count$ is the total number of data points.

$P(D|C)$ is calculated as the sum of standard deviation between data points and the cluster set, and will be big if the cluster is a bad fit to the data.

$$P(D|C) = \sum_{c \in C} \frac{\sum_{d \in D} distance(d, c.mediod)}{D.count - 1} \quad (6.8)$$

Combining equations 6.4 - 6.8 provides the MDL expression used in this research. The adapter uses a bootstrap function to try $K \in [1 \ldots K_{max}]$ and selecting the $k$ with minimal MDL. The optimal $K$ tends to be small compared to $n$, and $K_{max} = \sqrt{n}$ has empirically been found to be a reasonable setting.

### 6.3 Experiments

This section contains a comparative study of design options on benchmark test functions that are known to be challenging for GAs, and which can be easily tuned to a required level of difficulty.

Three types of experiments are conducted in this section. The first series compare static
topologies, and are meant to provide a baseline to compare the results of the migration topology adapter results against. The second series study the long-term impact of adaptation by comparing static and adaptive topology performance in scenarios with a single adaptation event. Finally, the effect of continuous, repeated, adaptation is evaluated. In this experiment, adaptation (with clustering and migration topology construction) takes place every 30 generations.

6.3.1 Test Problem

The adapter was tested using the benchmark problems from the Whitley test suite [123]:

\[
F_{101}(x, y) = -x \sin\sqrt{|x - \frac{y + 47}{2}|} - (y + 47) \sin\sqrt{|y + 47 + \frac{x}{2}|} \quad x, y \in [-512, 511]
\]

\[
F_{102}(x, y) = x \sin\sqrt{|y + 1 - x|} \cos\sqrt{|x + y + 1|} + (y + 1) \cos\sqrt{|y + 1 - x|} \sin\sqrt{|x + y + 1|} \quad x, y \in [-512, 511]
\]

\[
F_2(x, y) = 100 \times (x^2 - y)^2 + (1 - x)^2 \quad x, y \in [-2.048, 2.047]
\]

\[
F_8(x, y) = 1 + \frac{x^2 + y^2}{4000} - \cos x \times \cos \frac{y}{\sqrt{2}} \quad x, y \in [-512, 511]
\]

\[
F_8F_2(x, y) = F_8(F_2(x, y))
\]

These problems are multi-modal test functions that are relatively difficult for most optimisation algorithms. They were constructed to be non-symmetric with many local optima. In the experiments, harder versions of the test problems have been created by scaling them to \( N \) variables using this expansion:

\[
F(x_1, x_2 \cdots x_n) = \frac{F(x_1, x_2) + F(x_2, x_3) \cdots F(x_n, x_1)}{n}
\]
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>problem size</td>
<td>10 bits, 10 variables</td>
</tr>
<tr>
<td>number of islands</td>
<td>16</td>
</tr>
<tr>
<td>population size</td>
<td>30 individuals/island</td>
</tr>
<tr>
<td>elitism</td>
<td>1 individual</td>
</tr>
<tr>
<td>GA</td>
<td>Generational</td>
</tr>
<tr>
<td>crossover</td>
<td>one point, pc=0.7</td>
</tr>
<tr>
<td>mutation</td>
<td>bit flip, pm=0.005</td>
</tr>
<tr>
<td>migration topology</td>
<td>uni-ring</td>
</tr>
<tr>
<td>migration interval</td>
<td>5 generations</td>
</tr>
<tr>
<td>migration rate</td>
<td>1 individual (best replaces worst)</td>
</tr>
</tbody>
</table>

Table 6.1: GA Parameters

The expanded versions have non-linear interactions between the variables, which means that they are not separable and hard to solve with hill-climbing optimisation techniques.

To be able to compare results with previous research, the parameters from Harter [60] have been used in all experiments (see Table 6.1). The GA is run for 200 generations in each run, for a total of 50 runs. The experimental results show the averaged value for all runs.

### 6.3.2 Connectivity

An important measure of communication overhead is the connectivity metrics, which in this study is defined as the number of one-way migration paths between the islands. For $n$ islands, this means that a fully connected topology has connectivity $n(n-1)$, since each island send emigrants to each other island, and a uni-directional ring topology has connectivity $n$. The normalised connectivity is the connectivity degree relative to the fully connected topology.

For example, the normalised connectivity for a uni-directional ring is

$$c = \frac{n}{n(n-1)}$$

All connectivity measurements reported in the experimental results are normalised.
Table 6.2: Comparison of static topology performance: final mean and standard deviation of 50 independent runs.

<table>
<thead>
<tr>
<th>Topology</th>
<th>Mean fitness</th>
<th>Optimal fitness</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cube</td>
<td>0.79 ± 0.069</td>
<td>0.89 ± 0.076</td>
</tr>
<tr>
<td>Fully connected</td>
<td>0.74 ± 0.071</td>
<td>0.82 ± 0.077</td>
</tr>
<tr>
<td>Uniring</td>
<td>0.81 ± 0.071</td>
<td>0.92 ± 0.08</td>
</tr>
</tbody>
</table>

6.3.3 Static Topologies

6.3.3.1 Experimental Design

Static topologies are topologies that are fixed and do not change during the run. A static topology consists of a physical layout of the subpopulations and the connections between them. Three static topologies have been tested to provide a baseline to compare the other experimental results with. The topologies were fully connected (densely connected), hypercube (medium connected), and uni-ring (sparsely connected). The migration interval is fixed to 5 generations, while in this experiment the migration rate have been normalised so the total number of individuals migrating is the same for each topology. The migration rate is 1 for fully connected, 4 for hypercube, 16 for uni-ring topologies.

The normalised migration rate allows a more fair comparison between different topologies, since a non-normalised experimental set-up would allow densely connected topologies to transfer many more individuals during the course of the experiment than sparsely connected topologies.

6.3.3.2 Results

Figure 6.1 shows a comparison between static topologies, using the experimental setup described above, and the final results for the different topologies are listed in Table 6.2. The general observation from these experiments is that higher connectivity promote better average fitness, while lower connectivity promote better optimal performance. In denser topologies the new solution spreads quickly and helps to improve the average fitness quickly, but the sparser topologies allow more time for solutions to evolve in the islands and avoid premature convergence which is reflected in their superior optimal performance.
Figure 6.1: Optimal (above) and mean fitness (below) of static topologies (equal migration) for the F101 benchmark problem averaged over 50 independent runs.
6.3.4 Dynamic Topologies

6.3.4.1 Experimental Design

Dynamic topologies may change during the run, based on some set of criteria. The clusters created by the clustering algorithm were used to create four kinds of topologies (see also Figure 6.2):

- **Similar**: Ring topology between the islands in each cluster, no communication between clusters.
- **Diverse**: Each island in a cluster communicated with all islands in all other clusters.
- **Mixed**: Ring topology between islands in each cluster, and one link to each other cluster.
- **Fully**: Each island communicates with all other islands, regardless of the cluster.

Two series of experiments have been conducted:

- **Single Adaptation**
  
  In the first series of adaptation experiments, the DGA is run with a fixed random gen-
Table 6.3: Comparison of single topology adaptation performance for the F101 benchmark problem: final mean and standard deviation of 50 independent runs.

<table>
<thead>
<tr>
<th>Topology</th>
<th>Mean fitness</th>
<th>Optimal fitness</th>
</tr>
</thead>
<tbody>
<tr>
<td>Diverse</td>
<td>0.79 ± 0.072</td>
<td>0.89 ± 0.077</td>
</tr>
<tr>
<td>Fully connected</td>
<td>0.8 ± 0.053</td>
<td>0.88 ± 0.081</td>
</tr>
<tr>
<td>Mixed</td>
<td>0.8 ± 0.064</td>
<td>0.92 ± 0.076</td>
</tr>
<tr>
<td>Similar</td>
<td>0.74 ± 0.049</td>
<td>0.89 ± 0.082</td>
</tr>
</tbody>
</table>

Table 6.4: Comparison of continuous topology adaptation performance for the F101 benchmark problem: final mean and standard deviation of 50 independent runs.

<table>
<thead>
<tr>
<th>Topology</th>
<th>Mean fitness</th>
<th>Optimal fitness</th>
</tr>
</thead>
<tbody>
<tr>
<td>Diverse</td>
<td>0.8 ± 0.077</td>
<td>0.89 ± 0.082</td>
</tr>
<tr>
<td>Fully connected</td>
<td>0.8 ± 0.076</td>
<td>0.89 ± 0.082</td>
</tr>
<tr>
<td>Mixed</td>
<td>0.81 ± 0.074</td>
<td>0.92 ± 0.082</td>
</tr>
<tr>
<td>Similar</td>
<td>0.73 ± 0.049</td>
<td>0.89 ± 0.081</td>
</tr>
</tbody>
</table>

6.3.4.2 Results

Figure 6.3 shows the average and optimal performance of single adaptation, and the final statistics are summarised in Table 6.3. The normalised connectivity is: fully 1.0, diverse 0.8, mixed 0.1, and similar 0.07. As in the static topology case, the densely connected topologies have better average fitness and worse optimal performance. The similar topology has 7% of the connectivity of the fully connected, and performs worst in both categories. However, only a 3% increase in connectivity allows the mixed topology to improve the average fitness significantly, and to achieve the best optimal performance. The improved results can be explained thus; the clusters concentrate on exploring promising partitions in search space. The added connectivity in the mixed model is not big enough to force premature convergence, while still allowing information exchange between the clusters.
Figure 6.3: Optimal (above) and mean fitness (below) of single topology adaptation for the F101 benchmark problem.
Figure 6.4: Optimal (above) and mean fitness (below) of continuous topology adaptation for the F101 benchmark problem.
Table 6.5: Comparison of dynamic topology adapters: mean and optimal fitness of F101 at generation 100.

<table>
<thead>
<tr>
<th>Paper</th>
<th>Mean fitness</th>
<th>Optimal fitness</th>
</tr>
</thead>
<tbody>
<tr>
<td>Harter</td>
<td>0.69</td>
<td>0.88</td>
</tr>
<tr>
<td>this work</td>
<td>0.8</td>
<td>0.92</td>
</tr>
</tbody>
</table>

Figure 6.4 shows the average and optimal performance of continuous adaptation. The results can be directly compared with the Harter study [60] since both studies use the same GA parameter settings. Table 6.5 shows that the migration topology adapter with mixed adaptation outperforms Harter’s best dynamic adapter both in mean and optimal fitness. While Harter failed to use island performance feedback to improve performance, the migration topology adapter has similar or better performance than the best static topology, with much reduced connectivity. The normalised connectivity is: fully 1.0, diverse 0.8 (on average, but oscillating wildly between 0.6 and 1.0), mixed 0.09, and similar 0.07. The results of continuous adaptation are similar to the single adaptation, but the improved quality of mixed adaptation is even more pronounced.

The continuous adaptation also improves the performance of the similar topology, so that it produces better results than the diverse topology. In single adaptation some islands will be isolated from each other, while the continuous adaptation enables the islands to change cluster group and makes migration between all islands possible.

An examination of the number of clusters formed shows that the mixed topology keeps the number of clusters up, compared to other dynamic topologies, thus dividing the islands into smaller groups and slowing down information exchange between the clusters. This seems to provide a good balance between exploration and exploitation, which is the key to good GA design [80]. This also explains the fairly slow mean fitness convergence seen in Figure 6.4. While the mixed topology quickly finds high quality solutions in some islands groups, the reduced connectivity means that it takes longer time before these solutions reach other groups. This behaviour improves diversity and makes the DGA less prone to premature convergence, but it also slows the mean fitness convergence for the whole DGA.
### 6.3.5 Other Experiments

The same set of experiments reported in on F101 have also been carried out with the F102 and F8F2 benchmarks from the same test collection. All experiments used ring topologies within the clusters. The results for continuous adaptation, using the same parameters as the F101 experiments, are summarised in Table 6.6.

For optimal performance the the observed behaviour for the F101 problem also applies for the other test problems, with the mixed topology performing best. The mean performance of mixed topology is somewhat low, which suggests that F8F2, and especially F102, are harder than F101. This observation agrees with the experimental results reported by Whitley et al. [123]. The reduced connectivity of the mixed topology means that although good solutions are found also for the harder problems, they do not have time to spread across the total population before the DGA is terminated at generation 200.

It is interesting to note that the mixed topology has much better optimal performance than the similar topology, despite having only a small increase in connectivity, adding more supporting evidence to the conclusion drawn in the previous section: namely that mixed adaptation is the best dynamic topology.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Adapter</th>
<th>Fitness Mean</th>
<th>Fitness Optimal</th>
<th>Connectivity</th>
</tr>
</thead>
<tbody>
<tr>
<td>F101</td>
<td>fully connected</td>
<td>0.8 ± 0.076</td>
<td>0.89 ± 0.082</td>
<td>1.0</td>
</tr>
<tr>
<td>F101</td>
<td>diverse</td>
<td>0.8 ± 0.077</td>
<td>0.89 ± 0.082</td>
<td>0.8</td>
</tr>
<tr>
<td>F101</td>
<td>mixed</td>
<td>0.81 ± 0.074</td>
<td>0.92 ± 0.082</td>
<td>0.09</td>
</tr>
<tr>
<td>F101</td>
<td>similar</td>
<td>0.73 ± 0.049</td>
<td>0.89 ± 0.081</td>
<td>0.06</td>
</tr>
<tr>
<td>F101</td>
<td>uni-ring</td>
<td>0.80 ± 0.068</td>
<td>0.89 ± 0.078</td>
<td>0.07</td>
</tr>
<tr>
<td>F102</td>
<td>fully connected</td>
<td>0.75 ± 0.033</td>
<td>0.94 ± 0.037</td>
<td>1.0</td>
</tr>
<tr>
<td>F102</td>
<td>diverse</td>
<td>0.71 ± 0.033</td>
<td>0.94 ± 0.04</td>
<td>0.5</td>
</tr>
<tr>
<td>F102</td>
<td>mixed</td>
<td>0.70 ± 0.028</td>
<td>0.96 ± 0.036</td>
<td>0.08</td>
</tr>
<tr>
<td>F102</td>
<td>similar</td>
<td>0.64 ± 0.021</td>
<td>0.92 ± 0.031</td>
<td>0.06</td>
</tr>
<tr>
<td>F102</td>
<td>uni-ring</td>
<td>0.67 ± 0.025</td>
<td>0.93 ± 0.04</td>
<td>0.07</td>
</tr>
<tr>
<td>F8F2</td>
<td>fully connected</td>
<td>0.76 ± 0.028</td>
<td>0.944 ± 0.04</td>
<td>1.0</td>
</tr>
<tr>
<td>F8F2</td>
<td>diverse</td>
<td>0.62 ± 0.032</td>
<td>0.947 ± 0.037</td>
<td>0.6</td>
</tr>
<tr>
<td>F8F2</td>
<td>mixed</td>
<td>0.74 ± 0.022</td>
<td>0.954 ± 0.039</td>
<td>0.09</td>
</tr>
<tr>
<td>F8F2</td>
<td>similar</td>
<td>0.6 ± 0.026</td>
<td>0.948 ± 0.032</td>
<td>0.06</td>
</tr>
<tr>
<td>F8F2</td>
<td>uni-ring</td>
<td>0.56 ± 0.031</td>
<td>0.95 ± 0.037</td>
<td>0.07</td>
</tr>
</tbody>
</table>

Table 6.6: Summary of benchmark testing on the migration topology adapter. The reported values are averaged over 50 runs, each running for 200 generations. The table reports the normalised connectivity (see Section 6.3.2).
6.3.6 Comparison with Random Topology

Experiments have been carried out to verify that the clustering process itself, rather than the level of connectivity alone, adds to the good performance of the dynamic topologies. Figure 6.5 shows a comparison of the mixed topology and a topology which has been generated at random, but designed to have the same connectivity. The experiments run for 400 generations, and the results are averaged over 100 independent runs. The mixed topology, which uses clustering information to connect the clusters, has better performance with a final result of $0.925 \pm 0.08$. The random topology has a final result of $0.905 \pm 0.06$. Since the number of independent runs is large a student t-test was used to compare the topologies. The optimal performance of the mixed topology is statistically significantly better on a 95% confidence interval, with a two-tailed P value that equals 0.0469.

6.4 Summary

This chapter has presented an adaptive migration topology optimiser for Internet-based island model GAs. Benchmark testing was used to evaluate different migration design options, and to compare adaptive and static topology performance.

Several strategies for building dynamic migration topologies from clustering information were introduced and compared in the experiments. The mixed topology, which focuses migration within the clusters, but adds single connection between them, was found to perform best, and even to outperform fully connected topologies in terms of optimal solution quality. I think the reason for its success is that it encourages exploitation by connecting islands working in a similar section of the search space, while still maintaining a limited influence of outside material which may help the GA to avoid getting stuck. Since the clusters, and the migration topologies, are regularly updated the islands are allowed to change partners as their search states grow different from their former partner.

The best approach was found to the mixed topology adaptation. The experiments show that the system is able to achieve good quality with low levels of connectivity, making it suitable for Internet GAs.
Figure 6.5: Optimal (above) and mean fitness (below) comparison in continuous mixed and random topologies for the F101 benchmark problem.
In future research, I hope to extend on the current system and investigate the applicability of the proposed approach on optimisation of other parameters that have an impact on communication overhead, such as migration rate and interval. Another interesting extension to the current research would be to investigate the influence of different migration topology adaptation strategies during different stages of the search.
Chapter 7

Adaptation of the Population Size and the Number of Islands

This chapter presents a population size adapter that automatically searches for a good combination of the number of islands, and number of individuals on each island, in distributed GAs \(^1\). This population size adapter can be used by itself, or in combination with the migration topology adapter previously described in Chapter 6.

7.1 Introduction

The performance of a GA depends on a number of parameters including representation, crossover and mutation rates, population size, and selection pressure. For efficiency reasons it is often advantageous to parallelise and distribute the GA over several processors, which typically introduces even more parameters, such as the number of islands, and migration parameters. Determining the proper parameter set is a non-trivial task which depends on the nature of the problem, its representation, and the GA operators.

\(^1\)This method has earlier been presented in part at the Genetic and Evolutionary Computation Conference, Washington DC, USA, 2005 [16].
tolerant, while a proper population size is vital and can to some extent overcome sub-optimal settings of other parameters [113, 59]. This makes population sizing one of the most important questions facing the GA practitioner. The existing theoretical work on population sizing is essentially based on Goldberg’s building block-based population sizing model [54]. This approach focuses on ensuring BB supply, and making a correct decision between a BB and its closest competitor in a given partition. The decision process takes place in the presence of noise that comes from the other partitions. The probability of making a bad decision is a function of the population size, and by increasing the size it is possible to make this decision error as small as possible. This model is hard to apply in practice since it assumes knowledge on problem characteristics that are hard to assess for realistic real world problems, such as the selective advantage of a building block compared to its closest competitor, and the fitness variance of the building block. Many practitioners resort to trying various parameter combinations to tune the GA to good performance. This approach obviously requires time and computational effort which sometimes can exceed the cost of solving the actual problem itself. An alternative approach is to use adaptive methods that adjust the parameters according to observed performance during the run of the GA [62].

The main contribution of this chapter the formulation of an adaptive technique that automatically searches for a good combination of the number of islands, and the population size on each island, in distributed GAs. The term population adapter will be used to denote the proposed technique. This population adapter is applicable to a wide range of GA algorithms, and makes DGAs easier to use by reducing the number of user-set parameters and aiding in construction of robust GA applications with good performance.

The remainder is organised as follows. Section 7.2 describes the proposed method, and Section 7.3 presents experiments showing how the population adapter works on typical GA problems. Finally, the results and contributions are summarised in Section 7.4.

7.2 Proposed Method

Finding the proper population size for a given GA problem is of crucial importance for good performance, and the intention of the proposed method is to automatically find a set
of values for the number of islands and island sizes that provides a total population size big enough to find solutions of similar or better quality than those found by a hand-tuned algorithm. If the population size or number of islands are too small the GA will converge prematurely, and too large values waste computational resources. A manual approach to population sizing could be to first try with a small population and number of islands, and then increase each parameter until no further gain in performance is detected. The proposed method works in a similar manner, but instead of working in an ad-hoc way, the population adapter automates the process and hides it from the user.

The basic idea of the population adapter is to run several distributed GAs with competing sets of number of islands and island population size settings in parallel. Let $n$ denote the number of islands, and $d$ denote the population size on each island. The total population is $n \times d$. At any given time we run three DGAs (denoted $DGA_0$, $DGA_1$, and $DGA_2$) in parallel with the following set of parameters:

$$< n, d >, < n, 2d >, < 2n, d >$$

This sets up a competition between the basic $DGA_0$, $DGA_1$ with more islands, and $DGA_2$ with bigger island population sizes. There are no upper bounds on the population size or number of islands, and the DGAs are will run until one of the DGAs overtake its competitors, or the DGAs termination criterion has been met ².

A similar approach is used by the parameter-less GA [59], which is intended to simplify the application of GAs by automating the experimentation process. The algorithm sets crossover and selection pressure to fixed values that satisfy the schema theorem, and uses a competition between GAs with different population sizes to find one big enough to achieve good performance. The smaller population is eliminated and new competition started with bigger population sizes if the bigger GA overtakes the smaller GA.

Compared to the parameter-less GA, the population adapter proposed here makes sig-

²In the early stages of the research I experimented with restricting the range in which these parameters are allowed to vary relative to each other in order to avoid scenarios with DGAs with very big populations and few islands, or very small populations and lots of islands. However, I found that the adapter automatically alternates between increasing the population size and the number of islands, and that explicit range checks are not needed.
significant new contributions:

- Both population size, and number of islands are adjusted in parallel.

The parameter-less GA is designed for sequential GAs only, and uses competition between GAs with different population sizes to find one big enough to achieve good performance. The population adapter instead uses a competition between DGAs that have both different population size, and different number of islands.

- The population adapter is both competitive and collaborative.

Section 7.2.1 below gives a detailed description of how the DGAs compete. When a DGA is found to be inferior, it is restarted with new parameters. In this work a collaborative approach to restarting is suggested, which inserts elite individuals in the new DGA to give new DGAs a bias toward promising regions. This technique, also known as seeding, is described in Section 7.2.2.

- The population adapter terminates automatically when no further improvements are found.

The parameter-less GA continues to adapt the population size until it is manually halted by the user. The population adapter automatically stops, using an algorithm that draws upon both adapter search state information and problem-dependent termination criteria. This algorithm is described in Section 7.2.3.

### 7.2.1 Competitive Evaluation

The evaluation of a DGA can be halted if it is being overtaken by other DGAs, or converging. Overtaking is detected by comparing average fitness of a DGA with DGAs with bigger total population size, since it is unlikely that the smaller DGA with lower average fitness will succeed in getting better optimal results than the bigger DGA. Once convergence or overtaking is detected, the population adapter takes the following action:
• $DGA_0$ converges
  
  No action.

• $DGA_1$ and $DGA_2$ converged
  
  All DGAs restarted.

• $DGA_1$ overtakes $DGA_0$
  
  $d = d \times 2, DGA_1 \rightarrow DGA_0, DGA_1$ and $DGA_2$ restarted (using seeding).

• $DGA_2$ overtakes $DGA_0$
  
  $n = n \times 2, DGA_2 \rightarrow DGA_0, DGA_1$ and $DGA_2$ restarted (using seeding).

The intuition behind this algorithm is that three hypotheses are evaluated in parallel, and when a DGA overtakes its competitors, it is an indication of the need to adjust the parameter set in the direction suggested by the winning DGA. $DGA_1$ is testing if more islands are beneficial, and when it overtakes the other DGAs, the routine increases the number of islands in the next round of competitions. $DGA_2$, which is testing the benefit of increasing the population size on each island, works in the same way.

One potential problem is that the population adapter may overshoot. If $n = 125$ is the theoretically best value island population size for a given problem, then the population adapter is likely to find $n = 80$ or $n = 160$ since $n$ is doubled each time it is incremented. Although another constant than 2 is possible, $n$ is still likely not to be able to hit the exact optimal value. It is possible to add backtracking to the population adapter so that if $n = 160$ is found in the example above, the population adapter tries smaller population sizes until the smallest $n$ solving the problem it found. However, initial experiments with backtracking show that it requires a big increase in computational effort. Since a DGA with a too big population is likely to terminate the population size adapter immediately, trading the limited inefficiency of the original adapter for certain backtracing overhead appears not to be a promising approach and backtracking has not been investigated further in this work.
7.2.2 Collaborative Restart

The basic case of restarting a DGA is to simply reinitiate the population on each island. This may be inefficient, since each newly restarted DGA will need time to catch up with its competitors even if its population parameter settings are better. As an alternative the population adapter can use seeding. With seeding, each island in the newly restarted DGA reinitiates its population, but also inserts the best individuals from the other DGAs. For example, if $DGA_1$ is restarted, each island’s initial population includes the best individual from $DGA_0$ and $DGA_2$, respectively. The other $n - 2$ individuals are randomly generated. In this way, the new DGAs are biased toward promising regions of the search space, which makes the population adapter more efficient.

7.2.3 Termination

The population adapter should terminate the DGAs when there is a relatively slim chance of finding a better set of sizing parameters than the current set. This termination criterion in the population adapter should not depend on problem specific criterion. Rather, termination uses the status of each DGA, and terminates when all DGAs have converged. Since it is possible that DGAs converge in early trials because of the small values of $n$ and $d$, it is not possible to terminate as soon as a convergence has been detected. The population adapter therefore does not terminate until (i) each DGA has converged at least once, and (ii) no new best solution is found during the detection phase. If a new best solution is detected, then the best solution so far is updated and the termination detection process is reset.

Problem specific termination criterion may be specified in the GAs running on each island, by adding them into the problem initiation code. In the experiments reported in Section 7.3, the global optima is known and the population adapter terminated once it has been found.
7.2.4 Putting It All Together

The DGAs run in parallel with the population adapter, which uses the average population fitness and convergence status of each DGA to update the $n$ and $d$ parameters, and restart DGAs as needed. This is illustrated by the pseudo-code in Algorithm 6. The initial values for $n$ and $d$ are small, typically $n = 5$ and $d = 1$.

**Algorithm 6** Pseudo-code for the population adapter

```
set n and d to small initial values
convergenceCounter=0
DGA(0)=<n, d>
DGA(1)=<n, d*2>
DGA(2)=<n*2, d>
Run DGA(0..2) in parallel
do
    Get DGA(0..2) mean fitness and convergence status
    If new optima found
        Reset termination detection: convergenceCounter=0
    If DGA(1) and DGA(2) have converged
        ++convergenceCounter
        Restart DGA(0..2)
    If DGA(1) overtakes the other DGAs
        increase the number of islands: d=d*2
        DGA(0)=DGA(1)
        Restart DGA(1)
        Restart DGA(2)
    If DGA(2) overtakes the other DGAs
        increase the population size: n=2*n
        DGA(0)=DGA(2)
        Restart DGA(1)
        Restart DGA(2)
until optima found or convergenceCounter>1
```

7.3 Experiments

This section presents experiments which have been conducted to evaluate the population adapter, both as a stand-alone feature, and in combination with the migration topology adapter described in Chapter 6.
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>K</td>
<td>4</td>
<td>Number of levels</td>
</tr>
<tr>
<td>B</td>
<td>8</td>
<td>Number of bits in each block</td>
</tr>
<tr>
<td>G</td>
<td>7</td>
<td>Number of bits in each gap</td>
</tr>
<tr>
<td>V</td>
<td>0.02</td>
<td>Part reward/penalty</td>
</tr>
<tr>
<td>M*</td>
<td>4</td>
<td>Required number of 1 bits in Part calculation</td>
</tr>
<tr>
<td>U</td>
<td>0.3</td>
<td>Bonus reward</td>
</tr>
<tr>
<td>U*</td>
<td>1.0</td>
<td>Reward for first completed block in a level</td>
</tr>
</tbody>
</table>

Table 7.1: Fitness calculation parameters for the Royal Road function

In the experiments all parameters are fixed except for the population size and the number of islands. The methods were implemented on the G2DGA framework and run on a simulated homogeneous 100MB LAN network. The experiments are run for 50 independent runs using the same random seed sequences. For ease of comparison of the experiments, their main statistics (number of evaluations, population size) are reported as mean and standard deviation over all runs. However, the calculation of statistical significance is done using the Wilcoxon Rank Sum test [87] which does not rely on the normal distribution assumption.

7.3.1 Test Problems

It is possible to devise particular problems that are suited to particular search methods, including various types of GAs. An implication of this is that, if a particular form of algorithm is good at solving a particular problem, this does not necessarily mean that it is good for other problems. It is thus extremely difficult to make any definitive statements about the merit of a particular algorithm unless it is tested for several types of problems. For the evaluation of G2DGA’s adaptation methods I use two benchmark problems that have been widely used in the literature; Holland’s Royal Road problem, and the F101 benchmark problem. These problems are designed to be challenging for GAs, but are also quite different in character, with Royal Road being focused on building block discovery and utilisation, while F101 is multi-modal with non-linear interaction between its parameters.
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>elitism</td>
<td>1 individual</td>
</tr>
<tr>
<td>GA</td>
<td>Generational</td>
</tr>
<tr>
<td>crossover</td>
<td>two point, pc=0.9</td>
</tr>
<tr>
<td>mutation</td>
<td>bit flip, pm=0.001</td>
</tr>
<tr>
<td>convergence</td>
<td>no new optima in 20 generations, or optima found</td>
</tr>
<tr>
<td>migration topology</td>
<td>uni-ring</td>
</tr>
<tr>
<td>migration interval</td>
<td>5 generations</td>
</tr>
<tr>
<td>migration rate</td>
<td>1 individual (best replaces worst)</td>
</tr>
</tbody>
</table>

Table 7.2: Royal Road GA parameters

### 7.3.1.1 Royal Road

The Royal Road function is designed to be difficult for hill-climbing algorithms to solve due to having a large number of local optima, but to provide a GA with an easy path toward convergence by explicitly rewarding the combination of pre-defined building blocks. However, it was early found that the Royal Road function caused unexpected difficulties for GAs which makes it a suitable test problem. To see this, it should be remembered that according to the building block hypothesis, once building blocks are identified it should be possible for a GA to combine these into larger sequences to eventually find the optimum. However, Mitchell et al. showed that an individual with good building blocks will have more offspring related to itself containing copies not only of the good building block, but poor ones as well [82]. Therefore, an increased supply of good building blocks at one locus could decrease the building block supply at other loci, making the problem prone to premature convergence when the population size is small. Since the solution quality depends so much on the population size, it is a good test problem for the population adapter.

I use the Royal Road function described in [66], with the default settings listed in Table 7.1. The input to the Royal Road function is a binary string of $2^K$ blocks, called level-0 blocks, of length $B + G$. In the fitness calculation only the first $B$ positions (block bits) in each block are considered, and the last $G$ positions (gap bits) ignored. A level-0 block is complete if all its block bits are set to 1. Level-0 blocks can be combined in higher level blocks, such that a level-$i$ block consists of even-odd numbered pairs of level-$(i − 1)$ blocks. A level-$i$ block is complete if both its level-$(i − 1)$ are complete.

The fitness is calculated in two steps: Part and Bonus. The Part calculation rewards
the level-0 blocks using two parameters, $M^*$ and $V$. Let $j$ be the number of 1s in the $B$ block bits for a level-0 block $k$. The Part contribution of this block is:

$$p(i) = \begin{cases} 
    j \cdot V & : j \leq M^* \\
    (M^* - j) \cdot V & : M^* < j < B \\
    0 & : j = B
\end{cases}$$

The Bonus calculation rewards the number of completed blocks at each level using two parameters $U^*$ and $U$. Let $j$ be the number of completed blocks at level $i$. The Bonus contribution of this level is:

$$b(i) = \begin{cases} 
    U^* + (j - 1) \cdot U & : j > 0 \\
    0 & : j = 0
\end{cases}$$

The total fitness is the sum of Part and Bonus contributions over all blocks and levels:

$$f = \sum_{i=0}^{2^K} p(i) + \sum_{i=0}^{K} b(i)$$

The GA parameters used for the Royal Road experiments are listed in Table 7.2. A run is considered successful is the optimal value of 12.8 is found.
7.3.1.2 F101

F101 from the Whitley test suite [123] is designed to be difficult for a GA to solve. It is non symmetric with many local optima, and scalable with nonlinear interactions between the variables:

\[ F_{101}(x, y) = -x \sin \sqrt{\left|x - \frac{y + 47}{2}\right|} - (y + 47) \sin \sqrt{\left|y + 47 + \frac{x}{2}\right|}, \quad x, y \in [-512, 511] \]

F101 is not separable and hard to solve with hill-climbing optimisation techniques, and it can be scaled to \( n \) variables using this expansion:

\[ F(x_1, x_2 \cdots x_n) = \frac{F_{101}(x_1, x_2) + F_{101}(x_2, x_3) \cdots F_{101}(x_n, x_1)}{n} \]

The parameters used for the F101 experiments are listed in Table 7.3. A run is considered successful if a solution is found within 1% of the global optima.

7.3.2 Manual Population Sizing

To provide a reference for the population adapter experiments, experiments with manual settings of the number of islands and the population size on each island were run. The outcome for static (non-adaptive) manual sizing is displayed in Table 7.4 for F101 and Table 7.6 for Royal Road. These results suggest that a total population size of 1280 for F101 and 2560 for Royal Road is required for solving the problem reliably, and bigger island population sizes give a lower total number of function evaluations.

A series of experiments were also run with migration topology adaptation enabled to verify that it will not interfere negatively on the working of the population adapter. For this purpose, various population sizes where tried, and the outcome is summarised in Table 7.5 for F101, and Table 7.7 for Royal Road. Comparing the two tables shows that the DGA with migration topology adaptation performs better than the non-adaptive DGA in terms of success rate, mean population fitness, and number of evaluations. Since this trend is consistent for both problem types and over all population sizes, including sizes that are far
<table>
<thead>
<tr>
<th>Population size</th>
<th>Success</th>
<th>Evaluations</th>
<th>Average fitness</th>
<th>Optimal Fitness</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>total</td>
<td>n, d</td>
<td>mean</td>
<td>stddev</td>
</tr>
<tr>
<td>160</td>
<td>20, 8</td>
<td>3/20</td>
<td>18381</td>
<td>4479</td>
</tr>
<tr>
<td>320</td>
<td>20,16</td>
<td>5/20</td>
<td>43300</td>
<td>1005</td>
</tr>
<tr>
<td>640</td>
<td>20,32</td>
<td>8/20</td>
<td>86151</td>
<td>19814</td>
</tr>
<tr>
<td>40,16</td>
<td>9/20</td>
<td>63548</td>
<td>23562</td>
<td>0.79578</td>
</tr>
<tr>
<td>1280</td>
<td>20,64</td>
<td>5/20</td>
<td>168660</td>
<td>21440</td>
</tr>
<tr>
<td>40,32</td>
<td>40,16</td>
<td>9/20</td>
<td>71767</td>
<td>25601</td>
</tr>
<tr>
<td>80,16</td>
<td>20/20</td>
<td>57374</td>
<td>98132</td>
<td>0.77541</td>
</tr>
<tr>
<td>160,8</td>
<td>20/20</td>
<td>54046</td>
<td>14174</td>
<td>0.83598</td>
</tr>
<tr>
<td>320,4</td>
<td>16/20</td>
<td>74940</td>
<td>58359</td>
<td>0.84717</td>
</tr>
</tbody>
</table>

Table 7.4: Manual sizing with the F101 benchmark, without migration topology adaptation.

<table>
<thead>
<tr>
<th>Population size</th>
<th>Success</th>
<th>Evaluations</th>
<th>Average fitness</th>
<th>Optimal Fitness</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>total</td>
<td>n, d</td>
<td>mean</td>
<td>stddev</td>
</tr>
<tr>
<td>160</td>
<td>20, 8</td>
<td>4/20</td>
<td>17601</td>
<td>4672.7</td>
</tr>
<tr>
<td>320</td>
<td>20,16</td>
<td>11/20</td>
<td>27653</td>
<td>10232</td>
</tr>
<tr>
<td>640</td>
<td>20,32</td>
<td>11/20</td>
<td>60635</td>
<td>19857</td>
</tr>
<tr>
<td>40,16</td>
<td>10/20</td>
<td>55247</td>
<td>23082</td>
<td>0.82343</td>
</tr>
<tr>
<td>1280</td>
<td>20,64</td>
<td>20/20</td>
<td>82312</td>
<td>15134</td>
</tr>
<tr>
<td>40,32</td>
<td>20/20</td>
<td>58400</td>
<td>8374.9</td>
<td>0.7968</td>
</tr>
<tr>
<td>80,16</td>
<td>20/20</td>
<td>51847</td>
<td>8598.5</td>
<td>0.79609</td>
</tr>
<tr>
<td>160,8</td>
<td>20/20</td>
<td>49450</td>
<td>8592.8</td>
<td>0.83969</td>
</tr>
<tr>
<td>320,4</td>
<td>17/20</td>
<td>69599</td>
<td>41802</td>
<td>0.83861</td>
</tr>
</tbody>
</table>

Table 7.5: Manual sizing with the F101 benchmark using migration topology adaptation.

It is too small to solve the problem reliably, it seems that the migration topology adaptation only has beneficial impact on the population sizing problem.

### 7.3.3 Population Adapter

Table 7.8 shows the outcome of 50 runs of the population adapter, using the F101 benchmark. The population adapter successfully finds the optimal value for all runs, but there are clear differences in the number of evaluations, depending on whether seeding is used or not. This is reflected in the number of evaluations, where seeding has a lower average number of generations (and thus total number of evaluations) needed, suggesting that seeding leads to more reliable results. Seeding also gives a higher mean and lower standard deviation of the population size found by the adapter, suggesting that the seeding helps the adapter to quickly scale to the size needed to solve the problem.
<table>
<thead>
<tr>
<th>Population size</th>
<th>Total</th>
<th>n, d</th>
<th>Success</th>
<th>Evaluations</th>
<th>Average fitness</th>
<th>Optimal Fitness</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>mean</td>
<td>stddev</td>
<td>mean</td>
</tr>
<tr>
<td>160</td>
<td>0/20</td>
<td>9211.7</td>
<td>1017.7</td>
<td>4.4973</td>
<td>1.2302</td>
<td>4.634</td>
</tr>
<tr>
<td>320</td>
<td>0/20</td>
<td>19093</td>
<td>2047.5</td>
<td>4.7951</td>
<td>0.9204</td>
<td>6.1</td>
</tr>
<tr>
<td>640</td>
<td>0/20</td>
<td>37399</td>
<td>2347.6</td>
<td>4.5733</td>
<td>3.7425</td>
<td>7.285</td>
</tr>
<tr>
<td>1280</td>
<td>0/20</td>
<td>40076</td>
<td>2126.3</td>
<td>7.5776</td>
<td>0.61925</td>
<td>10.234</td>
</tr>
<tr>
<td>160</td>
<td>1/20</td>
<td>78306</td>
<td>2029.8</td>
<td>4.9144</td>
<td>0.2525</td>
<td>8.794</td>
</tr>
<tr>
<td>320</td>
<td>1/20</td>
<td>81180</td>
<td>5466.3</td>
<td>7.4248</td>
<td>0.61003</td>
<td>11.441</td>
</tr>
<tr>
<td>640</td>
<td>3/20</td>
<td>70716</td>
<td>9127.5</td>
<td>9.0226</td>
<td>0.63078</td>
<td>12.27</td>
</tr>
<tr>
<td>1280</td>
<td>8/20</td>
<td>67878</td>
<td>13890</td>
<td>9.8395</td>
<td>0.94033</td>
<td>11.396</td>
</tr>
<tr>
<td>2560</td>
<td>8/20</td>
<td>57376</td>
<td>12997</td>
<td>9.7917</td>
<td>0.96781</td>
<td>11.517</td>
</tr>
</tbody>
</table>

Table 7.6: Manual sizing with the Royal Road problem, without migration topology adaptation.

<table>
<thead>
<tr>
<th>Population size</th>
<th>Total</th>
<th>n, d</th>
<th>Success</th>
<th>Evaluations</th>
<th>Average fitness</th>
<th>Optimal Fitness</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>mean</td>
<td>stddev</td>
<td>mean</td>
</tr>
<tr>
<td>160</td>
<td>0/20</td>
<td>9319.3</td>
<td>969.74</td>
<td>4.6907</td>
<td>1.0678</td>
<td>4.867</td>
</tr>
<tr>
<td>320</td>
<td>0/20</td>
<td>19992</td>
<td>2190.3</td>
<td>5.242</td>
<td>1.2615</td>
<td>7.002</td>
</tr>
<tr>
<td>640</td>
<td>3/20</td>
<td>38641</td>
<td>1440.4</td>
<td>4.9474</td>
<td>0.43615</td>
<td>9.181</td>
</tr>
<tr>
<td>40,16</td>
<td>0/20</td>
<td>42972</td>
<td>2335.6</td>
<td>8.0777</td>
<td>0.55458</td>
<td>10.319</td>
</tr>
<tr>
<td>1280</td>
<td>8/20</td>
<td>82180</td>
<td>1324.1</td>
<td>5.2262</td>
<td>0.26019</td>
<td>10.681</td>
</tr>
<tr>
<td>40,32</td>
<td>17/20</td>
<td>84151</td>
<td>4017.9</td>
<td>8.064</td>
<td>0.30642</td>
<td>12.482</td>
</tr>
<tr>
<td>80,16</td>
<td>19/20</td>
<td>67996</td>
<td>9041.3</td>
<td>8.9363</td>
<td>0.67057</td>
<td>12.694</td>
</tr>
<tr>
<td>160,8</td>
<td>12/20</td>
<td>61486</td>
<td>13958</td>
<td>9.5107</td>
<td>1.0014</td>
<td>11.805</td>
</tr>
<tr>
<td>320,4</td>
<td>11/20</td>
<td>53997</td>
<td>12651</td>
<td>9.6406</td>
<td>0.94011</td>
<td>11.835</td>
</tr>
<tr>
<td>2560</td>
<td>5/20</td>
<td>157340</td>
<td>7079</td>
<td>6.9369</td>
<td>0.38461</td>
<td>11.199</td>
</tr>
<tr>
<td>80,32</td>
<td>19/20</td>
<td>134980</td>
<td>18616</td>
<td>8.63</td>
<td>0.59502</td>
<td>12.694</td>
</tr>
<tr>
<td>160,16</td>
<td>20/20</td>
<td>110930</td>
<td>16472</td>
<td>8.7025</td>
<td>0.60802</td>
<td>12.8</td>
</tr>
</tbody>
</table>

Table 7.7: Manual sizing with the Royal Road problem using migration topology adaptation.

<table>
<thead>
<tr>
<th>Seeding</th>
<th>Success</th>
<th>Evaluations</th>
<th>Population size</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mean</td>
<td>Stddev</td>
<td>Mean</td>
</tr>
<tr>
<td>No</td>
<td>50/50</td>
<td>113056</td>
<td>76760</td>
</tr>
<tr>
<td>Yes</td>
<td>50/50</td>
<td>99037</td>
<td>75735</td>
</tr>
</tbody>
</table>

Table 7.8: Population adapter with the F101 benchmark.

<table>
<thead>
<tr>
<th>Seeding</th>
<th>Success</th>
<th>Evaluations</th>
<th>Population size</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mean</td>
<td>Stddev</td>
<td>Mean</td>
</tr>
<tr>
<td>No</td>
<td>40/50</td>
<td>279478</td>
<td>78248</td>
</tr>
<tr>
<td>Yes</td>
<td>46/50</td>
<td>198137</td>
<td>75547</td>
</tr>
</tbody>
</table>

Table 7.9: Population adapter with the Royal Road problem.
Table 7.10: Population adapter with the F101 benchmark, combined with migration topology adaptation.

<table>
<thead>
<tr>
<th>Seeding</th>
<th>Success</th>
<th>Evaluations</th>
<th>Population size</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Mean Stddev</td>
<td>Mean Stddev</td>
</tr>
<tr>
<td>No</td>
<td>50/50</td>
<td>98071 71378</td>
<td>348 229</td>
</tr>
<tr>
<td>Yes</td>
<td>50/50</td>
<td>81828 67054</td>
<td>397 274</td>
</tr>
</tbody>
</table>

Table 7.11: Population adapter with the Royal Road problem, combined with migration topology adaptation.

The same behaviour is found when using the Royal Road problem (see Table 7.9). The adapter finds larger population sizes, which was predicted by the manual sizing experiments previously. Some runs converge on a sub-optimal value, which is probably due to the difficulty of the problem (it is designed to be deceptive and difficult for standard GAs to solve) and the convergence criterion used in the experiments. It is interesting to note that, like in F101, seeding improves the robustness and performance, giving less sub-optimal runs.

The population sizes found are smaller than the ones found to be needed for reliable performance in manual experiments. This is due to the nature of the population adapter, where three DGAs are run in parallel and restarted if they are stuck or found to perform badly. In practice, this means that the population adapter gets more chances to get it right, even if the population size itself is too small to guarantee success. Compared to the manual experiments, the population adapter uses about two to three times the number of evaluations, since some effort will be wasted in DGAs that loose to their competitors.

7.3.4 Combined Population and Migration Topology Adaptation

Another series of experiments were conducted where the migration topology adaptation was enabled in the competing DGAs. The migration topology adaptation was shown in previous chapters to improve performance of the DGA, and it is expected that adding migration topology adaptation will improve the performance of the population adapter as well.
<table>
<thead>
<tr>
<th>Problem</th>
<th>Experiment 1</th>
<th>Experiment 2</th>
<th>Winner</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>RR</td>
<td>no seed</td>
<td>seed</td>
<td>seed</td>
<td>4.2809</td>
</tr>
<tr>
<td>RR</td>
<td>no seed + topology</td>
<td>seed + topology</td>
<td>seed + topology</td>
<td>3.3406</td>
</tr>
<tr>
<td>RR</td>
<td>no seed</td>
<td>no seed + topology</td>
<td>-</td>
<td>1.1385</td>
</tr>
<tr>
<td>RR</td>
<td>seed</td>
<td>seed + topology</td>
<td>-</td>
<td>0.3852</td>
</tr>
<tr>
<td>RR</td>
<td>no seed</td>
<td>seed + topology</td>
<td>seed + topology</td>
<td>3.8205</td>
</tr>
<tr>
<td>F101</td>
<td>no seed</td>
<td>seed</td>
<td>-</td>
<td>1.1547</td>
</tr>
<tr>
<td>F101</td>
<td>no seed + topology</td>
<td>seed + topology</td>
<td>-</td>
<td>1.5821</td>
</tr>
<tr>
<td>F101</td>
<td>no seed</td>
<td>no seed + topology</td>
<td>-</td>
<td>0.9755</td>
</tr>
<tr>
<td>F101</td>
<td>seed</td>
<td>seed + topology</td>
<td>-</td>
<td>0.8582</td>
</tr>
<tr>
<td>F101</td>
<td>no seed</td>
<td>seed + topology</td>
<td>seed + topology</td>
<td>2.3749</td>
</tr>
</tbody>
</table>

Table 7.12: Performance comparison of population adapter using Wilcoxon rank sum test. ‘seed’ means that seeding was used in the population adapter. ‘topology’ means that the migration topology adapter was activated in the DGAs. The Winner column contains the best DGA or ‘-’ if the difference is not statistically significant. The Z column is the result of the test. A Z score > 1.96 is statistically significant with 95% confidence level.

Table 7.10 shows the outcome of 50 runs of the population adapter with the migration topology adapter enabled, using the F101 benchmark. The population adapter successfully finds the optimal value for all runs, and like in the previous experiment seeding gives better performance with a lower average number of evaluations needed to find the optimal value. A comparison of the both experiments shows that the runs with combined population and migration topology adapters outperform population adaptation only.

Table 7.11 shows the outcome of 50 runs of the population adapter with the migration topology adapter enabled, using the Royal Road problem. The general conclusions from F101 apply to Royal Road as well, with the best performance found in experiments where seeding and migration topology adaptation is used.

The experiments were compared using the Wilcoxon Rank Sum test (a nonparametric statistic [87]) to see if the performance gains are statistically significant, as measured by the number of evaluations. The results are displayed in Figure 7.12. If the test produces a Z score greater than 1.96, then the results from the two experiments are considered to be significantly different with a confidence level of 95%.

The difference in performance between non-seeded population adaptation only and the combination of seeding and migration topology adaptation was found to be statistically

---

3Specifically, the `ranksum` function in Matlab 6.5 was used to calculate the Z score. See also Section 9.2.4 for a more detailed description of the Wilcoxon Rank Sum test.
significant in both the F101 and Royal Road experiments. In the Royal Road it is also clear that the seeding itself has a significant impact on performance. The F101 problem appears to be somewhat easier for the GA to solve, and the same comparison of seeding vs. non-seeding is not statistically significant.

7.3.5 Dynamic Network Environments

The experiments in the previous sections test GA performance in static networks that resemble cluster computing environments. In this section the testing is extended to heterogeneous networks that are more like Internet computing environments. In the heterogeneous environment, the computers’ computational capacity is randomly distributed in the range 1.0 - 3.0, meaning that the average computer runs at half of the speed of the fastest computer, and twice the speed of the slowest computer. In every 5 generations 5% of the islands are randomly moved to another computer. This gives a rough indication how the GA responds to dynamic events in the network, where volunteer computers may enter and leave the computation at any time.

Table 7.13 shows the outcome of 50 runs of the population adapter on static and dynamic networks, using the F101 benchmark. For each network, the GA has been tested with combinations of manual population sizing, adaptive population sizing, and migration topology adaptation. All experiments use seeding, which was shown to improve performance and robustness in Section 7.3.4 above.

The GA needs a bigger population size to search well in the dynamic network environment, since the search is distorted by the different computational capabilities of the available computers, which make some islands run faster than other islands. The manual experiments indicate while a total population size of 1280 which was sufficient in the static environment, the dynamic environment require a total population size of 2560. Apart from the bigger population size, the it is clear from the data that the general conclusions from the static environment experiments also hold true in the dynamic environment, namely that both adaptation methods contribute to improved performance in terms of success rate and total number of evaluations.
As in the static experiments, the population size adapter requires smaller population sizes than those required by the manual population size experiments, which is due to the fact that the population size adapter, by restarting multiple times, is less likely to be trapped in local optima. However, one notable difference in the total effort required by the population size adapter compared to the manual experiments. In the static experiments the population sizer needs two to three times the effort, while in the more difficult dynamic network the difference has been reduced to almost similar levels. The important observation is that while the manual population size needs to increase to counter heterogeneity and dynamic events, more volatile network environments give the adaptive methods more scope to have an impact on the performance of the GA. The results indicate that the the adapter methods are able to detect the current search state, which is influenced by the current distribution of computational resources, and able to rebuild the population distribution and migration topology dynamically.

Table 7.14 shows the outcome of the same experiments on the Royal Road problem, and the trends and conclusion are similar to the F101 experiments. The performance of the adaptive GA will be investigated further in Chapter 9, where a real world problem is used to test scalability and robustness in various network environments. This includes a more dynamic case where, in addition to islands being forced to move when a volunteer computer withdraws from the calculation, the supply of volunteer computers is also varies, so that several islands may have to share host computers when the number of volunteers is low.

7.4 Summary

This chapter has presented a population adapter for DGAs that uses an adaptive approach to find a good combination of number islands and island population sizes. The method relies on a race between DGAs using competing parameter settings, but uses a collaborative method to jump-start new competitions. Although the experiments were carried out on standard GAs, the method can be applied to more advanced GAs, such as messy GAs [55] or the Bayesian optimisation algorithm [91], without significant modifications.

The collaborative restart using seeding was shown to give more robust performance in all
<table>
<thead>
<tr>
<th>Network</th>
<th>Adapter(s)</th>
<th>Population size</th>
<th>Evaluations</th>
<th>Success rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>static</td>
<td>-</td>
<td>1280</td>
<td>51847 ± 8599</td>
<td>20/20</td>
</tr>
<tr>
<td>static</td>
<td>mig</td>
<td>1280</td>
<td>39304 ± 23279</td>
<td>20/20</td>
</tr>
<tr>
<td>static</td>
<td>pop</td>
<td>504 ± 105</td>
<td>99037 ± 75735</td>
<td>50/50</td>
</tr>
<tr>
<td>static</td>
<td>pop+mig</td>
<td>397 ± 74</td>
<td>81828 ± 67054</td>
<td>50/50</td>
</tr>
<tr>
<td>dynamic</td>
<td>-</td>
<td>1280</td>
<td>74151 ± 47074</td>
<td>42/50</td>
</tr>
<tr>
<td>dynamic</td>
<td>mig</td>
<td>1280</td>
<td>55890 ± 22214</td>
<td>46/50</td>
</tr>
<tr>
<td>dynamic</td>
<td>-</td>
<td>2560</td>
<td>101220 ± 23467</td>
<td>49/50</td>
</tr>
<tr>
<td>dynamic</td>
<td>mig</td>
<td>2560</td>
<td>96051 ± 13126</td>
<td>50/50</td>
</tr>
<tr>
<td>dynamic</td>
<td>pop</td>
<td>1036 ± 379</td>
<td>139730 ± 27549</td>
<td>50/50</td>
</tr>
<tr>
<td>dynamic</td>
<td>pop+mig</td>
<td>650 ± 269</td>
<td>108830 ± 17458</td>
<td>50/50</td>
</tr>
</tbody>
</table>

Table 7.13: Performance of adapters in different network environments using the F101 benchmark with migration topology adaptation (top) and population size adaptation (pop).

<table>
<thead>
<tr>
<th>Network</th>
<th>Adapter(s)</th>
<th>Population size</th>
<th>Evaluations</th>
<th>Success rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>static</td>
<td>-</td>
<td>2560</td>
<td>110930 ± 16472</td>
<td>20/20</td>
</tr>
<tr>
<td>static</td>
<td>mig</td>
<td>2560</td>
<td>99531 ± 9640</td>
<td>20/20</td>
</tr>
<tr>
<td>static</td>
<td>pop</td>
<td>1244 ± 284</td>
<td>198137 ± 75547</td>
<td>46/50</td>
</tr>
<tr>
<td>static</td>
<td>pop+mig</td>
<td>1345 ± 358</td>
<td>224102 ± 113749</td>
<td>50/50</td>
</tr>
<tr>
<td>dynamic</td>
<td>-</td>
<td>2560</td>
<td>114504 ± 19257</td>
<td>42/50</td>
</tr>
<tr>
<td>dynamic</td>
<td>mig</td>
<td>2560</td>
<td>116872 ± 18323</td>
<td>45/50</td>
</tr>
<tr>
<td>dynamic</td>
<td>-</td>
<td>5120</td>
<td>210296 ± 27090</td>
<td>50/50</td>
</tr>
<tr>
<td>dynamic</td>
<td>mig</td>
<td>5120</td>
<td>224444 ± 27113</td>
<td>49/50</td>
</tr>
<tr>
<td>dynamic</td>
<td>pop</td>
<td>1913 ± 1261</td>
<td>209606 ± 97725</td>
<td>50/50</td>
</tr>
<tr>
<td>dynamic</td>
<td>pop+mig</td>
<td>1363 ± 686</td>
<td>172744 ± 77843</td>
<td>50/50</td>
</tr>
</tbody>
</table>

Table 7.14: Performance of adapters in different network environments using the Royal Road benchmark with migration topology adaptation (top) and population size adaptation (pop).
experiments. This is due to the random drift and risk of premature convergence displayed by initial runs of the population adapter when the total population size tested is still too small to give a robust performance. Seeding gives the restarted DGA a bias toward promising regions of the search space, making it more likely to avoid low quality optima, and provide information that allows the population adapter improve its performance.

It was shown that migration topology adaptation does not conflict with the working of the population adapter. The combination of migration topology and population size adaptation was tested and shown to perform significantly better than population size adaptation alone.

The population adapter requires up to about three times the effort needed to solve the problem when the optimal population size is known beforehand. Some problems have characteristics that make current population sizing theory applicable, but for the more messy real world problems the practitioner is left with at least some manual trial and error testing. In this case, the population adapter overhead is not unreasonable, as the manual process would certainly require a fair number of trials as well. The population adapter has the additional benefit of relieving the user from the tedious testing process, and performing the tests in a predefined, systematic manner. Furthermore, it was shown that the difference in computation effort grows smaller as the network becomes more heterogeneous and dynamic, since the more difficult environment gives more room for the adaptation methods to influence the performance of the GA.
Chapter 8

G2DGA Application: VLSI Floorplanning

In this chapter an application for solving the multi-layer floorplanning problem is presented. The application demonstrates that G2DGA can be used to solve challenging real world problems, and it is extensively used in the scalability and robustness investigation described in Chapter 9.

8.1 Introduction

This chapter describes and evaluates a G2DGA application for solving the multi-layer floorplanning problem. The application was chosen because it is an example of the real world problems that GAs often are applied to. The implementation and execution of this application on G2DGA provides a challenging test problem to complement the benchmark testing results in the evaluation of the framework. The original contribution of the chapter is a representation which, to the best of my knowledge, is the first three-dimensional slicing structure floorplan representation described in the literature.

The material in this chapter has earlier been presented in part at the EuroGP Workshops, Coimbra, Portugal, 2004 [15].
The remainder of the chapter is organised as follows. First, background on VLSI design and the floorplanning problem is discussed in Sections 8.2-8.3. The problem is formally specified in Section 8.4. Section 8.5 describes the application’s slicing structure representation and the GA extensions. Two series of experiments have been conducted to test the proposed floorplanner. Section 8.6 presents experiments on area optimisation using a sequential GA. Section 8.7 presents a full-scale floorplanner that optimises both area and wirelength using a true multi-objective GA. Finally, the work is summarised in Section 8.8.

8.2 Background

The rapid increase in VLSI integrated circuit complexity has been made possible by the development of Electronic Design Automation (EDA) tools that can simplify the design process by presenting abstractions of hardware at several levels from system specification to packaging. One of these phases is physical design, where the logical layout of a circuit (a list of components and connections) is transformed to a physical layout. Floorplanning is the problem of placing a set of large sub-circuits (blocks) without overlap on a layout surface to meet a set of design goals and constraints. It is an essential step in the hierarchical physical design of deep sub-micron VLSI circuits, and is often solved by simulated annealing, force-directed heuristics or aggregate methods.

As the VLSI complexity increases, interconnect is increasingly dominating the overall design. Three-dimensional (3D) IC design has been recognised as a possible remedy [90]. The inter-layer connections, also known as vias, needed for 3D IC production carry some cost in terms of space and speed, but the beneficial effect of being able to place more transistors within a clock cycle of each other can justify this expense. As of 2005, there is already some commercial interest in 3D technology for special circuits, e.g. adding high-performance memory to a processing unit [9]. Companies that work with 3D IC production include Terrazon Semiconductor (Illinois, USA), and Tru-Si Technologies (California, USA).

New production methods call for new EDA tools, especially in critical early stages such as floorplanning. Instead of solving the floorplanning problem layer by layer and leaving interlayer connections for the later partitioning and routing stages, a floorplanner that treats
the multi-layer interconnect as an integral part of the floorplanning problem should be able to place tightly connected modules closer, thus making it easier to meet their timing goals in the routing phase. The semiconductor industry has expressed the need for such a floorplanner [90], and theoretical aspects of multi-layer floorplanning have been investigated [97].

8.3 Related Work

The floorplanning problem is a generalisation of the quadratic assignment problem, which is an NP-hard problem [48]. The reported approaches can be divided into three general cases: constructive, knowledge-based, and iterative [96]. The constructive approach starts from a seed module, and adds modules to the floorplan until all modules have been selected. The knowledge-based approach uses a database with expert knowledge to guide the floorplan development. The iterative approach starts from an initial floorplan which undergoes a series of transformations until the optimisation goal is reached. The iterative class includes force directed relaxation methods, simulated annealing, and evolutionary algorithms.

A number of representations have been proposed for the floorplan problem, including direct representation, slicing structures, and non-slicing representations [96]. A slicing structure divides the floorplan by vertical and horizontal segments into a finite number of non-overlapping rectangles and fits one block into each segment [88]. Slicing floorplans are easy to use for optimisation, but they are only a subset of all possible floorplans and in recent years much research has been directed to devising more exact representations of the topology using directed acyclic graphs such as B*-trees [29], TCG [74], and Twin-Binary Sequences [125], or by using direct representations with parameters for the orientation and location of each block. More recently, 3D extensions to these non-slicing floorplans have been suggested as well [34, 103].

The slicing structure representation has been extensively used for iterative algorithms since its simplified structure makes it possible to develop efficient representations and operators for manipulation. Although slicing structures cannot represent all possible floorplans, empirical evidence suggests that the slicing floorplan can be quite efficient in packing blocks tightly [126]. The reduction of the solution space can be advantageous as long as the re-
maining solutions are good enough, and it has been argued that this is the case when the floorplan contains soft modules that have flexible shapes [126]. The enveloping area can be estimated in polynomial time by the shape curve method, which can handle combinations of soft, hard, and pre-placed modules [127]. Flexibility in module sizes is common in the early stages of design, making slicing structures suitable for the floorplanning problem. This can be compared with non-slicing floorplanners which handle floorplans with hard modules efficiently, but rely on complicated and often computationally intensive procedures to handle soft modules and pre-placed blocks [73].

Research in using GA with slicing structure representations has been reported for sequential GA [112, 101, 43] and parallel GA [33]. These GAs encode the slicing structure as Polish (postfix) expression strings in the genome, and define crossover operators that produce children by identifying and swapping subtrees in the parents. Tests with non-trivial benchmark problems give solutions of good quality compared to other iterative methods. The main drawback is that they are slow, and more recently progress has been made in reducing the runtime by developing more efficient representation and recombination operators, such as crossover operators that actively search for fit subtrees in each parent [73]. Another study uses a normalised postorder genome representation that guarantees a one-to-one mapping between genotype and phenotype [117]. By using an algorithm that constructs normalised expressions from a permutation list of blocks, and applying well-known methods for enclosing area estimation of Polish expressions, the authors of the study are able to beat a simulated annealing algorithm both in quality and number of evaluations when optimising for minimal dead space.

One recent paper uses a simplified multi-level floorplan representation with GA optimisation called LIFT [98], where the floorplan is coded as a set of 2D slicing plans, one for each layer. The present work is significantly different, in that the floorplan is a true 3D representation which can represent building blocks not only in each layer, but also between different layers.
8.4 Problem Formulation

In this application three optimisation objectives are used: total area, balanced layer, and wirelength.

8.4.1 Area Optimisation

**Input:** A set of $n$ rectangular blocks $\{B_1, B_2, \ldots, B_n\}$ and a list of triplets $\{A_1, r_1, s_1\}$, $\{A_2, r_2, s_2\}$, $\ldots$, $\{A_n, r_n, s_n\}$, where $A_i$ is the area, and $r_i$ and $s_i$ are the lower and upper bounds on the aspect ratio of block $B_i$. The blocks can be either soft (flexible shape) or hard, with $r_i < s_i$ if the block is soft, and $r_i = s_i$ if the block is hard. The blocks are divided into two sets, $S_1$ and $S_2$, such that $S_1 \cup S_2 = \{B_1, B_2, \ldots, B_n\}$ and $S_1 \cap S_2 = \emptyset$, where $S_1$ represents the blocks with fixed orientation and $S_2$ represents the blocks with free orientation. Symbols $p$ and $q$ represent the upper and lower bounds on the shape of each layer, and $m$ represents the maximum number of layers allowed.

**Output:** A feasible floorplan solution consisting of enveloping rectangle $R_i$ for layer $i$, $1 \leq i \leq j$, where $j$ is the number of layers. Each $R_i$ is subdivided by horizontal and vertical cuts into $p_i$ non-overlapping rectangles. The rectangles are labelled $R_1, R_2, \ldots, R_j$. The floorplan is subjected to the following constraints:

1. $j \leq m$
2. $n = \sum_{i=1}^j p_i$
3. $p \leq \frac{H_i}{W_i} \leq q$, $1 \leq i \leq j$, where $H_i$ and $W_i$ are the height and width of the enveloping rectangle for layer $i$.
4. $A_i = w_i \times h_i$ for all $R_i \in S_1$, where $h_i$ and $w_i$ are the height and width of $B_i$.
5. $r_i \leq \frac{h_i}{w_i} \leq s_i$ or $\frac{1}{s_i} \leq \frac{h_i}{w_i} \leq \frac{1}{r_i}$ for all $R_i \in S_2$

**Total area objective:** The area estimation is normalised by dividing the total block area with the sum of each layer’s enveloping rectangle area. $f=1.0$ is a perfect floorplan with no

---

2Free rotation means that the blocks can be rotated 90°.
waste:

\[ f = \frac{\sum_{i=1}^{n} A_i}{\sum_{j=1}^{j} H_i \times W_i} \]  

(8.1)

**Balanced layer objective:** Estimate if the layers are balanced (of similar size and shape) by comparing each block’s area with the bounding rectangle of the total floorplan. Figure 8.1 illustrates how the bounding rectangle for the full floorplan is related to the enclosing rectangle for each layer. \( f = 1.0 \) if all layers are of identical size and shape:

\[ f = \frac{\sum_{i=1}^{n} A_i}{j \times \max(H_i) \times \max(W_i)} \]  

(8.2)

### 8.4.2 Wirelength Optimisation

**Input:** For a given floorplan, each module has coordinates \((x, y, z)\). The connections between modules in the floorplan are given by a set of \(n\) netlists, where each netlist represents a wire connecting two or more modules.

**Output:** The total length needed to connect the modules, using the semi-perimeter method [96], where the wirelength \(w\) is estimated as half of the perimeter of the smallest bounding rectangle enclosing all modules in a given netlist.

Let \(M_i\) be the set of modules for netlist \(i\), and \(x_j\) and \(y_j\) be the \(x\) and \(y\) coordinates for module \(j\) in \(M_i\). Then the wirelength is:

\[ w = \sum_{i=1}^{n} (\max_{j \in M_i} x_j - \min_{j \in M_i} x_j) + (\max_{j \in M_i} y_j - \min_{j \in M_i} y_j) \]

Interconnect can in a similar way be estimated as:

\[ \alpha = \sum_{i=1}^{n} k \cdot (\max_{j \in M_i} z_j - \min_{j \in M_i} z_j) \]

where \(k\) estimates the extra wiring needed by inter-layer via connections.
Wirelength objective: Minimise total wirelength objective \( w + \alpha \):

\[
f = \sum_{i=1}^{n} (\max_{j \in M_i} x_j - \min_{j \in M_i} x_j) + (\max_{j \in M_i} y_j - \min_{j \in M_i} y_j) + k \cdot (\max_{j \in M_i} z_j - \min_{j \in M_i} z_j)
\] (8.3)

8.5 Representation

In the 3D floorplanner, a floorplan is encoded in normalised Polish expressions that use the symbols ‘H’, ‘V’, and ‘Z’, representing horizontal, vertical, and lateral cuts, respectively. The integers 1 \ldots n represent the blocks in a problem. The Polish expressions can be mapped to a corresponding binary tree. An example can be found in Figure 8.2, which shows the binary tree for the Polish expression “3 1 6 8 Z H Z 2 7 Z V 5 4 H V.”

To evaluate the floorplan it is necessary to break down the 3D slicing structure. In this work a splicer algorithm has been developed which accepts a 3D floorplan and the maximum number of layers, and returns a slicing structure for each layer. Figure 8.2 illustrates how the splicer algorithm transforms the 3D floorplan tree (left side) into one 2D floorplan for each layer (right side).

The basic task of the splicer algorithm, outlined in Algorithm 7, is to examine the binary tree for the 3D slicing floorplan, and remove ‘Z’ nodes, leaving only a 2D slicing tree with ‘V’ and ‘H’ nodes. All ‘Z’ nodes in the current layer are replaced with their left child while their right child is inserted into the next layer. Since the maximum number of layers is fixed,
the function wraps around and starts reinserting nodes in the first layer when it reaches the top layer.

If the next layer is empty, the right child becomes the next layer; otherwise a new root node joins the current content of the next layer and the right subtree. I consider subtrees to be the natural building blocks for iterative methods to work with, and take special care to keep their topological relationships intact when splitting the full 3D slicing structure into separate 2D structures for each layer. The ID (‘V’ or ‘H’) of the new root node is defined as the common ancestor in the original 3D floorplan between the most previously inserted subtree and the subtree to be inserted. In this way the relative location between subtrees is maintained. Figure 8.3 shows the slicing structures corresponding to the examples used in Figure 8.2. Blocks 6 and 1, originating from the left-most ‘Z’ node in Figure 8.2, are put to the right of block 7 which originates from the right-most ‘Z’ node, since the common ancestor of both subtrees is the left-most ‘V’ node in the 3D tree from Figure 8.2.

Once all ‘Z’ nodes have been removed from the current layer the next layer becomes the current layer, and the algorithm iterates until no new layers are generated.

134
Algorithm 7 Pseudo-code for the splicer algorithm

Function Splicer
[Input: \( m \) = number of layers, 3DTree = Floorplan Tree]
[Output: array of 2D floorplans]
layerNumber = 0
currentLayer = 3DTree
do {
    nextLayer = 0
    subtrees = FindZSubtrees(currentLayer)
    foreach(tree in subtrees) {
        currentLayer.ReplaceNode(tree.root, tree.left);
        subtrees.Append(FindZSubtrees(tree.left))
        if(nextLayer == 0) {
            nextLayer = tree.right
        } else {
            node = CommonAncestor(tree.right, lastLayer);
            while (node = 'Z') node = node.parent;
            nextLayer = Node(node.ID, tree.right, nextLayer)
        }
    }
    lastLayer = nextLayer
    2DTree[layerNumber % m] = currentLayer
    currentLayer = nextLayer
    layerNumber = layerNumber + 1
} while(currentLayer != 0)
return 2DTree[]

Function FindZSubtrees
[Input: binary 3D Floorplan tree]
[Output: array of subtrees]
Find all subtrees with a 'Z' node root without 'Z' node ancestors.
8.5.1 GA Encoding

The GA used to evaluate the 3D floorplan representation is based on a recently published normalised Polish expression GA. Below I briefly describe the extensions to handle the new 3D representation, and the reader is referred to [117] for an in-depth description of the GA.

The genome consists of an array of records, each of which corresponds to a block in the problem. Each record has three fields:

1. A block ID: this identifies the block from the data set
2. A chain type ID: this can be one of three operator chains; 1=VHVHV\ldots , 2=HVHVHV\ldots , and 3=ZHZV\ldots
3. Chain field length: this defines the maximum length of the operator chain associated with this field.

The phenotype (a normalised Polish expression) is constructed using this algorithm:

1. Examine the first record; copy the rectangle ID to the phenotype.
2. Generate the chain of alternating operators specified by the chain type ID.

3. Copy operators from the chain to the phenotype until the end of the chain is reached or more operators would invalidate the expression.

4. Go to 1 if there are more records to process, or else complete the expression by continuing to copy alternating operators until the expression is valid.

The genome is initialised as in the original implementation for 2D floorplans, with the exception of the chain type which is set to 1, 2 or 3 with equal probability. The genome is order based, and use a permutation crossover operator to assign blocks to leaves in the slicing tree. In Valenzuela and Wang’s original work [117] four permutation crossovers where tested and cycle crossover (CX) [80] was found to work best on the 2D representation. Since the 2D and 3D representation are similar in that in both cases partial arrays correspond to blocks that are located close to each other\(^3\), I also decided to use CX crossover in my application. The experimental results are good, as can be seen in Section 8.6.2 and 8.7.3 below. However, since the 2D and 3D representations are not identical, it is possible that another crossover operator, either custom-made or one described in the literature, would perform even better. This could be the subject of future research.

There are three mutation operators that modify the genome:

1. Swap positions of two block IDs.

2. Change the chain type (type 1 becomes 2, 2 becomes 3, 3 becomes 1).

3. Mutate the length by increasing or decreasing the chain length with equal probability.

8.5.2 Genome Decoder

The objective function first decodes the genome representation to a 3D Polish expression, and uses the splicer algorithm to generate a 2D Polish expression for each layer. The enveloping rectangle for each of these expressions is calculated using the curve method. The

\(^3\)In the 3D representation close neighbour blocks can also be located on a level above or below the current block level.
objective function also checks that the width to height ratio of the enveloping rectangles lies in the allowed \(0.5 \leq \frac{h}{w} \leq 2\) range, and the GA will reapply the selection and recombination operators until a valid offspring has been produced.

### 8.5.3 G2DGA Implementation

The floorplanner has been implemented in G2DGA by writing a genome class, NPEGenome (Normalised Polish Expression Genome), that encapsulates the representation and the recombination operators. Figure 8.4 contains an UML class diagram that shows implementation details including the most important classes, attributes, and methods. The GA, splicer algorithm, and the objectives are implemented in the FloorplannerProblem class, which is a subclass of GAPProblem. When deployed on the G2DGA framework, each island instantiates the GA through the Init method in GAProblem.
8.6 Area Minimisation

8.6.1 Experimental Design

In these experiments the problem is limited to optimisation of the floorplan area only. The GA is a simple GA with elitism 1, population size 200, maximum number of generations 2000, 2-tournament selection, crossover probability 0.9, and mutation probability 0.1. The number of layers is set to 4.

To evaluate the new representation objective functions have been defined to test for total area minimisation, and for balanced layer minimisation where each layer will be of similar size and shape. Initially, several alternative fitness functions were tried, including minimising for area or waste, and optimising on all layers or for the largest layer only. Most of these variations tended to put a small number of blocks on the higher layers that fit together with almost no waste, and put the remaining blocks in the bottom layer. The best result was achieved when calculating the fitness as an aggregate multi-objective problem, combining equation 8.1 and 8.2:

\[
\text{fitness} = p \cdot \frac{\sum_{i=1}^{n} A_i}{\sum_{i=1}^{j} H_i \cdot W_i} + (1 - p) \cdot \frac{\sum_{i=1}^{n} A_i}{j \cdot \text{max}(H_i) \cdot \text{max}(W_i)}
\]

where \(p\) is a control parameter that balances between the two objectives.

8.6.1.1 Total Area Objective

For the total area minimisation, dividing the total block area with the sum of each layer’s enveloping rectangle area gives a reasonably even distribution of blocks among the available layers while efficiently reducing waste.
Table 8.1: Area minimisation results with MCNC benchmarks.

<table>
<thead>
<tr>
<th>Problem</th>
<th>p</th>
<th>Best 3D Bounding</th>
<th>Area</th>
<th>Waste</th>
<th>Best 2D Bounding</th>
<th>Area</th>
<th>Waste</th>
</tr>
</thead>
<tbody>
<tr>
<td>ami33</td>
<td>1.0</td>
<td>1190*665</td>
<td>1193346</td>
<td>3.09%</td>
<td>1337*903</td>
<td>1207311</td>
<td>4.21%</td>
</tr>
<tr>
<td>ami33</td>
<td>0.3</td>
<td>651*504</td>
<td>1223530</td>
<td>5.48%</td>
<td>7084*5250</td>
<td>37191000</td>
<td>4.69%</td>
</tr>
<tr>
<td>ami33</td>
<td>0.0</td>
<td>658*504</td>
<td>1315944</td>
<td>12.1%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ami49</td>
<td>1.0</td>
<td>2296*5572</td>
<td>13891500</td>
<td>3.76%</td>
<td>7084*5250</td>
<td>37191000</td>
<td>4.69%</td>
</tr>
<tr>
<td>ami49</td>
<td>0.3</td>
<td>3248*3136</td>
<td>38179820</td>
<td>7.14%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ami49</td>
<td>0.0</td>
<td>4354*2394</td>
<td>41073760</td>
<td>13.7%</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

8.6.1.2 Balanced Layer Objective

For balanced layer minimisation, the fitness value is the sum of each block’s area divided by the number of layers times the area of the bounding rectangle of the total floorplan. Figure 8.1 illustrates how the bounding rectangle for the full floorplan is related to the enclosing rectangle for each layer.

8.6.2 Experimental Results

To verify the potential of 3D floorplanning the performance of the 3D floorplanner was compared with the best reported results on MCNC benchmarks [79]. The GA can easily find the optimal solutions for the soft block version of the benchmarks, so the more difficult hard version has been used to be able to make meaningful comparisons. Table 8.1 shows the results when testing for total area minimisation only (p = 1.0), and demonstrates how the new floorplanner, by taking advantage of the increased flexibility in block placement offered by the use of multiple layers, can get better packing than the previously known optimal packing for 2D floorplanners.

The total area minimisation objective tends to put blocks that fit well together in the upper layers and puts the rest of the blocks in the bottom layer which makes the upper layers smaller. In industrial applications it is more important to generate layers of similar size and shape, and therefore the testing was extended to experiments with multiple objectives (p < 1.0). The balanced layer objective sacrifices some packing efficiency for an even distribution of layer size, and too much emphasis gives poor results (e.g. p = 0.0). Empirically p = 0.3
8.7 Multi-objective Optimisation

The aggregating multi-objective method which was used in the area optimisation experiments in Section 8.6 is simple to use and understand, but in practice it is hard to set the weights as the number of objectives grow. In this section the floorplanner will be extended to three objectives: total area, balanced layer, and wirelength optimisation. In order to handle the increased complexity, a true multi-objective optimisation GA called the Combative Accretion Model (CAM) has been used. In this section the idea behind CAM, and the extension made to make it work with the floorplanning representation, are briefly described before the experimental design and results are presented.

8.7.1 Combative Accretion Model

CAM, introduced by Berry and Vamplew [19], is a multi-objective algorithm based on concepts from Artificial Life research, which moves away from explicit population-wide Pareto ranking, abandons both complex archiving and diversity measures, and incorporates a continuous accretion-based approach that is different from the discretely generational nature of traditional evolutionary algorithms.

Central to CAM is the notion of agents and agent interaction. An agent is representative of a complete solution to the given multi-objective problem and carries an *size* parameter that is representative of performance in the population. Agent interaction is strictly pairwise and the results are dictated by the dominance relationship between the two individuals. The result of an interaction does not necessarily infer agent reproduction, as in conventional
GA selection, but rather dictates agent survival and changes in agent size.

To drive agents toward the current Pareto front, CAM makes use of an elitist concept based around the temporary storage of genes sourced from successful agents. This storage is updated by any agent that passes a pre-specified agent size threshold (and is thus considered suitably fit). To ensure a static population size, agent creation only ever occurs upon the death of another agent, and the system is essentially a poor-performer replacement scheme, whereby successful agents are retained simply by surviving. New agent generation is performed by harvesting the storage to create a new individual.

Extensions to the Combative Accretion Model

The original implementation of CAM creates new agents by picking genes at random from the storage. This method is similar to traditional uniform crossover, and would work badly with permutation-based genomes, such as the floorplanning representation. In this work the agent creation model was implemented as described in [19], except for storage which was modified to store complete elite solutions. New agents are created by randomly selecting two elites in the store and using the crossover and mutation operators described in Section 8.5.1 on them. The crossover creates two children and one of them is randomly selected to be the new agent.

8.7.2 Experimental Design

In these experiments the problem is optimised for three objectives: total area, balanced layer, and wirelength optimisation. These objectives were previously described in Section 8.4. The parameters used (unless otherwise noted) are elitism 100, population size 200, maximum number of generations 2000, 2-tournament selection, crossover probability 0.9, and mutation probability 0.1. The number of layers is set to 4.
Figure 8.5: The Pareto front with three objectives; wirelength, total area, and balanced layer optimisation.

8.7.3 Experimental Results

Figure 8.5 shows the final set of non-dominated solutions for a run of the multi-objective floorplanning problem on the ami33 benchmark, and demonstrates visually that the solutions are roughly evenly spread over a solution surface. The floorplanner was also tested on the harder ami49 benchmark, and its performance was compared with results of two 3D floorplanners from the literature, LIFT [98] and CBA [34]. CBA is a non-slicing floorplanner for hard block floorplanning, while LIFT only reports experiments on soft blocks. The results of the comparison are summarised in Table 8.2. The floorplanner is able to find solutions for both versions of the problems that are better in area, and similar or better in wirelength. It is noteworthy that the CBA result puts much emphasis on wirelength minimisation and has a total area waste of 43%. This floorplanner’s result has a slightly longer wirelength, but by accepting a wirelength reduction of 2% this waste is cut in half. Even better area result can be obtained if the wirelength objective is relaxed further, and the best result for area minimisation only is 35.87 (1.2% waste).
<table>
<thead>
<tr>
<th></th>
<th>this work</th>
<th>LIFT</th>
<th>CBA</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>area</td>
<td>waste</td>
<td>area</td>
</tr>
<tr>
<td>ami49, soft</td>
<td>35.8</td>
<td>544.9</td>
<td>37.4</td>
</tr>
<tr>
<td>ami49, hard</td>
<td>43.6</td>
<td>472.9</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 8.2: Area and wirelength comparison among 3D floorplanners (4 layers).

Figure 8.6: Best balanced floorplan found for ami49 (hard blocks), total waste 7.14 %.
8.8 Summary

In this chapter a 3D slicing floorplan representation for the multi-layer floorplan problem was presented and evaluated. The packing efficiency is in line with the theoretical predictions for 3D slicing floorplans [97]. The experiments conducted so far have optimised the floorplan three objectives: total area, balanced layer, and wirelength. For future extensions other objectives could be considered, such as minimising the number of vias, or adding a thermal model to add temperature constraints.

The floorplanning problem is NP-hard and provides a challenging application for Internet DGAs. The floorplanner has been implemented on the G2DGA framework, and demonstrates that G2DGA can be used to solve real world problems. The application is extensively assessed in the scalability and robustness investigation described in the next chapter.
Chapter 9

Investigation of Scalability and Robustness

This dissertation has proposed a DGA framework that uses a hybrid P2P architecture and adaptive methods to make efficient use of available resources. The benefit of this approach has so far been demonstrated on standard benchmark problems in small homogeneous local networks. In this chapter the testing is extended to experiments that have been carried out to measure the scalability and robustness of G2DGA on real world problems in typical usage scenarios.

9.1 Introduction

The G2DGA framework is evaluated by a series of experiments which are designed to test GA performance under various conditions. The purpose of testing is to:

1. Study the resource and problem size scalability.

   A series of experiments are conducted to evaluate both resource and problem size scalability. The system is tested on how it scales with the number of machines, and also how it scales with problem size.
Table 9.1: VLSI Floorplanner GA Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Problem</td>
<td>ami33 or ami49 [79], 4 layers</td>
</tr>
<tr>
<td>elitism</td>
<td>1 individual</td>
</tr>
<tr>
<td>population size</td>
<td>80</td>
</tr>
<tr>
<td>number of islands</td>
<td>32</td>
</tr>
<tr>
<td>GA</td>
<td>Generational</td>
</tr>
<tr>
<td>crossover</td>
<td>CX crossover [80], pc=0.9</td>
</tr>
<tr>
<td>mutation</td>
<td>swap/length/type [117], pm=0.01</td>
</tr>
<tr>
<td>termination</td>
<td>0.99% fitness, or no improvement in 20 generations</td>
</tr>
<tr>
<td>migration topology</td>
<td>uni-ring</td>
</tr>
<tr>
<td>migration interval</td>
<td>5 generations</td>
</tr>
<tr>
<td>migration rate</td>
<td>1 individual (best replaces worst)</td>
</tr>
</tbody>
</table>


The ability of G2DGA to handle volatile and heterogeneous environments is tested in usage scenarios with varying degrees of dynamic events. While available computer labs are used in the basic experiments, the simulator is needed for scenarios that define specific patterns of volunteer availability and communication overhead.

The remainder of the chapter is organised as follows. Section 9.2 describes the test problem, baseline DGA, and network environments used in the experiments. Section 9.3 presents experiments on resource and problem scalability, while Section 9.4 presents experiments on robustness. Finally, the results are discussed in Section 9.5.

9.2 Experimental Design

9.2.1 Test Problem

The application used to test G2DGA is the VLSI floorplanner, previously described in detail in Chapter 8. VLSI floorplanning, which is one of the most important problems in the physical design of VLSI circuits, is the problem of placing a given a set of rectangular modules of arbitrary sizes within a minimum rectangular area such that no module overlaps. The problem is NP-hard and often solved with evolutionary algorithms or simulated annealing, making it a good representative of a real world GA problem. In the experiments
the floorplanner has been used with two well-known test problems, \textit{ami33} and \textit{ami49}. All parameters (operators, mutation and crossover rates, etc.) are fixed, using the values found to be good during the floorplanner application development. These parameter values are listed in Table 9.1.

### 9.2.2 Speedup Measurements

Speedup is one of the most common ways to report the performance of a parallel algorithm. The standard definition is as follows: let $T(p, n)$ be the time to solve a problem of size $n$ on $p$ processing units. Often $n$ is omitted and the expression simplified to $T(p)$. $T(1)$ is the serial or sequential time, using the best serial algorithm. Then speedup $s(p) = T(1)/T(p)$ is a measure of how much faster the the application runs on $p$ processors than on one processor. A “speedup plot” is $s(p)$ plotted versus $p$.

For DGAs the standard speedup definition is hard to apply because the addition of migration means that the distributed GA behaves differently than a sequential GA. Furthermore, GAs are stochastic algorithms, and comparisons of single runs may be misleading. Alba and Tomassini [3] argue that the best speedup measurement of GAs is to compare the same DGA with different number of islands $1 \ldots n$, measuring the average time to reach a solution of given quality. The speedup results reported in this chapter all use this method.

### 9.2.3 Network Environments

G2DGA is intended to use idle cycles from computational resources on the so-called edge of the Internet: the computers connected to the Internet at the office or at home. For the experimental validation of the framework its performance on three specific types of computing environments have been tested: homogeneous clusters of workstations, heterogeneous Internet computing, and networks with dynamics. These are described in greater detail in the following subsections.

**HOM: Homogeneous cluster of workstations**
The cluster environment is the traditional environment for DGAs, and provides a baseline for comparison with heterogeneous and volatile environments. The homogeneous clustering experiments were conducted on workstations located at the Faculty of Information Technology at the Queensland University of Technology. Each workstation runs Windows XP and contains a 2 GHz Intel Pentium 4 CPU, 512 MB of RAM memory and an Intel Pro 100 Mbps Ethernet card.

**HET: Heterogeneous cycle-stealing**

The volunteer computers available for cycle-stealing applications such as G2DGA typically have non-identical processor resources and unequal cost of communication between the end points. The heterogeneous test scenario includes resources which differ in two fundamental ways: their computational capacity for processing work units and the capacity of the communication links connecting the computers.

Due to limited availability of computers and networks, and the difficulty in replicating interesting network events over a series of experiments, network simulation is used to configure this environment. The computational capacity of the computers is randomly distributed in the range of 1.0 - 3.0, meaning that the average computer runs at half of the speed of the fastest computer, and twice the speed of the slowest computer. The computers are connected to the Internet by broadband connections with a bandwidth 1.5 MBit/second. The ping latency between two end points has a Gaussian distribution between 1 to 40 ms. This performance profile is an abstraction of performance factors on real computers, such as processor, memory, operating system, communication capabilities, and background load. The values are based on Internet performance data collected with the Network Weather Service monitoring software on Internet connections in Denmark and Norway [22], as well as measurements on networks at QUT [106].

**VOL: Volatile heterogeneous cycle-stealing**

The dynamic environment is an extension of the heterogeneous cycle-stealing WAN
discussed above, where the availability of volunteer computers is changing over time. This is modelled by different phases where the number of computers is expanding or shrinking. Initially 20 computers are connected to the network. This is less than the 32 islands needed by the baseline configuration, and some islands will share computers. The first phase of the DGA run is an inflation phase, where islands will be added to the network at randomly chosen times. This is followed by a deflation phase, in which islands are randomly moved until only 10 remain. Finally, the number of islands expands again to a total of 200 and remains stable until the calculation is finished.

9.2.4 Statistical Methods

In this investigation the performance of different DGAs were compared. For ease of comparison of the experiments, their main statistics (number of evaluations, population size) are reported as mean and standard deviation over all runs. However, for the calculation of statistical significance, the performance of each DGA was sampled in a series of independent runs, and the samples compared with the non-parametric Wilcoxon Rank Sum test which provides a procedure for checking whether the probability distributions of two samples are identical [87]. The test does not rely on a normal distribution assumption, and only assumes that the probability distributions from which the samples are drawn are continuous and that the samples are independent and randomly selected. The experimental setup match this criteria.

In the test the combined sample data are combined and ranked (see Sheskin for a detailed description of the ranking procedure [102]). The test produces a test score given by

\[ Z = \frac{U - \frac{n_1 n_2}{2}}{\sqrt{\frac{n_1 n_2 (n_1 + n_2 + 1)}{12}}} \]

where \( U \) is the sum of ranks in the first sample, \( n_1 \) is the size of the first sample, and \( n_2 \) is the size of the second sample. In order to be significant, the obtained absolute value of \( Z \) must be greater than a tabled critical value at a given confidence level. The critical values are 2.58 and 1.96 for a confidence level of 0.99% and 0.95%, respectively [102].
In the experimental results the Z score is reported for each comparison together with the best GA and the confidence level used. The Z score has been calculated in Matlab 6.5, using the \texttt{ranksum(X,Y,ALPHA)} function, where \(X\) and \(Y\) are the samples, and \(ALPHA\) is the level of significance.

9.2.5 Baseline DGA

To provide a baseline of comparison the best static DGA was found by testing different combinations of the number of islands and the population size, while keeping other parameter settings constant. The experiments were conducted on workstations located at the Faculty of Information Technology at the Queensland University of Technology. Each workstation runs Windows XP and contains a 2 GHz Intel Pentium 4 CPU, 512 MB of RAM memory and an Intel Pro 100 Mbps Ethernet card.

Table 9.2 shows the outcome of these experiments for the \texttt{ami33} problem, with the results averaged over 10 runs. The difference in solution quality between different experiments has been tested for statistical significance using the Wilcoxon Rank Sum test, and the results are listed in Table 9.3.

It appears that a total population size of 2560 is required for solving the problem reliably. The 80,32 configuration (32 islands with 80 individuals on each island) is significantly better than 40,64 configuration. The 160,16 configuration has better average population fitness than 40,16, but the optimal fitness is not significantly better. There is no significant difference between 160,16 and 80,32. Increasing the population size further does not give improved quality, since there is no significant difference in performance between 160,16 (or 80,32) and either of the 80,64; 160,32; or 320,16 configurations. However, 80,32 has significantly better optimal performance than 320,16, suggesting that more islands are beneficial.

The configuration with 32 islands and 80 individuals on each island was chosen as the baseline configuration to compare performance of G2DGA with. The baseline configuration combines a low total number of evaluations with an average population quality that is as good or better than experiments using twice the effort, and has better optimal quality than the 160,16 configuration.
Table 9.2: Manual sizing of the ami33 problem on a LAN cluster. The reported values for evaluations, average population fitness, and optimal fitness are averaged over 10 runs.

<table>
<thead>
<tr>
<th>Population size</th>
<th>Evaluations</th>
<th>Pop fitness</th>
<th>Optima</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>total</td>
<td>island size</td>
<td>number of islands</td>
</tr>
<tr>
<td>1280</td>
<td>40</td>
<td>32</td>
<td>253375</td>
</tr>
<tr>
<td>80</td>
<td>16</td>
<td>268063</td>
<td>46611</td>
</tr>
<tr>
<td>160</td>
<td>8</td>
<td>252629</td>
<td>52174</td>
</tr>
<tr>
<td>2560</td>
<td>40</td>
<td>64</td>
<td>483095</td>
</tr>
<tr>
<td>80</td>
<td>32</td>
<td>653779</td>
<td>74851</td>
</tr>
<tr>
<td>160</td>
<td>16</td>
<td>611743</td>
<td>99365</td>
</tr>
<tr>
<td>5120</td>
<td>80</td>
<td>64</td>
<td>1335348</td>
</tr>
<tr>
<td>160</td>
<td>32</td>
<td>972750</td>
<td>91571</td>
</tr>
<tr>
<td>320</td>
<td>16</td>
<td>1294597</td>
<td>233434</td>
</tr>
</tbody>
</table>

Table 9.3: Test of statistical significance in manual population sizing of ami33 floorplanner application. The Winner column contains the best DGA or '-' if the difference is not statistically significant. The Z column is the result of the test. A Z score > 1.96 is statistically significant with 95% confidence level.

<table>
<thead>
<tr>
<th>Comparison</th>
<th>Pop. fitness</th>
<th>Optimal fitness</th>
</tr>
</thead>
<tbody>
<tr>
<td>40,64</td>
<td>80,32</td>
<td>80,32</td>
</tr>
<tr>
<td>40,64</td>
<td>160,16</td>
<td>160,16</td>
</tr>
<tr>
<td>80,32</td>
<td>160,16</td>
<td>-</td>
</tr>
<tr>
<td>80,32</td>
<td>80,64</td>
<td>-</td>
</tr>
<tr>
<td>80,32</td>
<td>160,32</td>
<td>-</td>
</tr>
<tr>
<td>80,32</td>
<td>320,16</td>
<td>-</td>
</tr>
<tr>
<td>160,16</td>
<td>80,64</td>
<td>-</td>
</tr>
<tr>
<td>160,16</td>
<td>160,32</td>
<td>-</td>
</tr>
<tr>
<td>160,16</td>
<td>320,16</td>
<td>-</td>
</tr>
</tbody>
</table>

Winner | Z   | Winner | Z   |
--------|-----|--------|-----|
80,32   | 3.7418 | 80,32 | 3.5907 |
160,16  | 2.0032 | -     | 1.3985 |
-       | 0.9449 | -     | 1.4741 |
-       | 0.7181 | -     | 0.7181 |
-       | 0.3402 | -     | 1.7008 |
-       | 1.5342 | 80,32 | 2.2300 |
-       | 1.0961 | -     | 1.7764 |
-       | 0.7181 | -     | 0.0378 |
-       | 0.4914 | -     | 0.2646 |
Table 9.4: Run-time and performance comparison of ami33 floorplanner application using uni-ring, fully connected, and adaptive migration topologies.

<table>
<thead>
<tr>
<th>Type</th>
<th>Time</th>
<th>Population fitness</th>
<th>Optimal fitness</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>mean</td>
<td>stddev</td>
<td>mean</td>
</tr>
<tr>
<td>fully connected</td>
<td>53</td>
<td>13</td>
<td>0.9</td>
</tr>
<tr>
<td>uni-ring</td>
<td>120</td>
<td>28</td>
<td>0.89</td>
</tr>
<tr>
<td>adaptive</td>
<td>120</td>
<td>69</td>
<td>0.91</td>
</tr>
</tbody>
</table>

Table 9.5: Test of statistical significance in performance comparison of ami33 floorplanner application using uni-ring, fully connected, and adaptive migration topologies. The Winner column contains the best DGA or ‘-’ if the difference is not statistically significant. The Z column is the result of the test. A Z score > 1.96 is statistically significant with 95% confidence level.

<table>
<thead>
<tr>
<th>Comparison</th>
<th>Pop. fitness</th>
<th>Optimal fitness</th>
</tr>
</thead>
<tbody>
<tr>
<td>Winner</td>
<td>Z</td>
<td>Winner</td>
</tr>
<tr>
<td>Adaptive vs. fully connected</td>
<td>-</td>
<td>0.9449</td>
</tr>
<tr>
<td>Adaptive vs. uni-ring</td>
<td>Adaptive</td>
<td>2.0032</td>
</tr>
<tr>
<td>fully connected vs. uni-ring</td>
<td>-</td>
<td>1.0061</td>
</tr>
</tbody>
</table>

An initial series of experiments were conducted to further investigate baseline configuration and compare it with both adaptive and static migration topologies. Table 9.4 shows run-time and performance averaged over 10 independent runs each of adaptive (migration topology adaptation) and static (uni-ring and fully connected) versions of the floorplanner. The adaptive and uni-ring floorplanners have much lower connectivity levels than the fully connected topology, which converges faster and leads to lower run-times. The performance differences between fully connected and uni-ring topologies are not statistically significant (see Table 9.5), while migration topology adaptation performs significantly better than uni-ring topologies. This behaviour is consistent with the benchmark testing conducted on the migration topology adapter in Chapter 6.

9.3 Scalability Study

9.3.1 Resource Scalability

Two series of experiments were conducted to measure how well G2DGA scales with available computational resources: one static using a fixed migration policy, and one adaptive where
the migration topology is dynamically changed using the migration topology adapter method previously described in Chapter 6. The number of volunteer computers is the only variable in the experiments, with all other parameters fixed. The experiments were carried out on the homogeneous LAN network. All calculation times are averaged over 10 runs.

The upper graph in Figure 9.1 shows mean fitness for ami33 experiments with 1, 5, 10, and 20 computers plotted against time. The graph shows that the calculation time depends on the number of volunteers. The lower graph plots the same experiment against effort, and shows that the solution quality is close to identical for all experiments.

The speedup has been calculated and plotted in Figure 9.2. As the number of computers increases, there are fewer island nodes sharing computers and the computation time decreases. However, the communication becomes slower as more and more of the migration paths go through the LAN instead of using faster object-to-object communication on the same computer. This slows down migration and increases the effort needed to reach the desired quality, limiting the speedup when the number of volunteers is large. Another important observation is that G2DGA with adaptation has much better speedup than the baseline DGA.

9.3.2 Problem Scalability

Problem scalability measures how an algorithm scales as the size of the problem grows. To investigate the ability of G2DGA to scale with problem size, a series of experiments were conducted with the floorplanner on the more difficult ami49 benchmark. Except for the benchmark problem tested, the experimental set-up is identical to the ami33 experiments described in Section 9.3.1 above. The scalability of the system is evaluated by comparing the speedup and performance for the two problem sizes.

Figure 9.3 plots its mean fitness for ami49 experiments with 1, 5, 10, and 20 computers plotted against time. As with ami33, the performance of ami49 is similar for all experiments, while the speedup directly depends on the available computational resources.

Figure 9.4 shows the speedup plot for ami49. The graph has similar features to the
Figure 9.1: 20 island ami33 floorplanner application on 1, 5, 10, and 20 computer LAN. The upper graph plots performance against time, while the lower plots performance against effort.
ami33 speedup graph (see Figure 9.2), with good initial speedup which is eventually limited by increasing communication cost between the increased number of computers. The speedup, especially when using adaptation, is better than ami33. This is due to the increased difficulty of the problem, which leads to longer evaluation times. Since the migration interval is defined as the number of generations of the GA, it follows that there will be longer intervals between migration events, reducing the impact of communication overhead.

For both problem sizes, G2DGA consistently performed as well, and usually significantly better than, the baseline DGA. The speedup is consistent over the different problem instances, and the results suggest that G2DGA is able to scale to the increased problem difficulty in the ami49 benchmark.

### 9.4 Robustness Study

A series of experiments were conducted to study G2DGA behaviour in usage scenarios with varying degrees of dynamic events and heterogeneity. These experiments compare the
Figure 9.3: 20 island ami49 floorplanner application on 1, 5, 10, and 20 computer LAN. The upper graph plots performance against time, while the lower plots performance against effort.
Figure 9.4: Speedup for a 64 island ami49 floorplanner application with varying number of computers.

performance of the autonomous framework, employing both population sizing and migration topology adaptation, with the baseline DGA. Table 9.6 presents the evaluations, mean population fitness, and mean optimal results, for ten independent runs on each network environment (HOM, HET and VOL). The differences in performance between different experiments have been tested for statistical significance using the Wilcoxon Rank Sum test, and the results are listed in Table 9.7.

In terms of heterogeneity and dynamics, HOM is the most stable followed by HET and VOL. The most important observation is that while the performance of the static DGA degrades as the network environment becomes more unstable, the autonomous performance remains stable. For population fitness there is no statistically significant difference for HOM, but the performance of the autonomic framework is significantly better for HET and VOL. The results are the same for optimal fitness, with no significant difference for HOM, and significantly better results for the autonomic G2DGA in HET and VOL.

Figure 9.5 shows the fitness plotted against the number of evaluations for the baseline DGA and G2DGA. The G2DGA trends come from experiments with migration topology
adaptation enabled, using the number of islands and population size found by the population size adapter. For the HOM network, the use of adaptation clearly reduces the effort needed to find the solution compared to the baseline DGA. When the networks become increasingly volatile, the baseline DGA fitness decreases, while G2DGA avoids premature convergence and maintains its fitness by using more effort. The results demonstrate that G2DGA is able to detect and respond to changes in the execution environment.

The population sizes found by the autonomous framework are smaller than the ones found to be needed for reliable performance on manual experiments for HOM and HET. This is due to the nature of the population adapter, where three DGAs are run in parallel and restarted if they are stuck or found to perform badly, making it easier to avoid premature convergence. For VOL, the number of volunteer computers vary and the communication overhead becomes more unpredictable since islands may be moved at any time. In this environment, the autonomous framework requires larger population sizes. The increased population size enables G2DGA to keep the solution quality on the same level as on the HOM and HET networks, and demonstrates that the autonomous framework can adapt correctly to dynamic events.

9.5 Summary

This chapter has presented experimental results for trial runs with different combinations of real world applications and test environments. The following observations are intended to summarise the most important findings of these experiments.

The experimental result on benchmark functions in Chapter 6 and Chapter 7 indicated that the adaptive methods help G2DGA to achieve good solution quality with low connectivity. In this chapter the experiments were expanded, using a real world problem in VLSI floorplanning as an example of a hard and less well-defined problem. The speedup was measured and shows that G2DGA scales better than the baseline DGA. This is true both for the ami33, and the more difficult ami49 benchmark. Furthermore, it was shown that G2DGA can scale to the proper population size needed to achieve good quality in a given network environment.
<table>
<thead>
<tr>
<th>Test</th>
<th>Network</th>
<th>Population size</th>
<th>Evaluations</th>
<th>Pop. fitness</th>
<th>Optima fitness</th>
</tr>
</thead>
<tbody>
<tr>
<td>Static</td>
<td>HOM</td>
<td>80,32 (2560)</td>
<td>653779 ± 74851</td>
<td>0.94 ± 0.011</td>
<td>0.98 ± 0.0066</td>
</tr>
<tr>
<td></td>
<td>HET</td>
<td>80,32 (2560)</td>
<td>484544 ± 66711</td>
<td>0.89 ± 0.018</td>
<td>0.93 ± 0.025</td>
</tr>
<tr>
<td></td>
<td>VOL</td>
<td>80,32 (2560)</td>
<td>499518 ± 38462</td>
<td>0.88 ± 0.02</td>
<td>0.93 ± 0.016</td>
</tr>
<tr>
<td>Auto</td>
<td>HOM</td>
<td>40,32 (1280)</td>
<td>1310652 ± 380759</td>
<td>0.94 ± 0.018</td>
<td>0.97 ± 0.012</td>
</tr>
<tr>
<td></td>
<td>HET</td>
<td>40,32 (1280)</td>
<td>987157 ± 276994</td>
<td>0.92 ± 0.018</td>
<td>0.96 ± 0.015</td>
</tr>
<tr>
<td></td>
<td>VOL</td>
<td>80,32 (2560)</td>
<td>1699570 ± 848199</td>
<td>0.93 ± 0.02</td>
<td>0.97 ± 0.014</td>
</tr>
</tbody>
</table>

Table 9.6: Comparison of autonomous G2DGA and static DGAs on HOM, HET, and VOL network scenarios. The reported values are averaged over 10 runs. Evaluations is the total number of evaluations calculated in every island, including unsuccessful trials initiated by G2DGA’s population size adapter.

<table>
<thead>
<tr>
<th>Comparison</th>
<th>Pop. fitness</th>
<th>Optimal fitness</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Winner</td>
<td>Z</td>
</tr>
<tr>
<td>HOM-Auto vs. HOM-Static</td>
<td>-</td>
<td>0.0378</td>
</tr>
<tr>
<td>HET-Auto vs. HET-Static</td>
<td>HET-Auto</td>
<td>2.9103</td>
</tr>
<tr>
<td>VOL-Auto vs. VOL-Static</td>
<td>VOL-Auto</td>
<td>3.5907</td>
</tr>
<tr>
<td>HOM-Static vs. HET-Static</td>
<td>HOM-Static</td>
<td>3.7418</td>
</tr>
<tr>
<td>HOM-Static vs. VOL-Static</td>
<td>HOM-Static</td>
<td>3.7418</td>
</tr>
<tr>
<td>HET-Static vs. VOL-Static</td>
<td>-</td>
<td>1.0205</td>
</tr>
<tr>
<td>HOM-Auto vs. HET-Auto</td>
<td>-</td>
<td>1.6252</td>
</tr>
<tr>
<td>HOM-Auto vs. VOL-Auto</td>
<td>-</td>
<td>1.0961</td>
</tr>
<tr>
<td>HET-Auto vs. VOL-Auto</td>
<td>-</td>
<td>0.6425</td>
</tr>
</tbody>
</table>

Table 9.7: Performance comparison of autonomous G2DGA and static DGAs using Wilcoxon rank sum test. The Winner column contains the best DGA or '-' if the difference is not statistically significant. The Z column is the result of the test. A Z score > 1.96 is statistically significant with 95% confidence level.
Figure 9.5: Performance degradation in HOM, HET, and VOL networks. The upper graph plots the mean fitness for the 80.32 baseline DGA, while the lower graph plots mean fitness for the autonomous G2DGA framework.
In the robustness testing, G2DGA found solutions of better fitness than the baseline DGA with similar connectivity. Furthermore, the gap between G2DGA and the static DGA performance grows wider as the network environment becomes more volatile and heterogeneous. This shows that the autonomous G2DGA, by continuously adjusting the migration topology and the population size, is able to respond to dynamic Internet behaviour. While the performance of a static DGA deteriorates with increased network instability, the degradation of G2DGA performance is not statistically significant.

In each scalability and robustness experiment, G2DGA consistently performed as well, and usually significantly better than, the baseline DGA. The experimental results suggest that adaptation is an efficient way of responding to the dynamic and volatile nature of Internet computing. They also demonstrate that G2DGA’s supervisor/island design allows the implementation of adaptive methods in a P2P network environment.
Chapter 10

Conclusion

This chapter summarises the findings and contributions that originate from the research contained in this dissertation. It also presents possible future research directions.

10.1 Summary

In this thesis I have presented G2DGA, a new cycle-stealing framework for Internet-based parallel and distributed GAs, which has the potential to provide researchers with enough computational power to apply GAs to interesting real world problems at low cost.

The first part of this thesis investigated previous art and demonstrated that many available DGAs are unfit to run as cycle-stealing applications on the Internet. The high communication costs have implications for scalability, and the heterogeneous and volatile network environment makes it hard to design a DGA that works well in unpredictable conditions. Starting from this analysis, a user-transparent framework was proposed. The framework combines a hybrid P2P architecture with adaptive methods to improve performance. The hybrid architecture allows the DGA to monitor the global search space without relying on a fixed server, while using the direct P2P communication for island to island migration.

The performance of a DGA depends on many GA and migration policy parameters that
interact in non-linear ways, making the choice difficult even in static cases. The volatile cycle-stealing environment complicates the situation further, and in order to maintain good performance of the DGA, the parameters may have to be adjusted when the environment changes. In order to facilitate such continuous adjustments, one of the focuses of this investigation has been on adaptive methods. I have focused on migration topology and population size adaptation, since the quality of the search is mainly determined by the spread of solutions between islands, and the population size on each island. The migration topology adaptation method is able to use this information to reduce communication overhead by detecting and connecting only the islands that benefit from migration. Population sizing adaptation automatically determines the number of islands and island population sizes needed for a given problem and network configuration.

The scalability and robustness of G2DGA has been tested through experimentation on a set of typical networks (LAN, WAN, and cycle-stealing scenarios). LAN was tested in computer laboratories with up to 60 computers, while WAN and cycle-stealing scenarios were simulated in an Internet emulator. This testing effort has included speedup, scalability and robustness evaluation, using both benchmarks and a multi-layer VLSI floorplan application, which was used as an example of the challenging real world problems to which GAs are often applied. Experiments with G2DGA on the VLSI floorplanner suggested that the framework scales better and works more robustly than static DGA implementation of the same application as the network becomes more volatile and heterogeneous.

10.2 Major Contributions

A number of contributions have been accomplished during the course of research. The major contributions are enumerated and discussed, as follows:

1. A convergence model for asynchronous parallel GAs

The model is an extension of PGA convergence modelling to heterogeneous distributed island models. In contrast with previous work, the model can handle scenarios with computers of varying performance, and with customisable fault rates. Another
contribution is an extensive investigation of migration policy parameters in dynamic Internet networks, using the convergence model.

2. A framework for Internet-based DGAs

The G2DGA framework uses a novel hybrid P2P architecture with island node activity coordinated by supervisor nodes that offer global overview and opportunities for adaptation. A unique feature of the framework is that it can run either on a P2P network or in a simulator mode without requiring any recompilations or modifications.

3. Migration topology adaptation

The migration topology adaptive method uses a novel clustering approach to dynamically update the migration paths between islands. The adapter reduces connectivity while maintaining high solution quality. The experiments in Chapter 9 have demonstrated that the migration topology adapter gives G2DGA better speedup and more robust performance than static DGAs.

4. Population size adaptation

The population size adapter automatically searches for a good combination of the number of islands, and the population size on each island, in DGAs. This adapter is applicable to a wide range of GA algorithms, and makes DGAs easier to use by reducing the number of user-set parameters and aiding in construction of robust GA applications with good performance. The experiments have shown that the population size adapter reliably finds the number of islands, and the population size on each island, needed to solve a given problem. The population sizes found by the algorithm are comparable to those found with traditional trial-and-error approaches.

5. A novel multi-layer VLSI floorplainer application

The floorplanner is an example of the kind of challenging real world problems that a typical GA practitioner works with. In this NP-hard problem a novel slicing structure representation was used, and the application was implemented and tested in the G2DGA framework. In the dissertation, the application was also used to supplement experimental testing of benchmark functions.

6. Scalability and robustness studies in Internet computing environments
The experimental study of the framework is more extensive than earlier research on similar approaches to Internet-based DGAs, e.g. the DREAM project. The study is a combination of computer lab experiments involving up to 60 computers, and Internet simulation runs. In the experiments several network scenarios were tested, ranging from stable LAN to volatile and heterogeneous cycle-stealing environments. The experimental results demonstrate that G2DGA’s supervisor/island design allows the implementation of adaptive methods in a P2P network environment, and show that adaptation is an efficient way of responding to the dynamic and volatile nature of Internet computing.

10.3 Extensions

While there is still much work needed to create Internet-based DGAs that are adaptive and efficient over a wide range of network conditions while still being simple enough to use without requiring extensive experimentation or expert knowledge, this research provides a scalable and robust P2P DGA framework which I believe can serve as a foundation for future work in the field. However, as often happens in research, the process of answering one set of questions leads to new ideas that still remain to be answered. This section briefly discusses some of these ideas that may form the basis for future research efforts either by extending work that has been presented in this dissertation, or by branching out to possible new lines of inquiry.

One goal of G2DGA is to make it easy for non-experts to develop and deploy GA applications on a cycle-stealing network. Currently, the dotGALib support library, the G2DGA architecture, and the adaptive methods support and reduce the complexity of G2DGA development, but still require some knowledge on the design and implementation of the framework to use. A possible future extension of G2DGA is to add a setup tool which provides a user interface which would determine the needs of the user by collecting basic information on the problem. The tool would hide the implementation details for the non-expert user by automatically generating code and compiling the application.

This research has used the island model since it allows the population to scale. However,
for many problems much of the computation overhead is attributed to evaluation of fitness functions. Since these evaluations are independent of one another, they could be distributed onto other nodes for calculation. An obvious extension to the current framework is therefore to use evaluation nodes in addition to the island and supervisor nodes.

An interesting observation made in Section 7.3.3 is that the population size adapter uses smaller population sizes than those required for reliable performance in manual experiments. This may be a consequence of the nature of the population adapter, where three DGAs are run in parallel and restarted if they are stuck or found to perform badly. Due to time limitations, this phenomena was never investigated in detail but it would be an interesting topic for future research to try to find an theoretical estimate on the expected population size needed, and the total effort used, compared to a non-adaptive DGA using the same problem.

The use of the hybrid P2P architecture allowed the collection and analysis of the global search state in the supervisor nodes. In the current research this information was used to adapt the migration topology, which was identified as being one of the most important of the parameters that influence DGA performance. There are also opportunities to expand the research in adaptive parameter adjustment further, using the search state or other island statistics as input. Promising approaches for such research would be to adaptively adjust the island mutation or migration rates. Another interesting extension to the current research would be to investigate the influence of different migration topology adaptation strategies during different stages of the search.
Appendix A

G2DGA Implementation Notes

A.1 Software Used

The following software has been used in the G2DGA development:

- Microsoft Visual Studio .NET 2003 with C#;

- Microsoft .NET framework version 1.1;

- “DockingSuite” from “Divelements” software (http://www.divil.co.uk/net/) was used to implement docking windows in the GUI for the Analyser and Simulator packages. The software is available at no charge to use in any products royalty-free, as long as copyright is acknowledged;

- “FolderTreeView” by Furty (furty74@yahoo.com) was used to implement the directory view in the Analyser package. The software is free for any use, so long as copyright is acknowledged;

- GnuPlot version 4.0 (http://www.gnuplot.info) is used by the Analyser and Simulator to create graphs. GnuPlot is released under a license that allows royalty-free usage in other products.
A.2 File Structure

The project file structure is as follows:

**G2DGA** contains the subprojects needed to implement G2DGA

- **bin** contains executables and libraries for G2P2P and all G2DGA subprojects
- **doc** contains documentation
- **Analyser** code and make files for the Analyser software
- **Simulator** code and make files for the Simulator software
- **G2DGA** code and make files for the framework software
- **G2DGAWizard** code and make files for the set-up tool software
- **G2DGAConsole** code for the graphical front-end for the simulator or G2P2P client
- **Simulator** code and make files for the Internet emulator and G2DGA simulator
- **dotGALib** code and make files for the dotGALib library

The G2DGA folder contains a Microsoft Visual Studio solution file called “g2dga.sln”. By double-clicking this file and selecting **Build/Build Solution** in Visual Studio, all projects are compiled. The executables (G2DGAConsole and Analyser) can be started by double-clicking them in the G2DGA/bin directory.

A.3 Documentation

Doxygen (http://www.doxygen.org) was used to generate on-line and off-line documentation of the G2DGA software, by extracting data structures and comments directly from the source files. The on-line documentation, in the form of cross-referenced html files that can be viewed in an Internet browser, is located in the G2DGA/doc/html directory. Doxygen also creates an off-line reference manual, which can be found in the G2DGA/doc/latex directory. To view and print the book, the Latex source has to be compiled using a Latex compiler and the included Makefile.
Bibliography


176


178


