SUSTAINABLE EVOLUTIONARY ALGORITHMS AND SCALABLE EVOLUTIONARY SYNTHESIS OF DYNAMIC SYSTEMS

By

Jianjun Hu

A DISSERTATION

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

Department of Computer Science and Engineering

2004

ABSTRACT

SUSTAINABLE EVOLUTIONARY ALGORITHMS AND SCALABLE EVOLUTIONARY SYNTHESIS OF DYNAMIC SYSTEMS

By

Jianjun Hu

This dissertation concerns the principles and techniques for scalable evolutionary computation to achieve better solutions for larger problems with more computational resources. It suggests that many of the limitations of existent evolutionary algorithms, such as premature convergence, stagnation, loss of diversity, lack of reliability and efficiency, are derived from the fundamental convergent evolution model, the oversimplified "survival of the fittest" Darwinian evolution model. Within this model, the higher the fitness the population achieves, the more the search capability is lost. This is also the case for many other conventional search techniques.

The main result of this dissertation is the introduction of a novel sustainable evolution model, the Hierarchical Fair Competition (HFC) model, and corresponding five sustainable evolutionary algorithms (EA) for evolutionary search. By maintaining individuals in hierarchically organized fitness levels and keeping evolution going at all fitness levels, HFC transforms the conventional convergent evolutionary computation model into a sustainable search framework by ensuring a continuous supply and incorporation of low-level building blocks and by culturing and maintaining building blocks of intermediate levels with its assembly-line structure. By reducing the selection pressure within each fitness level while maintaining the global selection pressure to help ensure exploitation of good building blocks found, HFC provides a good solution to the explore vs. exploitation dilemma, which implies its wide applications in other search, optimization, and machine learning problems and algorithms.

The second theme of this dissertation is an examination of the fundamental principles and related techniques for achieving scalable evolutionary synthesis. It first presents a survey of related research on principles for handling complexity in artificially designed and naturally evolved systems, including modularity, reuse, development, and context evolution. Limitations of current genetic programming based evolutionary synthesis paradigm are discussed and future research directions are outlined. Within this context, this dissertation investigates two critical issues in topologically open-ended evolutionary synthesis, using bond-graph-based dynamic system synthesis as benchmark problems. For the issue of balanced topology and parameter search in evolutionary synthesis, an effective technique named Structure Fitness Sharing (SFS) is proposed to maintain topology search capability. For the representation issue in evolutionary synthesis, or more specifically the function set design problem of genetic programming, two modular set approaches are proposed to investigate the relationship between representation, evolvability, and scalability.

To my wife Fangfang, mom, dad, and sister

for their support and their pride of each progress
I made during this dissertation research

ACKNOWLEDGEMENTS

It is a risky enterprise to propose that a new sustainable evolution model is needed to achieve scalable evolutionary computation by replacing the "survival of the fittest" model held as a tenet for more than three decades. At the end of this expedition, I am indebted to many people for their inspirations, ideas, encouragement, support, and love that have endowed me with energy, courage, and enthusiasms.

Foremost, I'd like to thank my parents who have been always proud of their son for each progress he has made in study and research. I am also grateful to my parents-in-law for bringing up their lovely daughter who stays with me while they are left alone themselves.

My greatest gratitude goes to my advisor Erik Goodman, who has always been my ultimate source of inspiration, encouragement, and support during this hard journey. It is his challenges to my ideas, his vision of the fundamentals of evolutionary computation, and his insights in our numerous discussions that help to shape this dissertation. I also owe my thanks to my previous supervisor in China, Professor Shuchun Wang for guiding me into this exciting area of computational intelligence.

I would also like to thank my dissertation committee, Bill Punch, Charles Ofria, Ronald Rosenberg, and Charles MacCluer for sparing their precious time to serve on my committee and giving valuable comments and suggestions. Their help is not limited to the dissertation itself, but also includes many other aspects, such as advice concerning my job talk presentation style.

I am very happy to have worked with an exciting research group, the GPBG group at the MSU genetic algorithm research lab (GARAGe). I owe my special thanks to Dr. Kisung Seo for his help all through this three-year NSF project, to Dr. Ronald Rosenberg for his exemplary high standard of research, and to Zhun Fan for many interesting discussions.

Finally, I am especially indebted to my wife, Fang Han for her deep love, understanding, and no-effort-sparing support for my research, without which this dissertation is impossible. I am proud and fortunate enough to have such an encouraging and wonderful woman to share a happy life available only after her presence.

TABLE OF CONTENTS

LIST OF TABLES ..... **xi**

LIST OF FIGURES ..... **xiii**

1 INTRODUCTION **1**

1.1 Motivation and Background 2
1.2 Problem Statement 5
1.3 Proposed Evolution Models and Techniques 8
1.4 Contributions and Significance 10
1.5 Organization of the Dissertation 12

2 BACKGROUND AND RELATED WORK **14**

2.1 Overview of Evolutionary Algorithms 14
2.1.1 The Darwinian Root of Evolutionary Algorithms 15
2.1.2 Genetic Algorithms 16
2.1.3 Genetic programming 21
2.1.4 Evolution Strategies and Evolutionary Programming 24
2.1.5 Common Framework, Issues and Pitfalls 27
2.2 Previous Work on Sustainable Evolutionary Computation 28
2.2.1 Previous Approaches to Balance Exploration and Exploitation 29
2.2.2 Previous Approaches to Sustainable Evolution by Improving Evolvability ..38
2.2.3 A Common Root of Tragedies: the Enigma of Premature Stagnation 41
2.3 Overview of Topologically Open-Ended Evolutionary Synthesis 44
2.3.1 Topologically Open-Ended Synthesis: a Common Framework 44
2.3.2 Direct Encoding Techniques 47
2.3.3 Indirect Encoding/Generative Encoding Techniques 50
2.3.4 Artificial Embryology 52
2.4 Previous Work To Improve Scalability of Evolutionary Synthesis 54
2.5 Summary 56

3 TOPOLOGICALLY OPEN-ENDED SYNTHESIS OF DYNAMIC SYSTEMS BY GENETIC PROGRAMMING **58**

3.1 Brief History of Dynamic System Design Synthesis 58
3.2 The Representation Issue in Automated Dynamic System Synthesis 60
3.3 Bond Graph Representation of Dynamic Systems 66
3.4 Automated Synthesis of Bond Graphs by Genetic Programming 69
3.4.1 Program Architecture 69
3.4.2 Types of Building Blocks and Modifiable Sites 70
3.4.3 Embryos 71
3.4.4 Terminal Set and Function Set 72
3.4.5 Fitness Evaluation and Simulation Platform Selection 76

3.5 Benchmark Problems 76
3.6 Summary 80

4 HIERARCHICAL FAIR COMPETITION (HFC): A SUSTAINABLE EVOLUTIONARY COMPUTATION MODEL 81
4.1 The Convergent Nature of the Prevailing Evolutionary Computation Model..... 81
4.1.1 Two Observations and Two Common Implicit Assumptions in Typical EA Frameworks 81
4.1.2 Premature Convergence and Stagnation: an Explanation 88
4.1.3 Requirements for Sustainable Evolutionary Algorithms 90
4.2 The Hierarchical Fair Competition Model for Sustainable Evolution 95
4.2.1 The Metaphor: Fair Competition Principle in Biological and Societal Systems 95
4.2.2 The HFC Model..... 97
4.2.3 The HFC Principles for Sustainable Evolutionary Search 102
4.2.4 Designing Sustainable Evolutionary Algorithms Based on HFC 103
4.3 Generational HFC Algorithms: Static and Adaptive 104
4.3.1 The Motivation & Design Rationale 104
4.3.2 The Algorithm Framework 106
4.3.3 Test Suites..... 112
4.3.4 Experimental Results..... 114
4.4 CHFC: HFC Algorithm with Single Population, the Continuous HFC..... 121
4.4.1 The Motivation & Design Rationale 121
4.4.2 The Algorithm Framework 122
4.4.3 Test Suite 128
4.4.4 Experimental Results..... 129
4.5 HEMO: HFC Algorithm for Multi-objective Search 132
4.5.1 The Motivation & Design Rationale 132
4.5.2 The Algorithm Framework 134
4.5.3 Test Suites..... 137
4.5.4 Experimental Results..... 138
4.6 QHFC: Quick HFC Algorithm with Adaptive Breeding Mechanisms..... 140
4.6.1 The Motivation & Design Rationale 141
4.6.2 The Algorithm Framework 145
4.6.3 Test Suites..... 145
4.6.4 Experimental Results..... 146
4.7 Why and How to Use HFC for Sustainable Evolutionary Search 152
4.7.1 The Advantages of HFC-Based Evolutionary Algorithms 152
4.7.2 Relationship of HFC with Other Techniques to Sustain Evolution..... 157
4.7.3 Which One to Use: Comparison of HFC Family EAs..... 159
4.8 Summary 162

5 SCALABLE EVOLUTIONARY SYNTHESIS 164
5.1 The Scalability Issue of Evolutionary Synthesis 164
5.2 The Fundamental Principles for Handling Complexity 165
5.3 The Limitations of the Current GP Approach for Scalable Synthesis 169

5.4 The State –of–the Art in Scalable Evolutionary Synthesis 171
5.5 Future Directions 175

6 STRUCTURE FITNESS SHARING FOR SCALABLE EVOLUTIONARY SYNTHESIS 179
6.1 Balanced Topology and Parameter Search in Evolutionary Synthesis 179
6.2 Structure Fitness Sharing (SFS) 188
6.2.1 Labeling Technique in SFS 190
6.2.2 Hash Table Technique in SFS 190
6.2.3 The Structure Fitness Sharing Algorithm 191
6.3 Experiments and Results 191
6.3.1 Problem Definition and Experiment Setting 191
6.3.2 Experimental Results 194
6.3.3 Discussions and Summary 198

7 REPRESENTATION IN GENETIC PROGRAMMING FOR EVOLUTIONARY SYNTHESIS 201
7.1 The Representation Problem in Evolutionary Computation 201
7.2 The Representation Problem in GP-based Evolutionary Synthesis 203
7.3 Three Approaches to Evolve Bond Graph-Based Dynamic Systems 205
7.3.1 Basic Set Approach 206
7.3.2 Node-Encoding Approach 208
7.3.3 Hybrid-Encoding Approach 212
7.4 Benchmark Problem and Experiments 213
7.4.1 Experiment 1: Search Bias in Representation 215
7.4.2 Experiment 2: The Effect of Function Set on Performance 218
7.4.3 Experiment 3: Knowledge-Based Initialization Improves Performance 221
7.5 Summary 222

8 CONCLUSIONS 226
8.1 Summary 226
8.1 Main Conclusions 229
8.2 Future Research 230
8.2.1 Characteristics of HFC, Based on Extensive Experimental Studies 231
8.2.2 Adaptive Breeding for Continuous HFC 232
8.2.3 Application of HFC to Evolution Strategies 233
8.2.4 Application of HFC to Other Optimization and Evolutionary Systems 233
8.2.5 Better Strategy for Simultaneous Topology & Parameter Search 233
8.2.6 Evolvability in Evolutionary Synthesis of Bond Graphs 234
8.2.7 Development and Modularity in Evolutionary Synthesis 234

BIBLIOGRAPHY 237

LIST OF TABLES

Table 2.1 A skeleton of an evolutionary algorithm 16

Table 2.2 Outline of a classical genetic algorithm (GA) 17

Table 2.3 The preparatory steps of GP for symbolic regression problem 22

Table 2.4 Algorithmic procedure of evolution strategies 26

Table 3.1 Building blocks of bond graphs 68

Table 3.2 Types of building blocks and modifiable sites 71

Table 3.3 Definition of function set and terminal set 75

Table 4.1 Static generational HFC algorithm 108

Table 4.2 The generational HFC algorithm with adaptive admission thresholds (HFC-ADM) 111

Table 4.3 Shared parameters for even-10-parity problem 115

Table 4.4 Parameter Setting for HFC algorithms 115

Table 4.5 Parameter setting for 8-eigenvalue placement problem 118

Table 4.6 Comparison of average best-of-run distance error for 8-eigenvalue problem 118

Table 4.7 The t-test results for four GP algorithms to an 8-eigenvalue placement problem. . 119

Table 4.8 Comparison of stagnation time for the 8-eigenvalue placement problem. 119

Table 4.9 The Continuous HFC (CHFC) Algorithm 123

Table 4.10 Parameters Used in the Various GP Experiments on Santa Fe Trail 129

Table 4.11 Peformance comparison of CHFC and conventional GP in Santa Fe Trail artificial ant problem 129

Table 4.12 The HFC algorithm for multi-objective search (HEMO) 135

Table 4.13 Comparison of the robustness of PESA and HEMO with test function ZDT4 .. 139

Table 4.14 Opportunistic PESA and Robust HEMO 140

Table 4.15 Quick HFC (QHFC) Algorithm 143

Table 4.16 DCGA (-MR): Deterministic Crowding GA with/out multi-partial-reinitializations 146

Table 4.17 Comparison of features of HFC family algorithms 161

Table 6.1 The Structure Fitness Sharing algorithm 191

Table 6.2 Target Eigenvalues 192

Table 6.3 Parameter settings for experiments 193

Table 7.1 Three types of target bond graph topologies 216

Table 7.2 Topological transformation with different initial homogenous populations 222

LIST OF FIGURES

Figure 2.1 Two point crossover and mutation of genetic algorithms.. 18

Figure 2.2 Two randomly initialized tree-like programs 23

Figure 2.3 Crossover (left) and mutation (right) in GP on program trees.................................... 23

Figure 2.4 A system for design synthesis 45

Figure 3.1 Elements of blocks diagrams versus signal flow graphs.. 61

Figure 3.2 Relational graph (Squares represent relations; circles represent entities or terms) ... 61

Figure 3.3 Analog circuit and Bond graph representation of dynamic systems. 62

Figure 3.4 Graphical models representing computational structures (Cellier, 1991).................. 63

Figure 3.5 Bond graph representation of a mechanical system and an electric circuit................ 67

Figure 3.6 The causality concept in bond graphs 69

Figure 3.7 A typical GP tree 70

Figure 3.8 An embryo bond graph with one junction-type modifiable site at a 1-junction....... 72

Figure 3.9 Illustration of insert_X and add_X functions... 74

Figure 4.1 Enormity of the search space and of the number of local optima and
expansion in horizontal spreading EAs……………….…………………………85

Figure 4.2 The assembly line structure of the continuing EA model... 88

Figure 4.3 The fair competition model in educational systems of China 96

Figure 4.4 The streamlined structure of the HFC model 99

Figure 4.5 The organizational structure of the generational HFC model.................................. 104

Figure 4.6 Floating subpopulation & adaptive allocation of subpopulations to fitness levels 112

Figure 4.7 Embryo bond graph in eigenvalue placement problem ……………………………..113

Figure 4.8 Comparison of success rates after 300,000 evaluations.. 116

Figure 4.9 Comparison of average best-of-run fitness for HFC-GP and standard GP 117

Figure 4.10 Comparison of the average best-of-run location errors vs. number of evaluations for multi-population-GP and HFC-ATP. 120

Figure 4.11 Structure of the continuous HFC model. 122

Figure 4.12 Seven types of distribution of the breeding probability of all fitness levels........... 125

Figure 4.13 The convergent nature of conventional generational GP 130

Figure 4.14 Evolution of the individual distribution in the dimension of fitness of CHFC.... 131

Figure 4.15 Performance comparison of CHFC with steady-state and generational GP......... 132

Figure 4.16 Evolution pattern of individual distribution in NSGA-II. 133

Figure 4.17 The assembly line structure of HEMO Framework. 137

Figure 4.18 Distribution of individuals over objective space of GWK in HEMO 138

Figure 4.19 Comparison of QHFC with DCGA with/without multiple partial reinitializations in terms of robustness of the search capability and robustness in terms of population size 147

Figure 4.20 Comparing QHFC with DCGA with/without partial reinitializations in terms of average number of evaluations needed to find the optimal solutions. ... 149

Figure 6.1 The embryo bond graph model. 193

Figure 6.2 Experimental results on 6-eigenvalue problem. 196

Figure 6.3 Experimental results on 8 eigenvalue problem. 197

Figure 6.4 Experimental results on 10 eigenvalue problem 198

Figure 7.1 A causally well-posed bond graph of a low-pass analog filter. 205

Figure 7.2 Example of a redundant bond graph model and its simplified equivalent.........207

Figure 7.3 GP function set for node-encoding approach 209

Figure 7.4 Switches for elements attached to junctions………………………...…………….210

Figure 7.5 Function Insert_J0C_J1I_R and terminal EndBond in hybrid encoding approach for bond graph evolution. 211

Figure 7.6 Search effort to find three types of target topologies for 8-, 12-, 16-, 18, 20-eigenvalue problems with the node-encoding approach, averaged over 20 runs... 217

Figure 7.7 Comparison of hybrid encoding with node encoding approach for 8-, 12-, 16-, 18-, and 20-eigenvalue problems 219

Chapter 1

INTRODUCTION

From the mechanical man of ancient China in the third century B.C. (Needham, 1975) to the latest Sony Aibo electronic dogs, human beings have been dreaming about building intelligent machines that emulate the biological intelligence of living organisms. However, investigation of intelligent machines became serious only with the emergence of modern computers. In a pioneering paper entitled "*Intelligent Machinery*", Alan Turing (1948) outlined three approaches to machine intelligence, including the logic-driven search approach, the cultural search approach, and the evolutionary search approach. The stagnation of traditional symbolic artificial intelligence motivated largely by the first two approaches implies that, to emulate biological intelligence, studying biology itself rather than our symbolic abstractions is almost a necessity. Then the great evolutionary geneticist Theodosius Dobzhansky (1973) said, "Nothing in biology makes sense except in the light of evolution". And then Stuart Kauffman, the molecular biologist at Sante Fe Institute, told us that the order of the biological world is the result of not only natural selection but also self-organization of physical entities (Kauffman, 1993). Based on these ideas, our projection of artificial intelligence research is the following:

> To understand and emulate biological intelligence and to evolve solutions to complex problems, close investigation of biological evolution is a necessity, not just about the algorithmic aspect of the principle of evolution outlined by Charles Darwin, but all aspects of natural evolution such as the physical properties of the natural environments, the biological molecules and their interactions, and the developmental process.

This dissertation investigates additional aspects of the biological evolution process that we can model to improve the sustainability and thus the scalability of artificial evolution approaches for solving complex problems. It also analyzes some general principles for handling complexity for building large systems employed in both deliberately built artificial systems and blindly-tinkered bio-systems, which are not yet modeled in current artificial evolution approaches but are promising to significantly improve their scalability. Finally it investigates techniques for balanced topology and parameter search in evolutionary synthesis and function set design issue in genetic programming for evolutionary synthesis as well as the relationship between representation and evolvability in the context of evolutionary synthesis of dynamic systems using bond graphs.

1.1 Motivation and Background

In the May 2003 issue of the journal *IEEE Intelligent Systems*, genetic programming with its duplication or invention of a dozen human-competitive results is regarded as one of the major achievements of artificial intelligence (AI) in the 21st Century. A distinguishing feature of genetic programming compared to five other AI techniques considered is that genetic programming is the only approach that provides genuine open-ended solutions to given problems. This is one of the first few acknowledgements from the mainstream AI community of the capability of the blind evolutionary tinkering process (Dawkins, 1986). Indeed, evolutionary computation has become a powerful automated problem-solving tool and one of the central approaches to investigate all kinds of complex adaptive systems.

The broad area of applying evolutionary computation techniques to evolve complex machines is called topologically open-ended evolutionary synthesis. Along with the human competitive results of John Koza et al. (2003a, 2003b), evolutionary synthesis has infiltrated almost all areas of engineering design. To give a glimpse, the following applications have been sought by researchers. Examples include but are not limited to, analog and digital circuit design (Koza, 1994), metabolic path synthesis (Koza, 1999), control systems (Koza, 1999), telecommunication networks (Aiyarak et al., 1997), transmission mechanism synthesis (Kunjur, 1995; Fang, 1994), neural network synthesis (Yao, 1993), truss structure synthesis (Deb and Gulati, 2001), graph synthesis (Luke & Speacor, 1996), transportation network synthesis (Vitetta, 1997), mechatronic system synthesis (Seo et al., 2002), morphology and control system synthesis in robotics (Eggenberger, 1996; Funes and Pollack, 1998;), and evolvable hardware (Yao & Higuchi, 1999), etc.

Evolution in Nature works without understanding. There is no conscious mind that figures out the physical and chemical laws and designs astonishing artifacts like a human being. This amazing achievement is evolved by the clumsy but continuing innovation process of natural evolution, through billions of years of "blind watchmaker tinkering" (Dawkins, 1986).

Evolutionary algorithms (EAs) emulate the natural evolutionary process by employing a population of individuals and updating it continuously using variation operators such as crossover or mutation based on fitness-based selection. A prominent feature of evolutionary search is that given any representation scheme of the potential solutions and corresponding variation operators, the system can usually come up with some unusual and interesting results. EAs can be used for both topology synthesis and parameter optimization. This unconstrained search capability compared to other traditional optimization or search algorithms makes it very suitable for evolutionary synthesis of systems, where the structure search is critical.

However, like many evolutionary algorithms that may surprise you initially but not much later, the evolutionary approaches to system synthesis have not been able to generate significantly complex solutions compared to those evolved in natural evolution. Except for the automatically synthesized analog circuits with a hundred or so components (Koza, 1999) and some evolvable hardware circuits with simple functions, evolutionary synthesis seems to be strongly limited by some inherent difficulty, especially considering the huge computing power of a 1000-PC cluster available to Koza (1999). It is clear that there are some critical obstacles other than the computational power that need to be addressed before the ambition of building truly topologically open-ended invention machines can be realized.

Two challenges need to be addressed before we can achieve scalable topologically open-ended evolutionary synthesis. The first is the sustainability of the evolutionary algorithms. Sustainability here is defined as the capability to make additional progress given more computation resource unless the optimum is reached. Most existing evolutionary algorithms are inherently convergent and usually stagnate after a certain number of generations, however many more evaluations are given. The second challenge is the scalability related issues in topologically open-ended evolutionary synthesis. Examples include evolvability of the search space, emergence of modularity and hierarchy during evolution, genotype-phenotype mapping. Usually, it is the representation of the search space and corresponding genetic operators that determine the evolvability of the problem and thus the degree of complexity the evolved solutions could have. Guided by understanding of the fundamental principles underlying the sustainability, reliability, and scalability of evolving biological systems, this dissertation addresses these challenges by seeking effective techniques to improve the sustainability and reliability of evolutionary algorithms and the scalability of topologically open-ended evolutionary synthesis.

1.2 Problem Statement

We are interested in developing scalable evolutionary techniques to solve challenging real-world problems reliably and efficiently. In particular, the ultimate goal is to obtain the capability to automatically synthesize complex systems and inventions. Unfortunately, despite half a century's effort in the field of evolutionary computation, the art of artificial evolution still lags far behind natural evolution in terms of efficiency, robustness, and scalability.

One significant problem for all evolutionary algorithm practitioners is to handle premature convergence or stagnation problem. The evolutionary algorithms usually get trapped before obtaining satisfactory results and no additional progress can be made whatever more computations are allocated. To prolong the effective evolutionary search process, all kinds of tricks have been invented to check this degradation of search capability such as increasing population size, tuning crossover and mutation rates, and dynamically adapting running parameters. Unfortunately, these tricks can at most only delay the convergence process and fail to check the general trend of losing search capability. The issues of sensitivity of search performance of EAs with respect to running parameters remain unchanged. In this lock-in situation, we should ask, what is possibly wrong with our fundamental evolutionary computation model?

It is clear that current evolutionary computation models simulate a key algorithmic aspect of Darwin's evolution theory, the "survival of fittest" principle of natural selection. However, natural selection does not capture other important aspects of natural evolution, including the properties of the natural environment and of physical processes. The inherently convergent nature of the existing evolutionary algorithms arises from this inadequate modeling of the sustainable evolutionary process of natural evolution endowed by the hierarchical niches inherent in the natural environment. In this dissertation, we are interested in the following questions. First, what other aspects of the natural evolutionary process do we need to model to make our evolutionary algorithm more robust and sustainable? What are the underlying principles that support the sustainability of the natural evolution process? How can these principles be transformed into evolutionary algorithm techniques to ensure sustainable evolutionary search? Is it possible to improve the sustainability and robustness of evolutionary search and also to improve its efficiency, which is usually regarded to be impossible? What are the general principles to design a sustainable evolutionary search?

In the past few years, increasing attention has been targeted toward modeling the biological developmental mechanism in biological evolution to artificial evolutionary synthesis. However, as the exact understanding of the principles underlying biological developmental processes is not available, ad-hoc attempts to incorporate the developmental mechanism into evolutionary synthesis have largely failed to achieve significant progress in terms of scalability, actually even much less effective compared to traditional genetic programming. In this case, it is critical to emulate the biological evolutionary developmental process not just by copy-and-paste ad-hoc way, but based on an examination of the possible general principles that underlying the scalability of developmental way to build complex multi-cellular living organisms. For example, we may ask what possible role the introns and other non-encoding DNA may play in terms of evolvability of biological systems. This dissertation will explore several such principles to handle complexity employed by both human engineers and by the biological evolution process.

Another fundamental difficulty that limits the scalability of evolutionary synthesis is the parameter search scheme. Without a clear modularity concept in current GP-based evolutionary synthesis, the clumsy strategies used today to search a parameter space with hundreds of variables make it almost impossible to scale up. One serious problem in current GP-based evolutionary synthesis is the loss of topology search capability or the premature convergence of structures –the topologies in the population tend to get increasingly more homogenous. This phenomenon shares some fundamental causes with the convergent evolution model which can be addressed with the proposed sustainable evolution model. But another direct cause of this problem is the lack of explicit control of topology and parameter search for individuals in the population such that one or a few topologies with high-fitness tend to dominate the population. How to achieve balanced topology and parameter search to maintain sustainable topology search capability is one issue to be addressed in this dissertation.

Another difficulty to design GP-based synthesis engine is the selection of function set that determines its evolvability and thus scalability. What aspects of representation and operators provide for high evolvability of topology search space in topologically open-ended evolutionary synthesis? How do different representations affect their scalability? What kind of variation operators can facilitate the evolution of more complex solutions? What are the issues that need to be taken care of for scalable evolutionary synthesis in which the topology and parameters have to be evolved simultaneously? What kinds of techniques can we use to enable the evolutionary engine to synthesize large-scale dynamic systems with bond graphs and genetic programming? Is there any inherent limitation of current developmental genetic programming for scalable evolutionary synthesis?

We believe that these questions are important and critical to ensure that in the near future, evolutionary invention machines can routinely generate inventions and innovative solutions using reasonable amount of computational resources by any engineers.

1.3 Proposed Evolution Models and Techniques

Investigations in this dissertation consist of examinations of the fundamental principles that underlie the sustainability, robustness, and scalability of biological evolution and corresponding techniques inspired by these principles to address the limitations of current evolutionary algorithms and evolutionary synthesis approaches.

After a close analysis of the fundamental causes of premature convergence, it is identified that many problems with current EAs, such as lack of reliability and scalability, are derived from the underlying convergent evolution model. This dissertation proposes a new model, the Hierarchical Fair Competition (HFC) model, for sustainable evolutionary computation. This sustainable search capability is achieved by ensuring a continuous supply and incorporation of genetic material in a hierarchical manner, and by culturing and maintaining, but continually renewing, populations of individuals of intermediate fitness levels. HFC employs an assembly-line structure in which subpopulations are hierarchically organized into different fitness levels, reducing the selection pressure within each subpopulation while maintaining the global selection pressure to help ensure exploitation of good genetic material found. The structure of HFC does not allow convergence of the population to the vicinity of any set of optimal or locally optimal solutions, and thus it essentially transforms the convergent nature of the current evolutionary algorithm framework into a *non-convergent* search process. A paradigm shift from that of existing evolutionary algorithms is proposed: rather than trying to escape from local optima or delay convergence at local optima, HFC allows the continuing emergence of new or repeated optima in a bottom-up manner, maintaining low local selection pressure at all fitness levels, while fostering exploitation of high-fitness individuals through promotion to higher levels. Compared with previous techniques in the traditional evolutionary computation framework aimed at achieving sustainable evolution, HFC-based evolutionary algorithms have a number of significant beneficial characteristics:

1) Evolutionary algorithms based on the HFC model can achieve remarkable, sustainable search capability compared to conventional evolutionary algorithms with all kinds of enhanced techniques. HFC is an inherently non-convergent search process and offers to provide better solutions given more computation.

2) Evolutionary algorithms based on the HFC model can achieve much higher reliability in terms of search performance. It greatly changes the opportunistic nature of conventional evolutionary algorithms. This may make it possible to incorporate evolutionary algorithms into critical application areas.

3) Evolutionary algorithms based on the HFC model have shown strong robustness with respect to parameter setting, which is one of the most severe criticisms of evolutionary algorithms.

4) Evolutionary algorithms based on the HFC model can also achieve greatly improved efficiency while enjoying reliability and robustness. Compared to naïve restarting approaches triggered by the stagnation of conventional evolutionary algorithms, HFC makes full use of the search experience all the time rather than restarting from scratch again and again. The HFC model provides a win-win solution to the exploration and exploitation dilemma of evolutionary computation: one can run as fast as possible only if this aggressiveness is backed up by supporting populations. Especially, HFC provides a natural algorithmic process for hierarchical building block or stepping stone discovery and exploitation, which are deemed critical for scalable evolutionary computation.

5) As a generic framework, many existing techniques for single-level evolutionary algorithms can be combined with HFC algorithms and achieve synergetic effects.

The HFC sustainable evolution model extends the current evolutionary computation framework essentially by incorporating several inherent properties of natural evolution, the hierarchical niches in the natural environment and the vertical speciation of living organisms. According to HFC, Darwinian evolution theory only captured the evolution process happening in a single niche or a single level of niches. The whole story of natural evolution should be interpreted as concurrence of billions of interrelated local Darwinian processes in all hierarchical niches.

In the part of scalable evolutionary synthesis, four general principles including modularity, reuse, context, and generative representation are reexamined under the context of topologically open-ended synthesis. Two issues for achieving scalable evolutionary synthesis using GP are investigated. One is the balanced topology and parameter search problem, which leads to the structure fitness sharing technique. The other issue is the representation or function set design problem for GP-based evolutionary synthesis, which leads to the modular set approach for bond graph-based evolutionary synthesis of dynamic systems using genetic programming.

1.4 Contributions and Significance

The main contributions of this dissertation include:

- Proposing a new hypothesis to explain the premature convergence/stagnation in evolutionary search

- Proposing an unconventional sustainable evolutionary computation model named the hierarchical fair competition model. This model lays down the foundation for developing a new generation of scalable and robust evolutionary algorithms;
- Developing a multi-population sustainable evolutionary algorithm and its two adaptive versions (AHFC) based on the HFC model. This algorithm is easy for parallel implementation.
- Developing a single-population continuous sustainable evolutionary algorithm (CHFC) based on the HFC model, which is amenable to future theoretical analysis and is easy to implement within existing evolutionary algorithm packages.
- Developing an efficient, quick, sustainable evolutionary algorithm (QHFC) based on the HFC model by inventing an adaptive breeding strategy such that an evolutionary algorithm can run as fast as possible as long as this aggressiveness is backed up by the supporting populations.
- Developing a multi-objective sustainable evolutionary algorithm (HEMO) based on the HFC model.
- Demonstrating the sustainability, scalability, and reliability of the HFC model for sustainable evolutionary search by extensive experimental study.
- Formulating the balanced simultaneous topology and parameter search issue in topologically open-ended evolutionary synthesis and proposing the Structure Fitness Sharing technique to address it.
- Investigating the evolvability issue of bond graph evolution by genetic programming including function set and genetic operator design and proposing several techniques to improve the scalability of existing evolutionary synthesis of bond graphs

- Applying the HFC based sustainable genetic programming and Structure Fitness Sharing techniques to bond graph evolution.
- Establishing a benchmark problem, the eigenvalue placement problem, for bond graphs-based topologically open-ended synthesis of dynamic systems.

1.5 Organization of the Dissertation

The rest of the dissertation is organized into seven chapters:

Chapter 2 describes the background of the field of evolutionary computation, emphasizing the historical origin of the canonical evolutionary computation model. A comprehensive survey of previous work aimed to improve the sustainability of conventional evolutionary algorithms is then presented, followed by an analysis of how the inherently convergent nature of the canonical evolutionary computation model incurs difficulties for these techniques. The second half of this chapter provides a background of topologically open-ended evolutionary synthesis and a survey of available techniques.

Chapter 3 provides a historical background of topologically open-ended synthesis of dynamic systems, followed by the discussion of the representation issue of dynamic system synthesis. A framework for automated synthesis of dynamic systems using bond graphs and genetic programming is then formulated. A benchmark problem –the eigenvalue placement problem is presented at the end of the chapter.

Chapter 4 begins with a thorough analysis of the premature convergence/stagnation problem of conventional evolutionary algorithms and then describes the Hierarchical Fair Competition model for sustainable evolutionary computation, including the original metaphor that inspired its discovery. A set of requirements for designing sustainable evolutionary algorithms is then outlined. The rest of the chapter describes five sustainable evolutionary algorithms and their evaluation with a large number of genetic programming, and genetic algorithm benchmark problems and real-world problems. This chapter then concludes by summarizing the benefits of the HFC model for evolutionary algorithm development.

Chapter 5 examines the fundamental principles that underlie the scalability of building complex systems either by human engineers or by biological evolution process. Within the context, the limitations of current genetic programming for evolutionary synthesis are discussed and future research directions to address these limitations are proposed.

Chapter 6 presents the issue of balanced topology and parameter search in evolutionary synthesis. A corresponding technique, the structure fitness sharing (SFS) technique is introduced and evaluated with the benchmark problem proposed in Chapter 3.

Chapter 7 investigates the issue of representation or more specifically how function set design of GP would affect its performance. Two modular set approaches, node-encoding and hybrid-encoding, are proposed for synthesizing causally-well posed dynamic systems using bond graphs.

Chapter 8 summarizes the main results of the research and presents some conclusions. Some promising future research topics are described as a natural extension of this work.

Chapter 2

BACKGROUND AND RELATED WORK

Encouraged by the fact that natural evolution has generated living organisms with staggering complexity and extraordinary intelligence, researchers have been pursuing sustainable and scalable evolutionary computation techniques to evolve solutions to complex problems since the1950s. This chapter gives a brief introduction to major evolutionary algorithms (EAs) and examines their key ideas supporting sustainable evolution. This is followed by a survey of previous work to achieve sustainable and scalable evolutionary search. Fundamental issues of existing techniques for sustainable evolutionary search are then analyzed, which show that much of the difficulty arises from the common framework of existing evolutionary algorithms. The second half of this chapter then introduces the typical evolutionary techniques for topologically open-ended synthesis and the investigations aiming to improve the scalability.

2.1 Overview of Evolutionary Algorithms

Evolutionary algorithms belong to the broad area of meta-heuristic iterative search techniques, which work by repeatedly probing the search space, guided by some sort of memory of the information collected during the search process. This memory can be a population of solutions in EAs (De Jong, 2002), a single solution in simulated annealing (Kirkpatrick et al., 1983), a singe solution along with a tabu list in tabu search (Glover, 1989), or a distributed representation of the local decision information for constructing a complete solution in ant colony optimization (ACO) algorithms (Dorigo et al, 1999). Regarding the population as the memory or summary of past search experience enables us to understand a variety of EAs under a common framework.

2.1.1 The Darwinian Root of Evolutionary Algorithms

Evolutionary algorithms are a set of techniques inspired by the natural evolution process, more specifically, by Darwin's theory of evolution by natural selection. In "Origin of Species", Darwin made the following observations about the natural evolution process:

- A huge number of species of organisms are living on the earth. Each species has an enormous number of individuals –the population
- Resource in a given environment is limited and so only a limited number of organisms can be accommodated, leading to competition for survival—the selection process
- Surviving organisms multiply by asexual or sexual reproduction, during which random mutations often occur, and most of the characteristics of the parent(s) are inherited – inheritance with modification

Natural selection as a general principle underlying evolution has been well established. The fundamental algorithmic nature of the natural evolutionary process has inspired many of the pioneers of evolutionary computation to devise their artificial evolutionary algorithms, including, but not limited to, the widely known genetic algorithms, evolution strategies, evolutionary programming, and genetic programming. Briefly, all the EAs can be summarized with the following algorithm skeleton:

However, it has been argued that natural selection alone cannot explain the complexity of living organisms and must be complemented by the self-organization theory of complex systems (Kauffman, 1993). Kauffman defined a concept of "order for free" as the self-organization phenomenon arising from the inherent properties of physical building blocks. One question over the classical evolutionary algorithm models is whether the algorithmic abstraction in Table 2.1 is sufficient to support sustainable artificial evolution. And if it is not, what kind of other features of the natural evolutionary process must we model to devise sustainable evolutionary algorithms to evolve solutions to complex problems. Indeed, as is to be discussed in Section 2.2, most of the techniques invented so far to sustain evolutionary search are inspired by modeling other aspects of natural evolution in addition to the bare algorithmic process in Table 2.1.

Table 2.1 A skeleton of an evolutionary algorithm

```
Procedure EA
Initialize $P(0)$
$t \leftarrow 0$
 do until termination_criteria ($P(t)$)
     evaluate $P(t)$
     $P'(t) \leftarrow$ select from $P(t)$
     $P''(t) \leftarrow$ reproduce from $P'(t)$
     $P(t+1) \leftarrow$ replace with individuals from $P''(t)$ or $P(t) \cup P''(t)$
     $t \leftarrow t+1$
end
return best solution(s)
```

Despite the common root of Darwinian evolution theory, historically, existing evolutionary algorithms arose with quite different research backgrounds of their inventors. These differences lead to their specific characteristics and their preference regarding the representation, variation operators, and selection/reproduction operators. It is clear that each of them provides some unique features that may contribute to the development of sustainable evolutionary algorithms.

2.1.2 Genetic Algorithms

Genetic algorithms are a set of search techniques that simulate the genetic systems of living organisms. The earliest exploration of evolving purposive and adaptive behavior by emulating genetic systems was proposed by biologist Fraser in 1957, in which he developed a genetic simulation system with binary representation, recombination, and selection, etc (Fraser, 1957).

Important concepts such as genotype and phenotype mapping, and the linkage as the interaction of loci were also investigated. Two other biologists, Bremermann (1962) and Reed (1967) also proposed similar ideas. However, it is John Holland, with his interests in biology and his background in computer science, who conceived the current widely known genetic algorithms as a means of studying adaptive behaviors of natural and artificial systems by simulating the genetic systems, beginning in the 1960's and published in a book form in (Holland, 1975). The side interests of Holland and his students to use genetic algorithms as a set of easy-to-use global optimization methods corroborated by mathematical theory contributed to the popularization of genetic algorithms in optimization.

In the classical genetic algorithm, a population of individuals represented as fixed length binary strings compete for reproduction based on the fitness of their phenotypes decoded from their genotypic binary strings. The successful competitors are then subjected to recombination and a low probability of random mutation at each locus. These offspring will comprise the next generation population. The outline of the classical genetic algorithm is listed in Table 2.2

Table 2.2 Outline of a classical genetic algorithm (GA)

```
Initialize $P(0)$, where $P(t)$ is the population at time t
evaluate $P(0)$
$t \leftarrow 0$
do until termination_criteria ($P(t)$)
    $P'(t) \leftarrow$ select parents from $P(t)$
    $P''(t) \leftarrow$ recombine and mutate $P'(t)$
    evaluate $P''(t)$
    $P(t+1) \leftarrow P''(t)$
    $t \leftarrow t+1$
end
return best solution(s)
```

The evaluation of an individual in genetic algorithms is conducted by a genotype-phenotype mapping (decoding) process. For example, a 16-bit binary string 1001000010001111 might be decoded as two integer numbers (144, 143), depending on the encoding scheme. In the selection process of a canonical GA, each individual is assigned a probability of reproduction so that the likelihood of selection is proportional to its fitness relative to the other individuals in the population. This selection scheme is denoted as *fitness proportionate selection.* The recombination operation is accomplished by the one-point or two-point crossover of two parents (Figure 2.1). For each pair of parents selected for crossover, the crossover probability is p_c; otherwise, the parents are simply copied to the next generation $P''(t)$. For each offspring of the crossover, a random mutation operation is applied to each allele with a small probability to switch 1/0 states.

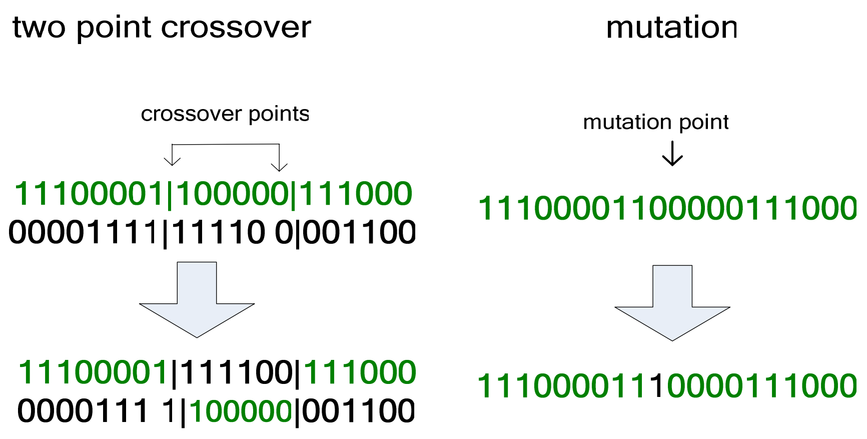


Figure 2.1 Two point crossover and mutation of genetic algorithms. One point crossover works by assuming the second crossover point is located at the end of chromosomes.

There are several important concepts in genetic algorithms critical to develop sustainable evolution. The most important idea is the operator of recombination, which distinguishes genetic algorithms and its derivative genetic programming from other EAs like evolution strategies and evolutionary programming and other global optimization algorithms. Recombination, the process whereby a new individual solution is created from the information contained within two (or more) parent solutions, is an extremely important strategy for evolving complex solutions. Sexual recombination of living organisms is essentially an effective reuse strategy. By combining good features from parents, the offspring do not need to reinvent the wheels again and again and enjoy both features. The ubiquitous existence of sexual reproduction in high order living organisms clearly demonstrates the advantage of recombination. Actually, recombination is also one of the most important characteristics of biological evolution underlying Darwin's examination of artificial cross-breeding (Darwin, 1859). Another demonstration of the power of recombination is at least partially illustrated by comparing the success of modern genetic programming (Koza, 1992) and the failure of an early attempt to evolve programs (Friedberg, 1958). The former employs the crossover as the major operator while the later only uses random mutation. In addition to the effect of assembling good features and building blocks, recombination also has the effect of genetic repairing, making the offspring more able to survive (Beyer, 1995).

Another important contribution of Holland's theory of genetic algorithms is the concept of building block, which is related to recombination. Only by evolving different levels of building blocks and modules and innovative ways of exploiting them, can the biological evolution evolve the complexity that we see today. With respect to mutation, evolution by building block discovery and composition can win over mutation by orders of magnitude in efficiency, as explained by Herbert Simon (1973). Despite decades of effort in genetic algorithms to invent new techniques to discover and exploit building blocks, it remains as one of the critical challenges to build sustainable evolutionary algorithms (Lipson, Antonsson & Koza, 2003).

The third contribution of genetic algorithms may be the genotype-phenotype mapping. Using a unified binary representation and a domain-specific decoding process, a genetic algorithm can be applied to many kinds of search or optimization problems. This is in sharp contrast of evolution strategies and other traditional optimization algorithms, which are limited to numerical optimization. In addition, the genotype-phenotype mapping of organic evolution, that is, the developmental stage of most multi-cellular organisms, is regarded as one of the most promising directions to scale up existing evolutionary algorithms (Stanley & Miikkulainen, 2003).

However, there are two aspects of canonical genetic algorithms that do not follow the natural evolution closely. One is the generational population update model, where the population of generation t is updated only when all individuals of generation t+1 are produced. This global synchronization process is not necessary and not suitable for parallel implementation. It is usually replaced by a steady-state model of population update, which is widely used in evolution strategies. An example of a steady-state genetic algorithm is the Genitor GA (Whitley, 1989). Another global operation in canonical genetic algorithms is the fitness-proportionate selection operator, which must collect the fitness of all individuals to allocate breeding opportunities for individuals in the current population, while in natural evolution the relative fitness is always evaluated by local competition. This idea leads to the wide use of tournament selection, where a number of randomly selected individuals compete against each other and the best one is selected. In addition to its simplicity, tournament selection is also considered as one of the best selection methods for genetic algorithms (Blickle & Thiele, 1996). Assuming the same selection intensity, it is proved that tournament selection has the smallest loss of diversity and the highest selection variance. We believe that a sustainable evolutionary algorithm should try to avoid using global information or operations.

2.1.3 Genetic Programming

Genetic programming is an extension of the genetic algorithm into the area of computer program induction by evolutionary search (Koza, 1992). The earliest attempt to evolve computer programs by evolution was investigated by Friedberg in the late 1950s (Friedberg, 1958; 1959). By making random modifications to the programs and testing the modified programs, Friedberg succeeded only in learning some trivial programs. It demonstrated the feasibility of computers to solve problems without being told how to solve them. The first attempt to apply genetic algorithms to tree-like program induction was proposed by Nichael Lynn Cramer in 1985 (Cramer, 1985). Some important concepts including the tree-representation of programs, the closure requirement (search in the space of syntactically correct programs), subtree crossover, and the computational primitives were proposed. However, Cramer used only a population size of 50 with an evolution of 13 generations and only demonstrated a very trivial result of genetic programming. It was John Koza, who deeply understood and explored the power of program induction by evolution and established the field of genetic programming through extensive demonstration of genetic programming as a domain-independent method that breeds a population of programs to solve problems. Later analysis showed that to emancipate the power of genetic programming derived from the classical genetic algorithm model, a large population size is a necessity. This partially explains why Cramer couldn't evolve interesting results with a population size of 50.

To apply standard genetic programming to a given problem, there are five elements to specify:

1) Decide the computational primitives or atomic building blocks. This includes the functions, which need a specified number of arguments to execute, and terminals, which do not need any argument. For example, in the symbolic regression problem, a numeric expression is to be evolved to approximate a data set. The functions can be

the arithmetic operators like +, -, *, /, or sqrt, exp, or conditional branching operators, etc. A terminal can be a constant random number like 3.244, or it could be an independent variable.

2) The fitness function, which determines how to evaluate the goodness of computer programs. Usually, the fitness of a program is obtained by executing the program for some number of time steps.
3) The termination criterion
4) The control parameters and related algorithm configurations like the selection scheme, etc. One of the major features of genetic programming is that it requires a large population size. However, as to be shown in chapter 4, this requirement can be greatly loosened by the hierarchical fair computation model proposed in this dissertation for sustainable evolution

Table 2.3 illustrates a complete set of information for symbolic regression:

Table 2.3 The preparatory steps of GP for symbolic regression problem

Problem definition	Evolve a program whose output is a symbolic expression that approximate a data set $(x_i, y_i)\ for\ i = 1,...,n$
Function set F	{+, -, *, /, sqrt}
Terminal set T	{x, R} where R represents constant numeric values generated randomly during evolution and fixed thereafter
Fitness function	For all data set $(x_i, y_i)\ for\ i = 1,...,n$, input x_i to the program and calculate its output y'_i, the fitness is assigned as error = $\sum_{i=0}^{n-1}(y'_i - y_i)^2$
Termination criterion	error <0.001
Control parameters and algorithm configurations	Population size =500, crossover rate 0.95, mutation rate 0.01, tournament selection, population initialization methods, constraints on the tree size or depths

The evolution of programs in classical genetic programming starts by generating a population of random tree-represented programs. Briefly, the generation process first chooses a function from the function set as the tree root and then generates a set of random sub-trees to set as its arguments, and then the same process applies to those tree roots of the sub-trees recursively. Some limits on the tree sizes are needed to stop this recursive calling. Figure 2.2 shows two of the randomly generated programs in a symbolic regression problem:

The crossover and mutation in a genetic program are different from those of genetic algorithms. The tree-like program structures naturally hint at the invention of subtree crossover and subtree mutation as illustrated in Figure 2.3.

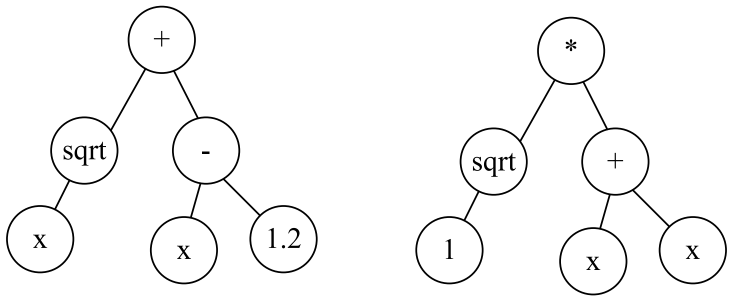


Figure 2.2 Two randomly initialized tree-like programs

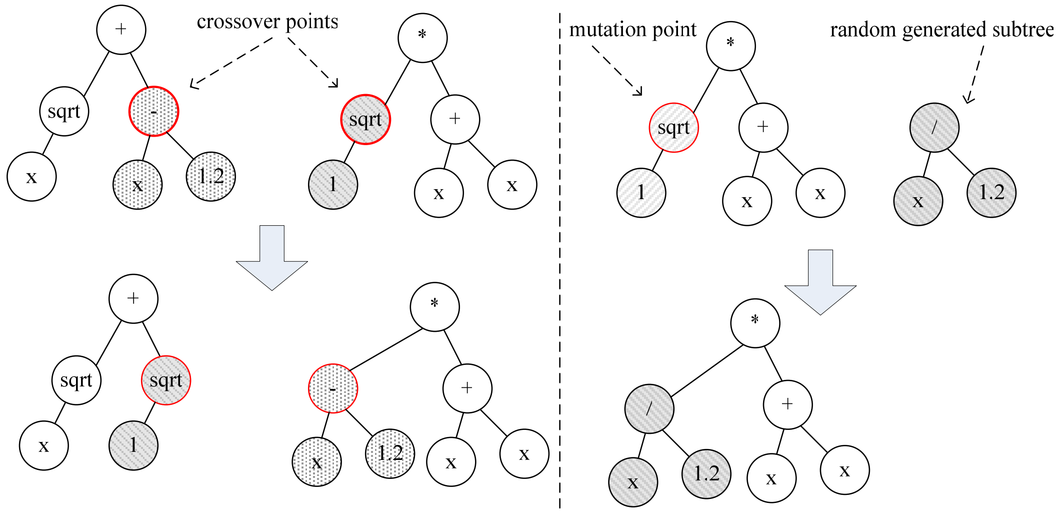


Figure 2.3 Crossover (left) and mutation (right) in GP on program trees

After the generation of the initial population, and using subtree crossover and mutation, genetic programming works in the same way as the genetic algorithm in Table 2.2, until termination.

Genetic programming has inherited most of the features of genetic algorithms, including crossover, mutation, and reproduction. It also introduces new concepts of gene duplication and gene deletion in advanced genetic programming techniques to automatically define the skeleton of automatically defined functions—high-level building blocks (Koza, 1994). As the most-investigated variable-size genetic algorithm, genetic programming has uncovered many important principles for evolving open-ended solutions to complex problems. Actually, the versatile open-ended search capability distinguishes genetic programming from most other EAs and contributes significantly to the success of evolving human-competitive solutions in analog circuits and controller synthesis (Koza, 2003a).

Looking back on the history of genetic programming, one major lesson we can get is that quantitative change can lead to qualitative changes in terms of the discovery capability of computers. Genetic programming as proposed by Koza clearly demonstrates that given sufficient population size and computing resources, what regarded as impossible -- to evolve programs by random crossover and mutation -- becomes a reality. This transformation is a significant insight of Koza's work in genetic programming.

As related to sustainable evolution, an important principle from the study of genetic programming is the realization of the importance of evolvability. Early attempts to evolve computer programs by making random modifications to the programs failed because of lack of evolvability—"a small conceptual modification to the behavior of the program is usually not represented by a small modification to the program" (McCarthy, 1987). This property of evolutionary search is labeled as "causality" by Rosca (1996). More discussion of this will follow in Section 2.2.

2.1.4 Evolution Strategies and Evolutionary Programming

In contrast with genetic algorithms that simulate the genetic system, evolution strategies and evolutionary programming simulate natural evolution only at the phenotypic level. Evolutionary programming was originally proposed by Fogel (1962, 1966), as a means to create artificial intelligence by evolution. The solutions to be evolved are represented as a finite state machine (FSM) used to predict events based on former observations. The only variation operator is mutation, modifying the graph structures of the FSM directly. Evolution strategies were introduced by Rechenberg (1973) with selection, mutation, and a population of size one and by Schwefel (1975) with recombination and populations with more than one individual.

Evolution strategies have a strong background of engineering optimization, in which a solution is represented as a real-valued vector with fixed dimension along with a vector of strategy parameters which are used to determine the perturbation magnitudes for each component. These strategy parameters are themselves perturbed during the search process. This will allow self-adaptation of the mutation magnitudes. Given $\vec{x}$ as the current solution and $\vec{\sigma}$ as a vector of variances corresponding to $\vec{x}$, a new solution is generated as:

$$\sigma'_i = \sigma_i \exp(\tau' \cdot N(0,1) + \tau \cdot N_i(0,1))$$
$$x'_i = x_i + N(0, \sigma'_i)$$

$with\ i = 1,...,n$, N(0,1) represents a single standard Gaussian random variable, $N_i(0,1)$ represents the ith independent identically distributed standard Gaussian, and $\tau\ and\ \tau'$ are parameters determining the global and individual step-sizes (Bäck and Schwefel, 1993).

Variation operator in evolution strategies was designed to simulate the mutation in evolution at the behavior level, achieved by applying a Gaussian perturbation to each component of the vector. The framework of evolution strategies (ES) and evolutionary programming (EP) can be summarized as Table 2. 4

Limited to numeric optimization, evolution strategies cannot readily be applied to topologically open-ended synthesis. But in terms of sustainable evolution, they provide some important ideas. One important concept is the self-adaptation of strategy parameters, which can automatically adjust the mutation operations during the search, while in canonical genetic algorithms, the crossover and mutation operation typically remain the same all the way.

Although DNA mutation does not work like Gaussian mutation, we believe that the mutation in the genetic systems of living organisms is influenced by the chromosome and recombination process that are evolving all the time themselves. How to implement adaptive mutation operators in genetic algorithms is not well explored despite some work addressing adaptive mutation rate (Thierens, 2002).

Table 2.4 Algorithmic procedure of evolution strategies

```
procedure ES/EP
        $t \leftarrow 0$;
  initialize population $P(t)$
  evaluate $P(t)$
  do until termination_criteria ($P(t)$)
    $t \leftarrow t+1$
    $P'(t) \leftarrow$ parent_selection $P(t)$
    $P''(t) \leftarrow$ recombine $P'(t)$ [exists only for ES]
    $P'''(t) \leftarrow$ mutate $P''(t)$
    evaluate $P'''(t)$
    $P(t+1) \leftarrow$ survive $P'''(t)$ or $P'''(t) \cup P(t)$
    return best solutions
end procedure
```

The second interesting concept of evolution strategies is correlated mutation to allow the distribution of new "trails" to adapt to contours on the error surface (Schwefel, 1981). A similar idea was also proposed in evolutionary programming, implemented as optimizing the covariance (Fogel et al., 1992). These techniques essentially try to capture how the interaction of the components affects their fitness. The underlying philosophy is closely parallel to linkage learning in genetic algorithms (Goldberg, 1989; 2002). Although building block concepts are not widely accepted in evolution strategies and evolutionary programming community, we believe that this correlated mutation concept works essentially by identifying separate mutation building blocks -- sets of coupled variables.

2.1.5 Common Framework, Issues and Pitfalls

Despite the apparent difference in terms of representation of individuals and the variation operators, all classical evolutionary algorithms share many characteristics in their frameworks, along with common issues and difficulties to achieve sustainable evolution. We have the following observation of their common features and comparison to those of natural evolution.

- The population size is usually quite limited, in sharp contrast with the huge population size in natural evolution.
- The average fitness of the population increases continuously until stagnation since only better individuals can survive under the increasing absolute selection pressure. In the late evolution stage, individuals with low fitness find it hard to survive. In nature, the diversity of environmental niches allows individuals with different levels of fitness to survive.
- All evolutionary algorithms include a selection operator that intensifies the search in promising directions, and one or more variation operators to explore new search areas. The diversification mechanisms in natural evolution are much more diverse.
- All evolutionary algorithms have to address the well-known exploration and exploitation issue existing in all exploratory learning systems including reinforcement learning, classifier systems, etc. Failing to control the balance of exploration and exploitation will lead to premature convergence in genetic algorithms and premature stagnation in evolution strategies and evolutionary programming. The balance of exploration and exploitation in natural evolution seems to be adaptively controlled by the evolution process itself.
- All evolutionary algorithms need good tuning of their parameters to do good search.

The roles of the three operators typically used in GA and ES, including selection, mutation and crossover, are well understood (Bäck, 1994). Mutation is used to introduce innovations into the population, recombination shuffles the existing information, and selection guides the search to promising regions of the search space. Mutation and crossover are normally regarded as means for exploration, while selection is for exploitation. It is widely acknowledged that the balance between the creation of diversity and its reduction by focusing on the individuals of the currently higher fitness is critical to achieve efficient search for complex optimization problems (Bäck, 1994). As good control of exploration and exploitation is not assured in many comparison studies, many comparison results of evolutionary algorithms are not so convincing as they turn out to be. In the following section, we will examine what kinds of techniques exist that are able to improve the search capability and, especially, sustainability of evolutionary algorithms.

2.2 Previous Work on Sustainable Evolutionary Computation

Sustainable evolutionary computation has been one of the most investigated issues in all genres of evolutionary algorithms. To solve complex problems, the capability to sustain the search sufficiently long to search widely without getting trapped in local optima is critical for all EA applications. Due to the convergent nature of standard EA frameworks (Thierens, 2000), many existing EAs suffer from the premature convergence/stagnation phenomenon, which has been extensively investigated for a long time (Carter & Park, 1994; Ryan, 1996; Darwen, 1996; Mahfoud, 1995; Cantú-Paz & Goldberg, 1999; Xie, 2001; Goldberg, 2002).

Most of the proposed techniques to sustain evolution can be largely classified into two classes. One is better control of the balance between exploration and exploitation. The other is improving the evolvability with a new representation of the search space, new variation operators, and new individual initialization methods, etc. The following section will survey the techniques one can use for solving difficult search problems using evolutionary algorithms.

2.2.1 Previous Approaches to Balance Exploration and Exploitation

Compared to natural evolution, why is artificial evolution more prone to premature convergence/stagnation? One of the fundamental reasons is that we have only a limited population size, which leads to confliction between exploitation and exploration. Most of the techniques aimed at preventing premature convergence essentially deal with the following problem: given a limited population size, what types of individuals should be kept and what type of individuals should be discarded, and when and how should the exploration and exploitation operations be allocated.

Techniques in the first category try to control the balance of exploration and exploitation by using only the fitness information of individuals. The simplest emulation of natural evolution by always keeping the best individuals discovered so far – survival of the fittest– does not work because loss of diversity leads to poor exploration in new search regions. This selection scheme is called *truncation selection*, which usually requires other mechanism to weaken the selection pressure (Eshelman, 1990). The other extreme is to make the selection process independent of the fitness. This would maintain the diversity of the population and may lead to more sustainable search, but lack of selection pressure implies that the search is not productive and exploitation capability is lost. One such selection scheme named fitness uniform selection (FUSS) was proposed by Hutter (2002) to maintain the population diversity. In this approach, the selection operator in an evolutionary algorithm is uniform with respect to the fitness of individuals. Then there is no selection pressure from the selection operator. However, Hutter demonstrated that in several problems, the fact that there are fewer high-fitness individuals compared to low fitness individuals in the current population at any time implies that the high-fitness individuals tend to be selected more frequently, leading to a certain level of selection pressure to drive the fitness up. Unfortunately, this approach will suffer from this extremely low selection pressure incurred only by the fitness distribution to exploit useful information collected so far. All other approaches in this category take intermediate positions.

Controlling Population Size

One of the approaches to sustain evolution is to control the population size. For example, to improve the search capability of an evolutionary algorithm, one can simply increase the population size, which is widely adopted in genetic programming and has enabled Koza to create human-competitive results with population sizes of several millions (Koza et. al., 2003). However, large population sizes usually mean large numbers of evaluations, which is usually undesirable. The resulting low selection pressure or convergence rate means weaker capability to effectively exploit the information discovered. So large population size without care usually means inefficient use of evaluations. Another complementary approach is to use smaller population size and use multiple runs when the current epoch gets stuck in local optima. Small population size usually means better exploitation, but has much greater probability to result in premature convergence because of sampling error or failing to accumulate required building blocks (Goldberg, 1989). And this shortcoming usually cannot be overcome by running more epochs.

Experimental comparison of multiple short-runs with small population sizes and a single run with a large population size has been conducted in both genetic algorithms (Cantú-Paz & Goldberg, 2003) and genetic programming (Luke, 2001). The conclusion of Cantú-Paz & Goldberg is that for complex problems, large population size should be used, and if a running epoch does not make progress beyond a threshold number of evaluations, it is better to restart

(Luke, 2001). But the question of how to decide an appropriate population size for a real-world problem is still an issue despite intensive theoretical analysis and population sizing models (Goldberg, 1989; Goldberg et al., 1992; Harik et al., 1997; Gao, 2003). In addition to the difficulty to estimate the parameters used in their theoretical population sizing models, their assumptions themselves are also questionable and the population sizing estimation only applies for the canonical types of genetic algorithms. The HFC model proposed in Chapter 4 will show how genetic algorithms based on HFC can usefully violate the existing population sizing theory.

To combine the benefits of large population sizes and small ones, adaptive population size control has been proposed by many researchers (Tan, Lee & Khor, 2001; Wu, 2003). The basic idea is to increase the population size when the population diversity is too low and decrease it otherwise. However, varying population size doesn't change the convergent nature of canonical evolutionary algorithms. However the population size be controlled, increasing population size or changing the population sizes adaptively can only delay premature convergence and a sustainable evolutionary search is still not available.

Choosing the Correct Selection Pressure

In contrast to the approach of increasing population size, which increases the diversity of the population by providing more slots for accommodating worse individuals, the speed of convergence or the selection pressure can also be adjusted by using the right selection schemes. Bäck (1994) provided a theoretical and experimental analysis of four typical selection schemes and proved that selection pressure increases in the order proportional selection, linear ranking, tournament selection, and $(\mu \overset{+}{,} \lambda)$ selection. However, control of selection pressure has to be in accordance with genotypic diversity constantly generated by genetic operators. When only weak diversity-introducing operators as in a typical genetic algorithm, low selection pressure should be used; otherwise high selection pressure can be used, as in evolution strategies. However, lowering the selection pressure is not a good strategy to delay premature convergence, because it usually leads to slow, unacceptable exploitation efficiency.

Adapting Parameters of Mutation and Crossover

In addition to the adaptation of the population size, varying many of the other running parameters of an evolutionary algorithm can also affect the balance of the exploitation and exploration and then can make the evolutionary search more sustainable. Eiben, Hinterding and Michalewicz (1996) classified all kinds of parameter adaptation approaches into three classes: deterministic, adaptive, and self-adaptive approaches. Boltzmann selection (Mahfoud, 1997), e.g., is a deterministic adaptation selection scheme that varies the selection pressure along the course of the evolution according to a pre-defined cooling schedule, similar to the idea of simulated annealing. In genetic algorithms, the mutation and crossover probability directly influence the balance of exploration and exploitation. But how to determine an appropriate rate of these parameters is difficult. Many mechanisms for adapting operator probabilities have been proposed (Tuson & Ross, 1996; Lis and Lis, 1996). In Lis and Lis (1996), several subpopulations are used, each with a different set of parameters. After a certain period of time, the performances of all subpopulations are compared and the parameters are adapted toward those of the most successful subpopulation. A similar two level genetic algorithm named DAGA2 was reported in Wang et al. (1996).

Unfortunately, parameter adaptation can only delay the speed of loss of diversity and cannot guarantee the diversity of the population unless the diversity of the population is explicitly used as the feedback to adjust the parameters. As a result, the majority of techniques aimed at increasing the search capability are centered on the concept of population diversity, either genotypic or phenotypic.

Increasing Population Diversity by Niching

Population diversity is a central concept of natural or artificial evolution. Diversified individuals in a population ensure that there exist sufficient genotypic materials for assembling and mixing. It also enables the quick adaptability of the population to a dynamically changing environment. From the point of view of search, diversity means the capability of exploration in different regions, thus avoiding stagnation or getting trapped in local optima. It should be noted that diversity is an inherent property of natural evolution, probably induced by the diversity of the natural environment. Another perspective on diversity is that the philosophy of diversity is superficially contrary to the principle of "survival of the fittest". But actually, as the natural environment is inherently diverse, "survival of the fittest" is best interpreted as a local principle and, indeed, there is no unified definition of absolute "fittest". So diversity is a fundamental principle of sustainable natural evolution.

As there is no diversified environment in most evolutionary algorithms, explicit mechanisms to promote diversity seem to be a necessity for effective evolutionary search. Actually, pioneers of evolutionary computation have already realized the importance of diversity to achieve sustainable evolution. As early as in 1970, Pincus identified that keeping individuals with lower fitness can lead to improved capability to escape local optima, and the lack of diversity is one of the major source of stagnation at ridges on the fitness landscapes in Bremermann et al. (1966) and Bossert (1967). It is also proposed that evolutionary innovations may be more likely to happen from "mutations of mutants" than from the variations of the superior individuals (Galar, 1985). More discussion of diversity is available in (Fogel, 1998).

Given the importance of population diversity, there are basically two types of strategies to achieve diversity: One is the diversity preserving strategy—keeping specific types of individuals to promote diversity. The other is the diversity generating strategy – increase diversity by introducing new genetic material either by importing new individuals or generating random individuals. Another classification method is whether or not a technique requires the definition of a so-called distance function between two individuals. Most existing approaches require such a function, either according to genotypic distance or phenotypic distance.

The first class of diversity preserving approaches is crowding-based techniques. In the crowding approach of De Jong (1975), each offspring is compared to a randomly selected set of K (crowding factor) individuals from the original population and the most similar (according to a distance function) individual is replaced by the offspring. This crowding idea by comparison with a random set is extended by Harik (1995) to incorporate the factor of fitness during replacement. In his restricted tournament selection, each offspring competes with the most similar one of a randomly chosen set according to fitness, and the offspring replaces the closest victim only when it has higher fitness than the victim. Another type of crowding technique exploits the fact that the offspring of crossover and mutation are most similar to their parents and competition can be made more efficient by forming competition in the family. Related techniques include preselection (Cavicchio,1970), where the single offspring of a crossover replaces the inferior of its two parents; deterministic crowding (Mahfoud, 1992), where the two offspring of crossover will pair with their two parents such that the sum of distance of the two pairs are minimal and the competition occurs between each pair according to their fitnesses; and probabilistic crowding (Mengshoel & Goldberg, 1999) which extends deterministic crowding by modifying the competition rule: the individual with higher fitness does not always win; it only has higher probability to win. These family-oriented crowding techniques are easy to use and are demonstrated to be effective for many multi-modal optimization problems.

The second class of diversity preserving approaches is inspired by the biological niche concept (Holland, 1975). Goldberg and Richardson (1987) developed the first widely accepted fitness sharing technique, in which the perceived fitness of an individual by the selection operator is modified according to both the actual performance fitness and its neighborhood information. The more crowded the area the individual is located in, the more its fitness is degraded. This creates a number of artificial niches, each having a carrying capacity proportional to the general quality of the solutions in the niche. One of the drawbacks of fitness sharing is that it requires the definition of a neighborhood radius σ_{share} which is hard to estimate. The performance of fitness sharing also degrades greatly if the distribution of the optima is irregular or too close to each other. The same is true if the fitness range is not well scaled.

A third class of diversity preserving approaches works by explicit segregation of competition according to some rules. The most widely used technique is the "island" parallel model, in which individuals are allocated to a number of subpopulations. Competition occurs only among individuals within the same subpopulation. Some individuals will be exchanged among these subpopulations with a specified frequency and exchange strategy. Another way to implement segregation of competition is by defining some kind of distance metric and only allowing individuals in the range of a specified distance to compete and mate (Gorges-Schleuter, 1989; Collins and Jefferson, 1991; Davidor, 1991). Spear (1992) proposed yet another approach to segregate individuals using tags, in which individuals with a given tag belong to the same niche and compete against each other. The underlying assumption of these spatial separation approaches is that different niches will converge to different areas of the search space and thus the diversity of the whole population can be maintained. Other implementations include the diffusion model (Bäck et al., 1997), the multinational EA (Ursem,

1999) and the religion-based EA (Thomsen, 2000). The idea that different groups of individuals with different characteristics usually converge to different search areas also underlies the design of ensemble genetic algorithms composed of a set of cooperating genetic algorithms each with different characteristics. The cooperation is achieved by exchanging individuals among member GAs. One of this type is the multi-resolution genetic algorithms, the iiGA (Lin and Goodman, 1994), in which a set of simple genetic algorithms are organized in a hierarchical way according to the resolution of their genotypic representation. Another example is the multi-representation GA (Fonlupt, Preux & Robilliard, 1994), which does not require the hierarchical organization of iiGA. The Species Adaptation GA (SAGA) proposed by Harvey (1992) essentially uses the same idea. Each time when the population converges somehow, a different representation of the problem is used. Harvey argued that variable-length genotype representation and neutral network helped SAGA to fight against premature convergence. The basic idea of all these approaches is that different representations of the search space have different local optima structures and individuals trapped in one search space usually do not get trapped in the alternatively represented space and then may help the converged member GA to escape the local optima.

Diversity of the population can also be used as a feedback to guide the evolutionary search process. In the Diversity-Control-Oriented genetic algorithm (Shimodaira, 1999), a diversity measure based on Hamming distance is used to calculate the survival probability for the individuals. A low Hamming distance between the individual and the current best individual is translated into a low survival probability. The Shifting-Balance genetic algorithm (Oppacher & Wineberg, 1999) also uses Hamming distance to calculate a containment factor of two subpopulations -- one of them is selected by fitness, the other by diversity. The whole objective is to increase the distance between the two subpopulations, and thus, the diversity of the whole population. The Forking genetic algorithm (Tsutsui et al., 1997) uses the global diversity metrics to determine when to separate a set of individuals out as an independent subpopulation to increase the diversity. De Jong et al. (2001) used the population diversity as an explicit optimization criterion and a multi-objective evolutionary algorithm is used to solve the original single-objective problem. Ursem (2002) proposed another approach to use the global diversity metric to determine a switch between the exploration stage, during which mutation operators are applied and the exploitation stage, during which selection and crossover are applied.

Increasing Population Diversity by Restarting and Rejuvenating

Another generic approach to achieve population diversity is to actively generate diversity. One simple approach is just to increase the mutation rate. While using high mutation rate can definitely increase the population diversity, it is at the cost of effective exploitation. In the later stages of search, a high mutation rate tends to disrupt existing individuals that have well-coupled components. The resulting individuals usually have very low fitness and cannot survive at all. It is clear that diversity incurred by high mutation rate does not imply effective search. Similar reasoning also applies to the partial-reinitialization approach, in which a portion of the converged population is replaced with randomly generated individuals. The multi-restart approach is even worse. In this approach, when the population shows the signal of getting trapped in local optima, an entire new epoch is started and the population is filled with new random individuals (Fukunaga, 1997). This will waste all useful information collected in the previous epochs. The CHC algorithm (Eshelman, 1990) employs the so-called Cataclysmic mutation to pursue sustainable search. When the population gets converged, a highly disruptive mutation is applied to the best individuals for multiple times to reinitialize the population by mutating some percentage of its bits (e.g. 35%). This approach is equivalent to the reinitialization approach when the uniform crossover is used.

2.2.2 Previous Approaches to Sustainable Evolution by Improving Evolvability

Evolvability as defined by Altenberg (1994) refers to "the ability of a population to produce variants fitter than any yet existing." The evolvability of an evolutionary system is largely determined by the representation of the search space (including the genotype-phenotype mapping if available) and the search operators. The joint effect can be represented as a parent-offspring transmission function, which maps parent points to a probability distribution from which their offspring are sampled. (Altenberg, 1994).

The sustainability of evolutionary search for a given problem depends strongly on the representation of its search space and related operators. One distinguishing example is the evolution of programs. The approach of Friedberg (1958) fails because the search space has low inheritability of the program space and thus low evolvability. The genetic programming approach of Koza (1992) succeeds because the syntactically valid tree representation provides sufficient heritability to the programs. Another example of improving the search is Gray coding in genetic algorithms for numeric optimization; it is proved that Gray coding produces many fewer local optima for the binary representation of numeric variables than the simple binary representation (Whitley, 1989). There are many other techniques aimed at improving evolvability by inventing new representations and new variation operators, but they are usually problem dependent.

One of the explanations of premature stagnation from the evolvability point of view is that when the fitness of individuals gets higher and higher, the variation of phenotypes becomes more and more costly. Selection then tends to favor individuals that increasingly conserve their fitness and exhibit less and less phenotypic variation (Glickman & Sycara, 2000).

Several approaches have been proposed to prolong evolutionary exploration. One approach is to increase the selection pressure, but that typically occurs at the cost of loss of diversity. Altenberg (1994) proposed a soft brood selection approach to counteract the tendency towards increasingly conservative individuals. In this approach, the number of offspring produced by parents is magnified and the competition occurs first among offspring of the same parents and those winners then compete at the full population level. Another approach is the group selection exemplified by nested evolution strategies (Herdy, 1992), in which a number of isolated subpopulations compete first and then competition occurs among individuals of the same subpopulation. Glickman and Sycara (2000) proposed that dynamic fitness, as is typical in nature, promotes the evolution of evolvability. This is similar to the idea of SAGA (Harvey, 1992).

While evolving a solution to a complex problem is hard, evolving a solution to its sub-problems is much easier. One approach derived from this divide-and-conquer strategy toward sustainable evolutionary search is conducted in the context of incremental evolution, where a simplified version (or sub-problems) of a problem is subjected to evolution, the results of which are used as starting point for a more complex version of the problem. This kind of incremental evolution with variable genotypes, when coupled with speciation, has achieved significant results in evolving neural networks (Stanley et al., 2002). A similar, but more complex, evolutionary approach has also been used to improve scalability for evolvable hardware, evolving increasing levels of user-defined subsystems (Torresen, 2002) with a simple divide-and-conquer approach. There are also some manual hierarchical divide-and-conquer strategies with multiple subpopulations (Lin and Goodman, 1994; Aickelin et al., 2001; Hsu et al., 2002; Torresen, 2002; Eby et al., 1999).

Another important line of study on sustainable EA design is centered around the concept of building block supply, growth and mixing (Goldberg, 2002). Messy GA and other competent GAs (Goldberg, 2002) are characterized by the *identification* and recombination of building blocks, and have mainly been applied to binary-encoded GAs. The pervasiveness of building blocks (or subcomponents) in the physical and biological world suggests strongly that competent EAs for complex real-world problems will rely heavily on the capability to maintain population diversity, to continue exploration, and to exploit extant building blocks (Holland, 2000). The success of competent GAs (Goldberg, 2002) for difficult problems confirmed this, but is only the first step toward exploiting the building block concept. However, the concept of good, co-adapted genetic material as used here does not require that it be tightly linked, as in the classical building block definition as a *short*, low-order, high-fitness schema. Good genetic material may also be of higher order, and loci need not be adjacent on the chromosome to constitute good genetic material for further evolution, as is amply demonstrated in evolution strategies work.

An important observation of the physical and biological world is that there exist successive levels of building blocks – from nuclei to atoms to molecules to polymers; from organelles to cells to tissues to organs to organisms to ecosystems. The maintenance and evolution of all levels of hierarchical subsystems or building blocks is essential to the efficiency of nature in forming complex systems. Simon (1973) showed that hierarchical systems evolve much more rapidly from elementary constituents than will non-hierarchical systems containing the same numbers of elements. This kind of hierarchical strategy for problem solving has been explored first in genetic programming with ADFs (Koza, 1994), module acquisition (Angeline et al., 1994) and Adaptive Representation (AR) (Rosca, 1994), and later in a genetic algorithm, hBOA, the hierarchical extension of BOA (Pelikan & Goldberg, 2001). These approaches are all somehow representation specific and rely on the explicit identification of building blocks in the forms of ADFs, modules, or merged variables. A similar hierarchical strategy is also used in co-evolution for subcomponent discovery and assembly (Potter & De Jong, 2000). It appears to the author that the fundamental search capability of evolution depends on the capability to discover and exploit increasing levels of modules or modularity of the search space.

Another important theory to promote the evolvability and thus the sustainability of evolutionary search is neutrality theory. This theory suggests that most mutations in biological evolution do not cause a phenotypic change because the mapping from genotype to phenotype is redundant. This many-to-one redundant mapping results in neutral networks: sets of genotypes connected by single point mutations that map to the same phenotype. The neutral network then allows the population to drift along these networks without degradation of fitness, but thereby to wander around in the genotype space. This will greatly increase the chance of finding phenotypes of higher fitness and reduce the risk of getting trapped in local optima (Huynen, 1996; Huynen, Stadler & Fontana, 1996). However, not just any type of redundancy is helpful for sustainable evolution. For redundancy to be of use, it must allow for mutations that do not change the current phenotype, thus maintaining fitness, and allow for moves to areas of genotype space where new phenotypes are accessible (Shackleton et al., 2000).

2.2.3 A Common Root of Tragedies: the Enigma of Premature Stagnation

Sustainable evolutionary search has been attracting a large amount of attention and effort in evolutionary computation. In summary, the major obstacles that prevent us from achieving sustainable evolution include the following:

a) Lack of diversity: some essential building blocks are missing

b) Founder effect (Holland, 2000): early-discovered building blocks interfere with finding and nurturing of other competing building blocks

c) Conservative lock-in: high-fitness individuals tend to allow less phenotypic variation without destroying their fitness for survival

d) Lack of evolvability: the search space is highly rugged and there exists little neutrality to allow effective genotypic space exploration by keeping sufficient fitness to survive under the competition

The dominating research on sustaining evolutionary search focuses on increasing or maintaining population diversity. Other techniques including the building-block-based competent GAs and techniques derived from the evolvability or neutrality theory, which can improve efficiency or delay premature convergence or stagnation. But existing techniques have only achieved limited success to sustain evolutionary search. Many of the difficulties outlined above arise from the assumptions and limitation of the underlying framework of all these techniques: the fundamental convergent evolutionary computation framework.

The most distinguishing feature of the convergent evolutionary computation framework is that the average absolute fitness of the population constantly increases until stagnation, and only individuals with competitive fitness can survive. This is why neutrality of search space can improve the search. It simply allows continuing search even when the population reaches the high-fitness stage. It is also the cause of the conservative lock-in effect, since high-fitness is the requirement for survival, and finding even higher fitness phenotypes at a high fitness stage of evolution is difficult. This "survival with high fitness" requirement also explains why the "founder effect" is hard for existing techniques to overcome. At the high-level fitness stage of the population, there is no opportunity to allow extensive exploration of new building blocks from lower levels to higher levels. The probability of discovering new building blocks by perturbing existing high-fitness individuals and maintaining the high-fitness is extremely small, if not zero. Another fundamental trap for existing techniques seeking to overcome premature convergence is the concept of escaping local optima, as derived from numerical optimization. As escaping from local optima almost always requires crossing degraded fitness areas to find better fitness phenotypes, it just cannot allow sufficient time to explore new building blocks or search areas, which usually need to be "cultured" up from the primitive levels.

So the necessary condition of sustainable evolution is that even at the high-fitness stage of the population, the algorithm must still allow sufficient time to explore the search space starting from a low fitness level and discovering new building blocks. This cannot be achieved by perturbation of existing high-fitness individuals in the competitive high-fitness population.

Let us examine whether existing techniques can overcome the four typical difficulties listed above:

Increasing population size or controlling the selection pressure cannot overcome this difficulty. Since the average fitness of the population (and so the level of competition) increases in all these techniques, the survival probability of the primitive individuals is not high enough to allow good exploration.

Diversity maintaining techniques can partially overcome this difficulty, assuming that the low-fitness individuals have quite different genotypes or phenotypes. However, if the individuals have very low fitness compared to the current average fitness level, they still cannot survive. For example, in fitness sharing, the number of individuals accommodated by a niche is proportional to the fitness level of that niche. So these primitive individuals can’t form stable niches to explore. In deterministic crowding, the competition among the family is still based on fitness and those primitive individuals cannot survive. Essentially, existing niching techniques only implement single-level niches, while the real niches in natural environment are vertical, and vertical speciation is critical in evolution (Chowdhury and Stauffer, 2003). Similarly, other diversity maintaining techniques will also fail to effectively overcome the "founder effect." Actually, diversity preserving techniques based on distance functions usually can only maintain the diversity of competitive high-fitness individuals discovered at a similar pace. It cannot maintain the diversity of high-fitness individuals while allowing exploration in new, low-fitness search areas. Moreover, it cannot distinguish two types of low-fitness individuals -- individuals degraded from high-fitness individuals and primitive individuals coming up from lower levels.

Current diversity generating techniques such as high-mutation rate, restarting and rejuvenating also cannot overcome the "founder effect." They can perturb existing high-fitness individuals, but they cannot allow sufficient time for low-fitness individuals from brand new search areas to explore and discover new building blocks.

It is clear that the "survival of the fittest" principle of natural selection is a local concept in natural evolution, and that the whole story of natural evolution includes many local competition niches exerting many different types and levels of selection pressure, reflecting the diversity of the natural environment. Evolution is going on simultaneously at all levels. Low-level evolution is always working to find good building blocks to be exploited by higher-level organisms. This essentially removes most of the obstacles for sustainable evolution and becomes the basis for sustainable evolutionary algorithms based on the hierarchical fair competition principle proposed in Chapter 4.

2.3 Overview of Topologically Open-Ended Evolutionary Synthesis

2.3.1 Topologically Open-Ended Synthesis: a Common Framework

Topologically open-ended synthesis is concerned with the task of selecting an appropriate set of components (either homogeneous or heterogeneous), connecting them into a whole, and adjusting the properties of the components and connections to implement a desired function (Figure 2.4). Essentially, most innovative design activities, especially the creative aspects of engineering design, are centered around topologically open-ended synthesis of systems, such as analog and digital circuits, control systems, telecommunication networks, transmission mechanisms, neural networks, graphs, transportation networks, mechatronic systems, the morphology and control systems of robots, evolvable hardware, etc. The connection of components can be either physical, such as the wire connections in analog circuits, or abstract, like the information flow in a control system. While there are tree-like or linear structures, a generic structure can be represented by a network composed of a set of components.

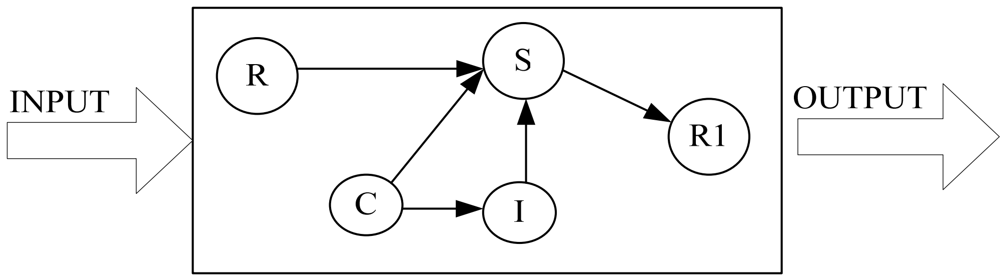


Figure 2.4 A system for design synthesis. Given a set of different types of building blocks, how can one select a set of elements and assemble them into a whole and adjust the properties of elements and connections to achieve a specified function?

Topologically open-ended synthesis can be divided into topology synthesis and parameter synthesis problems. These two are usually not separable, as the goodness or performance of a particular system structure also depends on the parameters of its subcomponents and their connection properties. However, in traditional engineering design, since the quality of the topology of systems is evaluated based on design knowledge about the logic of the topology, parameter selection is often regarded as an unimportant secondary problem.

Evolutionary synthesis is a class of techniques inspired by natural evolution, that may automatically evolve a complex system by selecting an appropriate set of building blocks or subcomponents, assembling them into a whole, and adjusting the properties of its components and connection properties to achieve a desired function. Evolutionary synthesis can also be used to solve some reduced problems such as system parameter optimization or system architecture synthesis.

Evolutionary algorithms emulate the natural evolutionary process by employing a population of individuals and updating it continually using variation operators such as crossover or mutation, coupled with fitness-based selection. A prominent feature of evolutionary search is that, given any representation scheme of the potential solutions and corresponding variation operators, the system can often come up with some unusual and interesting results. EAs can be used for both topology synthesis and parameter optimization. This unconstrained search capability compared to other traditional optimization or search algorithms make them very suitable for evolutionary synthesis of systems, where the topology search is critical.

With their widely applicable search capability, evolutionary algorithms are applied in many system synthesis application areas. Examples include, but are not limited to, analog and digital circuit design (Koza, 1994), metabolic path synthesis (Koza, 1999), control system design (Koza, 1999), telecommunication network design (Aiyarak et al., 1997), transmission mechanism synthesis (Kunjur, 1995; Fang, 1994), neural network synthesis (Yao, 1996), truss structure synthesis (Deb & Gulati, 2001), graph synthesis (Luke & Speacor, 1996), transportation network synthesis (Vitetta, 1997), mechatronic system synthesis (Seo et al., 2002), morphology and control system synthesis in robotics (Eggenberger, 1996; Funes & Pollack, 1998), evolvable hardware (Yao & Higuchi, 1996), etc.

Human designers depend on rich domain expertise accumulated for a long time to solve synthesis problems. Enumeration, heuristic guided enumeration, and knowledge-based approaches are used to help the design process. It is well known that training a qualified analog circuit designer requires a long time. In addition, human designers are strongly constrained by their limited domain knowledge and incomplete understanding of the physical process itself. In many under-exploited design domains such as adaptive systems, fault-tolerant systems, and poorly specified and unstructured systems, human designers often show their incapacity to handle that kind of flexibility and complexity and show that they lack the imagination to provide innovative solutions. With this background, the automatic synthesis techniques enabled by the unstructured search capability of evolutionary algorithms becomes extremely promising.

Basically, topologically open-ended evolutionary synthesis techniques can be classified into two categories: direct encoding techniques, including variable length genetic algorithms and genetic programming, and indirect encoding techniques, including developmental genetic programming, and artificial embryology.

2.3.2 Direct Encoding Techniques

The first category of evolutionary synthesis techniques is the direct encoding approaches using variable length genetic algorithms, genetic programming or evolutionary programming. In these approaches, each connection in the structure is represented explicitly in the genotype. Direct encoding schemes include linear encoding (Venkatasubramanian et al., 1994), tree encoding (Nachbar, 1998), graph encoding (Globus et al., 1999) or the hybrid (Pujol & Poli, 1998) representation. Matrices are widely used to encode topology (Tay, et al., 1998). Most evolutionary synthesis of neural network topology focuses on direct encoding. A comprehensive survey is provided in Yao (1999). A major difficulty of direct encoding is its scalability. A large structure will require a huge genome to specify its structure, and discovery and reuse of modules are difficult to implement and are usually not available. Another problem is the permutation (competing convention) problem, which says that two functionally equivalent but genetically different individuals will make crossover very inefficient. In the rest of this section, some major direct encoding schemes for evolutionary structure synthesis are reviewed.

Variable length genetic algorithms

Venkatasubramanian et al. (1994) have applied a genetic algorithm to a computer-aided molecular design problem. The structure of a molecule is represented by a string of symbols, encoding nested lists in Lisp. The string may be composed of one or more segments. These segments serve as the chemical building blocks called base groups. Such a symbolic representation is claimed to facilitate the incorporation of higher-level chemical knowledge more readily. The genetic operators for this symbolic string includes one-point crossover, two-point crossover, a blending operator, main-chain and side-chain mutation, insertion and deletion operators, and a hop operator, by which a randomly selected group in the main-chain of the molecule 'hops' to another randomly selected location in the main chain, causing a local search. It is apparent that this kind of string representation lacks the capability of module reuse, which is essential to the scalability of evolutionary synthesis. Its building blocks are more or less constrained to two-port components, and then it is effective mainly for finding chain-like structures of the building blocks. Another interesting approach to evolve neural networks, called NEAT, was proposed by Stanley and Miikkulainen (2002). The major innovation is the introduction of a historical marking technique which labels each gene with a unique number. These labels can be used to align genes of two chromosomes encoding two neural networks, thus giving a natural definition of distance in a space of complex topologies. This enables NEAT to apply fitness sharing to ensure topological innovation.

Genetic programming

A direct encoding technique called Cartesian genetic programming has also been used successfully to evolve digital circuits (Miller, 2000). In this approach, the genotype is represented by a list of integers, rather than trees as in standard GP that are mapped to directed graphs. Another straightforward encoding scheme to evolve graphs is to represent structure with (directed or undirected) graphs. Poli (1997) and Teller (1995) have studied this kind of direct graph encoding schemes. PDGP (Poli, 1997) is an extension of standard GP that replaces the tree-like representation with generic graphs. PDGP uses the 2D grid to position the nodes, similar to Miller's (2000) encoding for digital circuit design. Globus et al. (1999) proposed a cyclic graph representation with a clever graph crossover operator (Genetic Graph) to evolve molecules for pharmaceutical drugs. The crossover operator of Genetic Graph is more generic than that of PDGP, since it can use multiple crossover points to divide the system into two segments. In the case of crossover with different numbers of crossover points, it can create new edges to satisfy the excess crossover points on one fragment. The experiments show that using this direct encoding approach, it is hard to find even moderate-sized molecules or any non-trivial circuits. Another major deficiency of this generic graph crossover is that it usually does not preserve useful sub-graphs, so the crossover is extremely destructive. In this sense, the ideal crossover operator should exchange functional modules rather than syntactic sub-structures. These insights apply to almost all kinds of other direct encoding techniques. Molecule design has also been studied with GP (Nachbar, 1998). Rather than using the developmental process, like analog circuit design in (Koza, 1994), this approach uses trees to directly represent the molecule structure. Each tree node represents an atom, with the bonds existing between parent and child nodes. This same technique is also used by Koza to evolve controllers (Koza, 2003).

2.3.3 Indirect Encoding/Generative Encoding Techniques

In generative encoding of structures, the genotype does not represent the structure explicitly. Rather, it contains all the information that a construction process needs to generate the structure by a developmental phase. The first proposal of generative encoding for evolving of topology is from Kitano (1990), where context-free grammars are used to grow networks represented by network matrices. By optimizing grammar rules that generate network topology instead of directly optimizing networks, this research aims to achieve scalability by possibly reusing some sub-network-generating grammar rules. This would allow designers to first evolve some grammars for smaller problems and then use these rules as building blocks to attack larger problems (Gruau & Whitley, 1993). Generative encoding schemes are shown to have several significant advantages, such as the possible complexity of the evolved design and the speed of evolution (Hornby et al., 2001).

Developmental genetic programming

Cellular encoding (Gurau, 1992) is a technique that encodes a structure with a grammar tree of structure-transforming node operators (encoding tree) and evolves those trees using GP. Each node of the GP tree is a node operator, which accepts from its parent a specific node of the current structure and can then modify the structure around that node by creating new nodes or new edges, changing existing connections, or deleting old ones. This developmental process is executed by traversing the GP tree in a breadth-first, left-to-right manner and running the node operations of each GP tree node on the embryo graph. The weakness of cellular encoding lies in the context-sensitive nature of the GP tree encoding. A sub-tree, when moved to another place of the GP tree by crossover, can generate a dramatically different substructure. This is a common weakness of all tree-like encoding schemes, low heritability. The second shortcoming of cellular encoding is the lack of a sufficiently rich set of edge-modifying operators, often leading to quick multiplication of edges (Luke et al., 1996), which leads to a bias for generating highly connected topologies, as a result of its particular node-operating operators.

Edge encoding is introduced by Luke and Spector (1996), aiming to allow more flexible control of the edge growth in the topology evolution. Edge encoding also uses a tree-structured chromosome which can develop into a directed graph. Each node of the chromosome tree is an operator that can act on the edges of a graph. An edge operator can accept from its parent node a single edge in the graph, sometimes with additional data such as a stack of nodes. Edge encoding operators are executed in a pre-ordered way, modifying a graph edge and passing that edge (and any new edges) to its children for further modification. The starting point of the development can be a graph with a single edge or an embryo with many edges, but only one of them is fed to the root node of the chromosome tree as the starting point for development. Edge encoding is more suitable for evolving graphs with low connectivity, while cellular encoding is better for dense connectivity.

It is straightforward to combine cellular encoding (Gurau, 1992) with edge encoding for more flexible control of graph evolution. The developmental GP (Koza, 1994) for analog circuit synthesis uses such a hybrid version of cellular encoding and edge encoding. De Jong and Pollack (2001) also studied the hybrid encoding technique to demonstrate the utility of capturing bias from related small problems for solving larger problems.

Variable length linear encoding for circuit design is investigated in (Lohn & Colombano, 1999). In this approach, the genotype is composed of a linear sequence of circuit-constructing primitives. Each primitive contains a component-placing instruction (control instructions are not included, but may be very important) along with the genes for establishing the parameter values.

2.3.4 Artificial Embryology

Artificial embryology approaches to evolutionary topology synthesis refer to a class of techniques inspired by the developmental process of multi-cellular living organisms. The development process maps the static genotype description of an organism into a dynamic instance of an organism, the phenotype. The developmental concept has been intensively studied in evolutionary system synthesis, including robotics morphology, evolvable hardware (Gordon & Bentley, 2002) and circuit synthesis (Koza, 1994). Advantages of the developmental approach include compact genotype encoding for complex phenotypes, substantial phenotypic variation with simple genotypic mutation, allow both small and large-scale mutations, incremental building, and easy specification of modular design (Dellaert, 1995). Developmental models in EAs can be divided into three categories (Bentley et al., 2001):

- External developmental process

  The developmental process is simulated by a hand-designed fixed mechanism that is external to all genotypes. This mechanism is not evolvable itself. Its only function is to map static genotype information into a phenotype. Examples include many evolutionary art systems and Dawkins' Blind Watchmaker program (Dawkins, 1987).

- Explicit developmental process

  This type of model specifies each step of the developmental process in the genome in the form of growth instructions. Koza (1994) uses a tree-like genome that encodes all the instructions to develop a circuit embryo by continually operating on the developing embryo. Most other GP variants belong to this category.

- Implicit developmental process

  In these developmental models, instead of specifying each step in the developmental process, the growth process is implicitly specified by a set of genetic rules or instructions. Jakobi (1996) used such a system that models cell division, cell movement, etc., to evolve neural network robot control architectures. Lindenmayer systems (L-systems) have also been extensively used for a number of evolutionary design problems (Haddow & Tufte, 2001; Homby & Pollack, 2001; Boers & Kuiper, 1992; Kitano, 1997).

While most developmental models simply map the genotype into phenotype, a more biologically defensible developmental model is proposed by Dellaert & Beer (1994). In this work, the developmental model is divided into three levels:

- The molecular level, which can be represented by a genetic regulatory network
- The cellular level, which models how a cell behaves and the way the genome can influence its behavior.
- The multicellular organism level, which models the interaction of group of cells including intracellular communication and response to external signals.

Another genetic programming approach that explicitly models cell interactions is biomimetic genetic programming (Lones & Tyrrell, 2002). This enzyme GP emulates the principle of biological enzyme systems. In an enzyme system, the genotype defines only a collection of functional components without explicit storing of the connection topology. During the expression process, all components interact with each other to form complete computational structures. These interactions are based the chemical preferences of each component, independent of the position of components. Some components may not be expressed at all, depending on the existence of other components.

The key idea of enzyme GP is that the structure of a program is not given explicitly, but is rather derived from the connection (interaction) choice made by each component of the program. There are several important concepts in enzyme GP to evolutionary system synthesis: components, interface, specificity, and development.

2.4 Previous Work To Improve Scalability of Evolutionary Synthesis

Research on scalability of topologically open-ended evolutionary synthesis focuses on two aspects. One is the scalability problem of the evolutionary algorithms. Actually, due to the convergent nature of the standard EA framework (Thierens, 2000), many existing EAs suffer from the premature convergence phenomenon, which has been extensively studied for a long time (Carter & Park, 1994; Ryan, 1996; Darwen, 1996; Mahfoud, 1995; Cantupaz & Goldberg, 1999; Xie, 2001; Goldberg, 2002).

The second problem that limits the scalability is the scalable representation and operator methods for evolutionary synthesis and/or the evolvability of the evolution system. The scalability issues in many synthesis problems have been studied. Torreson (2000) proposed an increased complexity scheme to increase the scalability of evolvable hardware, in which the subsystems are first evolved, then the evolved subsystems are used as building blocks for higher-level evolution. The same problem is also discussed in (Yao & Higuchi, 1998). The scalability problem of digital circuit synthesis is studied in (Vassilev, 2000). In addition, it is shown that larger building blocks (Murakawa, 1996) can provide some kind of scalability, though by removing some structure space. Cooperative co-evolution (Potter & De jong, 2000) and speciation (Darwen, 1996) are also used to facilitate the discovery of co-adapted subcomponents to achieve automatic problem decomposition. Automatically defined functions (Koza, 1994) and adaptive representation (Rosca & Ballard, 1994) are similar modular techniques in genetic programming that aim to take advantage of discovery of different levels of building blocks. The manual divide-and-conquer strategy in evolutionary synthesis is also an important approach (Torresen, 1998, 2000; de Garis, 1990; Husbands & Mill, 1991). The biological developmental model is regarded as a most promising technique to improve the scalability of structure synthesis (Gordon & Bentley, 2002; Hornby & Pollack, 2001). A survey of artificial embryology approaches to evolutionary synthesis is provided by Stanley and Miikkulainen (2003).

All these approaches differ mainly in structure encoding schemes and the variation operators. The choice of structure encoding usually has drastic effect on the search efficiency, as it may introduce a strong bias into the search space. Generally, a good structure-encoding scheme should meet the following requirements:

a) The encoding should allow any structure (of interest) to be represented.

b) The structure-constructing (genotype-phenotype mapping) process should be fast. Traversal of a graph structure to ensure validity, which is usually computationally expensive, should be limited.

c) The encoding should ensure syntactic closure. That is, genetic operators should not create invalid circuit graphs from the valid parents, or an efficient repair mechanism should be available.

d) The encoding should be able to handle multi-port components with multiple inputs or outputs. Although in some cases only two-port components are needed, this multi-port capability is needed to allow the emergence of higher-level modules.

e) The encoding allows the exploitation of some kind of modularity. Dense connectivity in the module and sparse connectivity between modules is a general principle in many systems. The modularity feature of an encoding scheme is essential to its scalability for large problems.

f) The encoding allows the exploitation of some kind of block replication and parametric modules (productive rule).

g) The encoding allows using constraints to reduce the search space – e.g., only two compatible terminals can be connected; for example, a current output signal port should not be linked to a voltage input signal port. (In some contexts, such constraints are sometimes called "strong typing.")

2.5 Summary

This chapter examines the historical origins of evolutionary algorithms and provides an analysis of the features of each genre of evolutionary algorithm, aimed at contributing to the understanding required for development of sustainable evolutionary algorithms. A comprehensive survey of existing techniques aimed to improve the sustainability and scalability of evolutionary search illustrates that much of the limitation of these techniques arises from the underlying canonical framework for evolution, namely, the single-level emulation of the "survival of the fittest" of the natural evolution process. This hints at a new sustainable evolutionary model that simulates natural evolution with multiple levels and hierarchical niches.

The second half of this chapter provides a unified framework for topologically open-ended evolutionary synthesis. Existing techniques have been classified into two categories, including the direct encoding approaches and the generative encoding approaches. Many of these approaches are examined, with typical examples. Previous work aiming to improve the scalability of evolutionary synthesis is surveyed. It shows that evolvability in topology search is critical to scalable evolutionary search. While emulating the biological developmental process of multi-cellular organisms has the most promise to achieve scalability, the anticipated success of this approach has not yet been realized.

In the following chapter, we provide a framework for topologically open-ended synthesis of dynamic systems by bond graphs and genetic programming, which will establish a series of benchmark problems to be used for testing the sustainable evolutionary computation model and related algorithms proposed in Chapter 4.

C h a p t e r 3

TOPOLOGICALLY OPEN-ENDED SYNTHESIS OF DYNAMIC SYSTEMS BY GENETIC PROGRAMMING

Synthesis of dynamic systems is a major activity in engineering design. These physical systems usually cover several domains, such as hydraulic, acoustical, thermodynamic, mechanical, electrical, and electronic engineering systems. Designing such multi-domain systems is usually difficult and tedious for human designers. Especially, the very limited topological exploration capability of design engineers constrains their potential capability to design more complex, robust, and efficient dynamic systems. Topologically open-ended synthesis by genetic programming provides a generic, promising methodology to achieve improved design automation. This chapter will reexamine the advantages and disadvantages of several graphical modeling approaches in the context of automated synthesis of dynamic systems. The bond-graph-based representation and its evolutionary synthesis by genetic programming are then introduced. Finally, two benchmark problems for topologically open-ended synthesis are explained, which are used later on to evaluate different algorithms developed in this dissertation.

3.1 Brief History of Dynamic System Design Synthesis

Traditionally, the process of design of dynamic systems starts with an abstract functional design by design engineers, based on their expertise and understanding in a given domain. Then the mathematical model of the system is derived manually, followed by detailed analysis and simulation. If the designer is not satisfied with the performance, the current design solutions are modified and the whole procedure is repeated.

With the understanding that the major design decisions are centered on the functional design or topological design of the system, all kinds of graphical modeling tools have been introduced or applied to aid the conceptual design process, such as electric circuit diagrams in electrical engineering, block diagrams in control engineering (Ogata, 1997), signal flow graphs (Mason, 1953; Mason & Zimmermann, 1960), linear graphs (Trent, 1955) and bond graphs (Paynter, 1961; Karnopp, Rosenberg & Margolis, 2000). Initially, these graphical modeling methods were created to facilitate conceptual and mathematical derivation and analysis by human designers. More recently, with the wide use of computers, interactive graphical modeling tools appeared that allow designers to create a graphical model intuitively. The system then generates the mathematical model and conducts specified simulations to analyze the system performance. Most current computer-aided design (CAD) tools for dynamic system synthesis belong to this type. It is clear that the capability for topological innovation and exploration of dynamic systems remains as a challenge to the design community.

With ever-increasing computing resource available, driven by "Moore's law" and large-scale simulation platforms, a lot of effort has been put into seeking automated synthesis approaches for dynamic systems. Automated topological synthesis has been pursued in two different directions. One is toward the knowledge-based top-down approaches, which rely on the understanding of a problem domain by domain experts and knowledge engineers. In these approaches high-level specifications of functional requirements are transformed into lower-level tasks based on a set of heuristic rules (Harjani et al., 1989). In some sense, these approaches are not truly the topologically open-ended synthesis that we are interested here.

This kind of topology synthesis has been investigated using various search-based generate-and-test approaches, similar to the "blind watchmaker"-style "tinkering" design principle employed in the natural evolution process (Dawkins, 1986). Many successful stories have been reported in evolutionary-computation-based topology synthesis of dynamic systems. Tay, Flowers and Barrus (1998) proposed a genetic-algorithm-based approach to vary bond graph models. This approach adopted a variational design method, which means they made a complete bond graph model first, and then changed the bond graph topologically using a GA, yielding new design alternatives. But their approach could only search in a rather limited topological search space. Using genetic programming, Koza et al. reported "industrial strength" synthesis results of analog circuits and controllers (Koza et al., 1997; 2000). As dynamical systems can be represented and thus evolved in many different graphical models, understanding how representation affects evolutionary topology synthesis thus becomes very important.

3.2 The Representation Issue in Automated Dynamic System Synthesis

To explore the design space of dynamic systems, either by human or by computers, some kind of representation scheme is needed. Despite the fact that all dynamical systems can be reduced to a set of mathematical models such as differential equations, design spaces are often represented by graphical models such as electric circuit diagrams, block diagrams, linear graphs, bond graphs, or signal flow graphs. These graphical modeling methods originated from different domains, largely aiming to facilitate analysis and synthesis by human designers. It is not surprising that in automated synthesis by computers, it is also much more convenient and efficient to encode and manipulate the design structures in graphical forms rather than directly manipulate differential equations. However, the characteristics of graphical representation schemes need to be reexamined and compared in the context of automated synthesis and/or topology space exploration by computers.

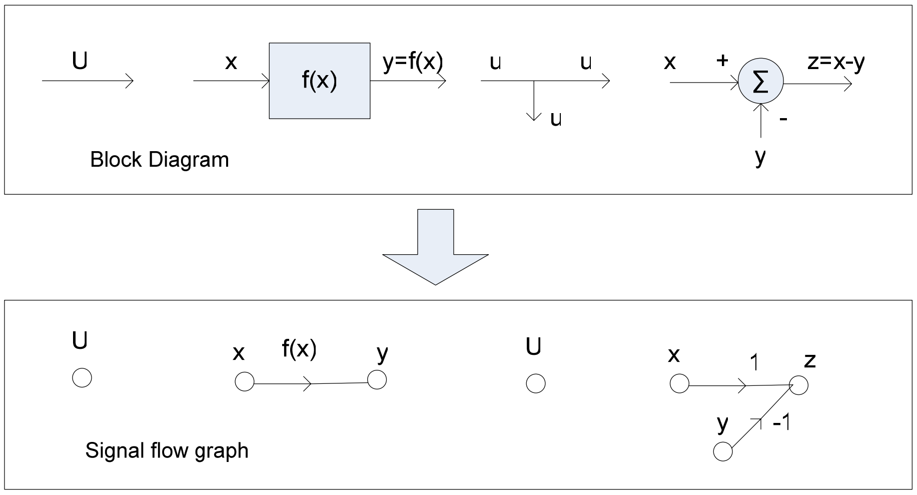


Figure 3.1 Elements of blocks diagrams versus signal flow graphs

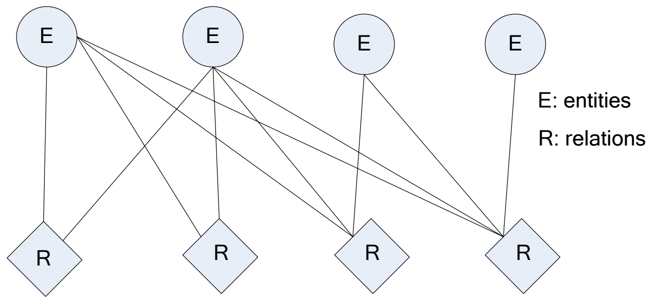
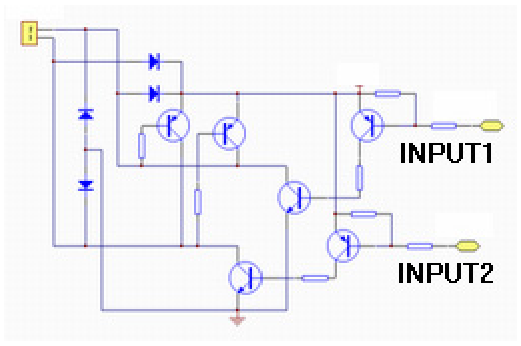
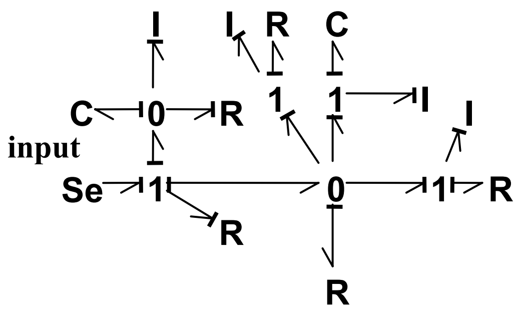


Figure 3.2 Relational graph (Squares represent relations; circles represent entities or terms)

Figure 3.1 shows the elements of dual representation schemes: block diagrams and signal flow graphs. Signal flow graphs use nodes to represent signals and edges for processes, while block diagrams make the opposite assignment. Since each node in a signal flow graph represents one signal, there is no signal flow graph equivalent for a multi-port block diagram box. So the representation capability of signal flow graphs is less powerful than that of block diagrams (Cellier, 1991). It is interesting that these two different types of graphical modeling can be unified by a generic graphical modeling approach: the relational bigraph invented by Polya (1962), as pointed out in Paynter (1992). In a relational bigraph, entities and relations between entities are both represented by nodes, but of different types. Figure 3.2 shows one example relational graph of four entities with four relations. In such cases, block diagrams use nodes to represent relations of signals that are labeled as edges, while signal flow graphs do the opposite. It is impressive that using a relational graph, Paynter was able to derive all commonly used system diagrams, such as block diagrams, signal flow graphs, circuit diagrams, and bond graphs from a single universal graph (Paynter, 1970).

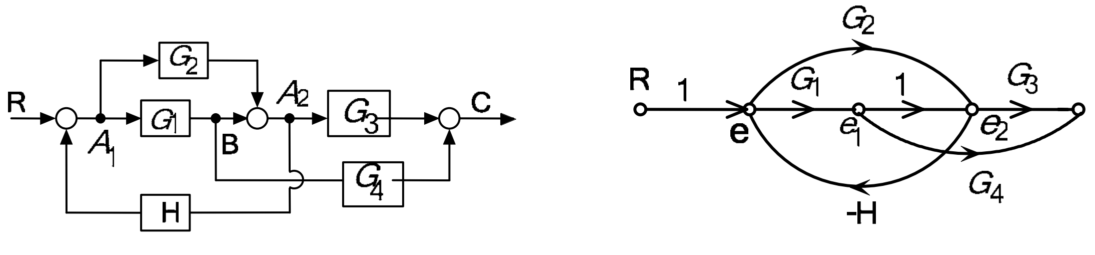




An analog circuit A bond graph

Figure 3.3 Analog circuit and Bond graph representation of dynamic systems

Graphical diagrams of dynamic systems can also be classified according to the topological space they represent. Basically, there are three types of graphical modeling. One type represents the topology of systems by their physical interaction/connection topologies of physical entities. Representative examples of this category include circuit diagrams (Figure 3.3). Chemical structure graphs also belong to this category. Computational structures in this type of graphical models are implicit. The second type of graphical modeling techniques instead aims to represent the computational structure of a dynamic system. Typical examples are block diagrams and signal flow graphs (Cellier, 1991) (Figure 3.4). The third type of graphical modeling framework tries to carry the advantage of both, preserving the topological structure shown in a circuit diagram and the computational structure shown in a block diagram. A representative of this type is the bond graph that we will explore in next section. Bond graphs have borrowed some ideas from relational graphs in terms of representing the relations among entities with explicit nodes: the 1-junction nodes for representing series relationships (or common flow relationship) and 0-junction nodes for parallel relationships (or common effort relationships).



a) A block diagram b) A flow graph

Figure 3.4 Graphical models representing computational structures

Given that there are different kinds of graphical modeling schemes, how should one decide which is most appropriate for automated topology synthesis in a specific domain? One simple answer is to use the most popular one in the problem domain, e.g., choose block diagrams for controller design, choose circuit diagrams for circuit design, and choose bond graphs for mechatronic system synthesis. But here we are interested in the differences among these modeling schemes in terms of evolutionary topology search and the criteria for choosing a modeling scheme when several graphical modeling approaches are available. The following factors need to be considered in deciding which one to use:

- Characteristics of the problem domain: physical topology or computational topology

  The distinctions in the nature of the topology are closely related to the synthesis problem domain itself. Control system design and other signal processing system synthesis domains are usually oriented toward computational topology synthesis or algorithm discovery. These systems are finally implemented in computer-related devices. So there are no physical building blocks that may constrain the topology search process. Any kind of computational units/process can be used. For synthesis of physical topologies, it is natural to use a modeling scheme that can explicitly represent the physical building blocks and their connections.

- Representing and preserving the building blocks of the design domain

  Dynamic systems can be evolved in two types of design space that preserve the topological structure of the systems. One is the physical design space, as used in the case of circuit diagrams, in which the nodes and edges represent the physical building blocks and connections. Another is the functional design space, as used in the case of bond graphs, where the nodes can be physical building blocks or functional building blocks, when used to model some mechanical systems such as linkage mechanisms. The edges in bond graphs do not correspond to actual physical connections between components, but correspond to functional (or computational) relations, as in the case of the block diagram. To make the evolved design solutions of bond graph implementable with physical building blocks, it is important to encapsulate the sub-module of a whole physical building block as one building block in the synthesis space and not to allow topological operations to disrupt its internal structure. We call this guideline *the principle of physical-meaning-preserving topology search*. To this point, it is not clear whether evolutionary topology synthesis in physical design space is better than in the physical-meaning-preserving functional design space or the contrary is true.

- Locality or causality of topology representations

  Locality of a representation measures how genotypic neighbors correspond to phenotypic neighbors. Sometimes, in evolutionary computation, it is also called causality (Rosca & Ballard1995), which is not to be confused with the notion of causality as used in the literature of bond graphs (also described below). In the context of topology synthesis, it measures how well topological similarity predicts functional similarity. Locality of representation is regarded as critical to the evolvability of the search space and the efficiency of evolutionary search (Rothlauf, 2002). One interesting comparison here is that of circuit diagrams with bond graphs. Both can used to encode and evolve lumped-parameter dynamic systems. But there is a significant distinction between these two – bond graphs use 0-junctions and 1-junctions to explicitly represent relationships between building blocks. This means that a single mutation on a junction type can lead to a change of relationship from parallel to serial or vice-versa. This kind of change will dramatically modify functional behavior of a dynamic system. Thus, bond graphs have lower locality. However, on the other hand, this means that bond graphs have stronger topology manipulation capability than local circuit diagram operations. How this difference of locality will affect topology search is not yet clear.

- Capability to support hierarchical modular modeling

  To increase the scalability of evolutionary topology synthesis, it is critical that the representation scheme support hierarchical composition — the capability to encapsulate sub-modules into higher-level building blocks. In this sense, support for multi-port building blocks becomes important, which means that the standard signal flow graphs mentioned in the section above are not appropriate in many cases. Bond graphs, circuit diagrams and block diagrams all support hierarchical composition of building blocks of different levels.

Since bond graphs have several unique advantages in representing multi-domain dynamic systems, we have chosen to use them in our work. The next section will provide a brief introduction.

3.3 Bond Graph Representation of Dynamic Systems

The bond graph is a modeling tool that provides a unified approach to the modeling and analysis of dynamic systems (Paynter, 1961; Karnopp et al., 2000). Bond graph models can describe the dynamic behavior of physical systems by the connection of idealized, lumped elements based on the principle of conservation of power.

Bond graphs consist of elements (nodes) and bonds (edges). The behavior of the elements is described by constitutive equations, power flow and causal relationships. The state of the system is represented in terms of generalized efforts and flows on the bonds. There are several types of basic elements (building blocks), each of which performs analogous roles across energy domains.

The first type -- C, I, and R elements -- are passive one-port elements that contain no sources of power, and represent capacitors, inductors, and resistors in the electrical domain and other equivalent elements in other domains.

The second type, Se and Sf, are active one-port elements that are sources of power, and that represent effort sources and flow sources, respectively (for example, sources of voltage or current, respectively, in the electrical domain).

The third type, TF and GY, are two-port elements, and represent transformers and gyrators, respectively. The transformer TF relates efforts to efforts and flows to flows, while gyrators GY relate the effort at one port to the flow at the other. Power is conserved in these elements.

A fourth type, denoted as 0 or 1 on bond graphs, represents junctions, which are N-port elements. They serve to interconnect other elements into subsystems or system models. A 0-junction represents the generic parallel connections where all connected bonds have identical “effort” and the sum of flows is 0. A 1-junction represents generic serial connections, where the connected bonds have identical “flow” and the sum of efforts is 0. It should be noted that junctions are not physical entities, they represent the relations among other building blocks, so a bond graphs is one kind of relational graph (Polya, 1962). Table 3.1 summarizes the major energy conservation/transformation elements of bond graphs in translational, rotational, electrical and hydraulic domains.

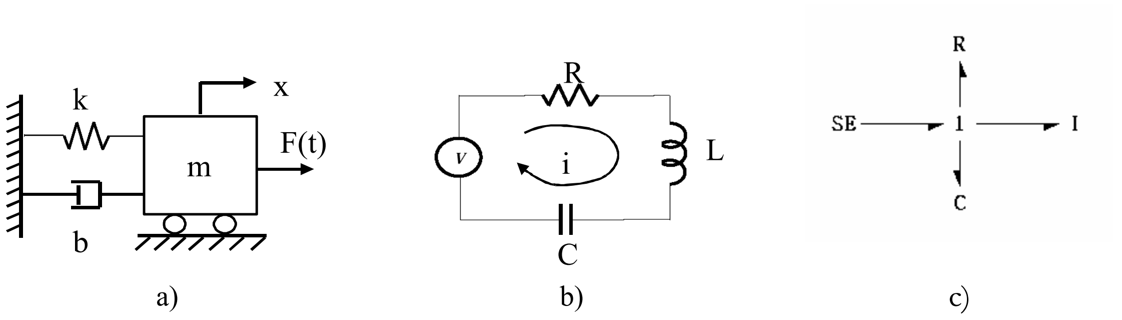
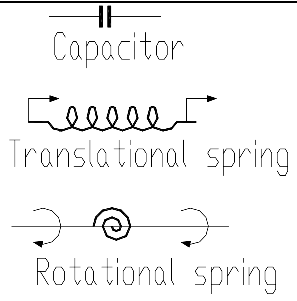
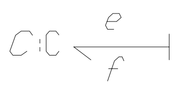
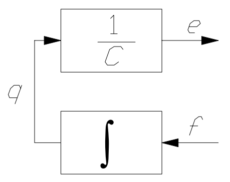
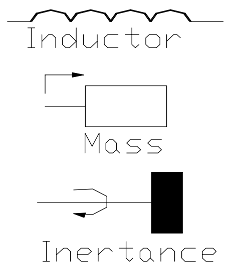
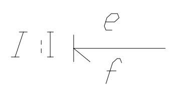
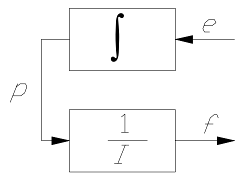
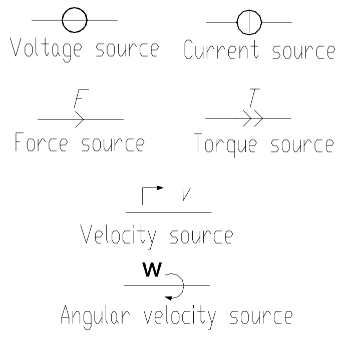
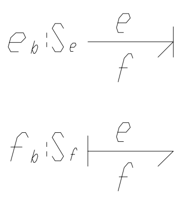
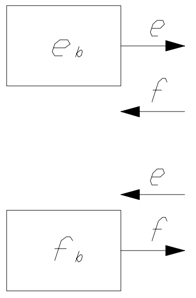
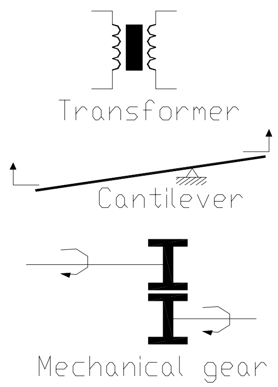
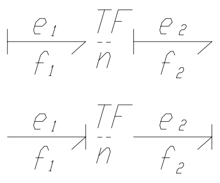
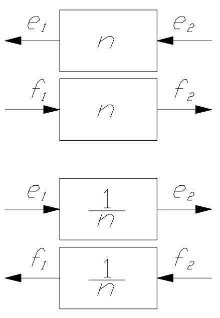


Figure 3.5 Bond graph representation of a mechanical system and an electric circuit

An example of a bond graph model is shown in Figure 3.5 (c), consisting of Se, 1-junction, C, I, and R elements, representing a bond graph model of a mechanical mass, spring and damper system (Figure 3.5 c), or an RLC electric circuit (Figure 3.5 b). The Se corresponds with force in the mechanical domain, and voltage in the electrical. The 1-junction means common velocity for 1) the force source, 2) the end of the spring, 3) the end of the damper, and 4) the mass in the mechanical system, and implies current in the RLC loop is common. The R, I, and C represent the damper, inertia (of mass), and spring in the mechanical system, respectively, and the resistor, inductor, and capacitor in the electrical circuit, respectively.

Table 3.1 Major energy conservation/transformation elements in bond graphs

Domain-specific symbols	Bond graph element	Equations	Block diagram expansion
		$e = \frac{1}{C} q$ $q = \int e dt + q(0)$	
		$f = \frac{1}{I} p$ $p = \int e dt + p(0)$	
 Voltage source Current source F Force source T Torque source V Velocity source W Angular velocity source		$e = e_b$ $f = f_b$	
	 e_1 f_1 TF n e_2 f_2 e_1 f_1 TF n e_2 f_2	$f_2 = nf_1$ $e_1 = ne_2$ $f_1 = f_2/n$ $e_2 = e_1/n$	 e_1 n e_2 f_1 n f_2 e_1 1/n e_2 f_1 1/n f_2

One of the important concepts in bond graphs is causality. If two components are bonded together in a bond graph, we can think of the effort as causing one component to respond with the flow while the flow causes the first component to respond with the effort. Thus the cause-effect relations for efforts and flows are represented in opposite directions. A single mark on a bond, which is called the causal stroke, indicates how e and f simultaneously are determined causally on a bond (Figure 3.6). This concept plays a great role in allowing determination of the feasibility of a design very simply in an early stage of evaluating the functional performance of a bond graph in evolutionary synthesis, as will be discussed in the next section.

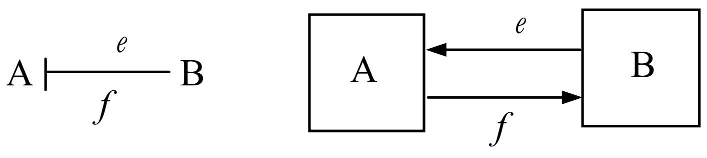


Figure 3.6 The causality concept in bond graphs

3.4 Automated Synthesis of Bond Graphs by Genetic Programming

Following the generic framework discussed in Section 2.3.3 for topologically open-ended synthesis by genetic programming, there are four preparatory steps that must be taken before applying genetic programming to topological synthesis of bond graphs. We discuss each of them and the decisions that we have made in this process.

3.4.1 Program Architecture

In our basic approach to bond graph evolution, an individual contains exactly one GP tree without any automatically defined function (ADF) branches. There are no other architecture-altering mechanisms that may introduce ADFs later on. This decision is based on our intuition that ADFs work only in problems whose solutions have strong regularity or repetitive modules and only if the context in which the ADFs are applied is appropriate. So for generic problems without repetitive modules, the capability of ADFs to improve scalability of evolution is limited. Actually, most of the inventions evolved by GP were also obtained without using ADFs (Koza et al., 2003b). Figure 3.7 shows one typical GP tree for bond graph evolution.

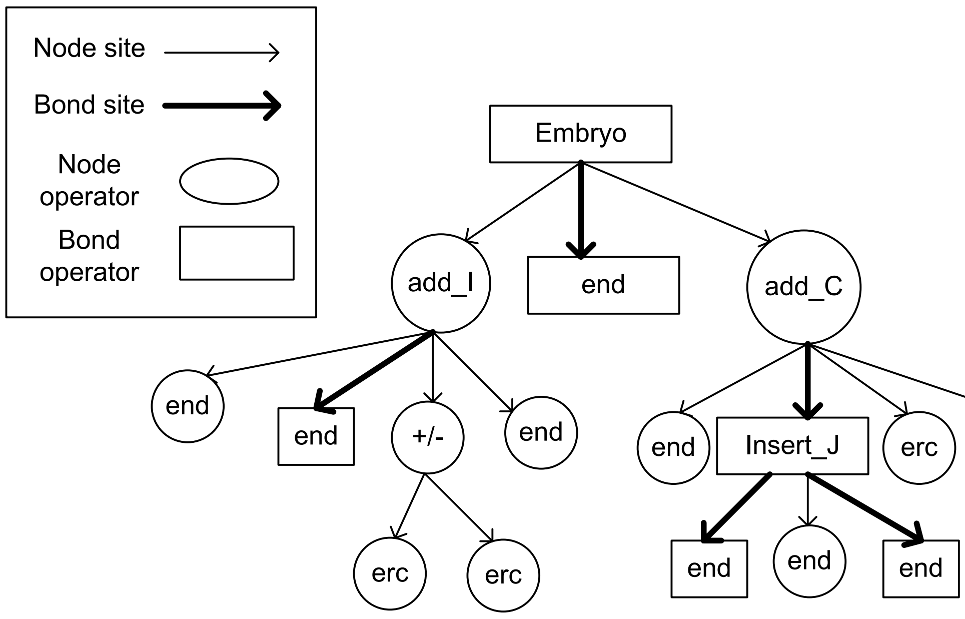


Figure 3.7 A typical GP tree

3.4.2 Types of Building Blocks and Modifiable Sites

Type is an important concept in topology synthesis; strongly typed genetic programming exploits this concept. The type of a building block or modifiable site specifies the characteristics that decide what kind of operations can be applied upon it. In the basic bond graph synthesis approach by genetic programming, five types of building blocks are used as illustrated in Table 3.2

A modifiable site is a place in the bond graph that can be modified to grow or shrink the topology. It may have any of the listed types in Table 3.2 except the numeric type. Actually, the types can be even more specific, such as making only a 1-junction a separate type. The objective of the type of a modifiable site is to allow only the appropriate types of topology operations to be applied to ensure valid topological manipulations.

Table 3.2 Types of building blocks and modifiable sites

Type	**Possible Operation**
Bond (2-port)	can be modified by inserting a two-port element or module such as a transformer, gyrator, etc
1-port Element	can be modified by changing its function (resistor, inductor, or capacitor, or other one-port module)
2-port Element	can be modified by removing the element and connecting its bond or separating its bond, by inserting a new two port element to either one of its two sides
Junction (N-port)	A junction can be modified by switching between junction 1 or junction 0; by attaching a one-port element or by connecting to another junction
Numeric	A numeric node can be replaced by another numeric node, either a random constant or a numeric expression sub-tree.

3.4.3 Embryos

An embryo bond graph specifies the starting point of the developmental growth process by executing the genetic programming tree. Each embryo will have one or more modifiable sites where the topological operations can be applied to grow the topology. In evolutionary synthesis of bond graphs as described in this dissertation, there are two types of embryos. One is the type with a minimal set of components, which aims to introduce as little bias as possible into the evolutionary design process. This case corresponds to classic open-ended design. The other type is for re-design problems, in which a bond graph model of an existing dynamic system is used as the starting point and there are one or more modifiable sites to allow further growth. In most of the problems in this dissertation, the minimal embryo bond graphs are used, as illustrated in Figure 3.8.

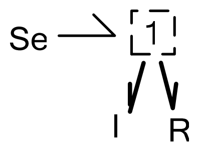


Figure 3.8 An embryo bond graph with one junction-type modifiable site at a 1-junction

There is an important issue in choosing embryos with multiple modifiable sites. In the standard developmental genetic programming for topology synthesis outlined in Section 2.3.3, there is no interaction between different branches of the GP tree. As a result, the evolved bond graphs usually have several separate branches stretching away from the embryo. For many systems, this bias may not be acceptable and special mechanisms are needed to allow merging of the separated local bond graph modules or interaction of separate topology growth processes.

3.4.4 Terminal Set and Function Set

The terminal set and function set determine all the possible building blocks in the developmental program in a GP tree. The difference between terminals and functions is simply that the former can only be leaf nodes in the GP tree, without requiring any child nodes or arguments, while the latter are not leaf nodes.

According to their usage, terminals and functions can be classified into different groups. The following set of terminals and functions are used in the basic approach for bond graph evolutionary synthesis:

- **GP Functions for topology manipulation/growth**

  Add_X functions (Figure 3.9**):** where X can be a resistor, a capacitor, an inductor or any other one-port component. These terminals add the component X to an existing junction and create two new modifiable sites; one is of the element type, and the other is of the bond type. Since ADD_X needs a junction as the modification point, its input (return) type is junction. Since it inherits one modifiable junction site and creates one new bond-type modifiable site and one element type modifiable site, it has three branches (arguments). According to how many parameters X has, more numeric-type branches can be added, one branch for each parameter. The execution process of this function is simply first to retrieve the modification point from the current topology and then add an X, create the new modifiable sites in the topology, and then call its sub-tree routines. It uses a pre-order traversal.

Replace_X functions: replace current one-port element with a new element X, where X can be a resistor, a capacitor, an inductor or any other one-port component. Its input type is one-port element. Since it only inherits one modifiable site, it has only one topology modification branch.

Insert_X functions (Figure 3.9**)**: insert a two-port component into the bond-type modifiable site. Its input type is bond. When X is a junction, it creates a new junction and a new bond. As a result, it has three branches for topology growth.

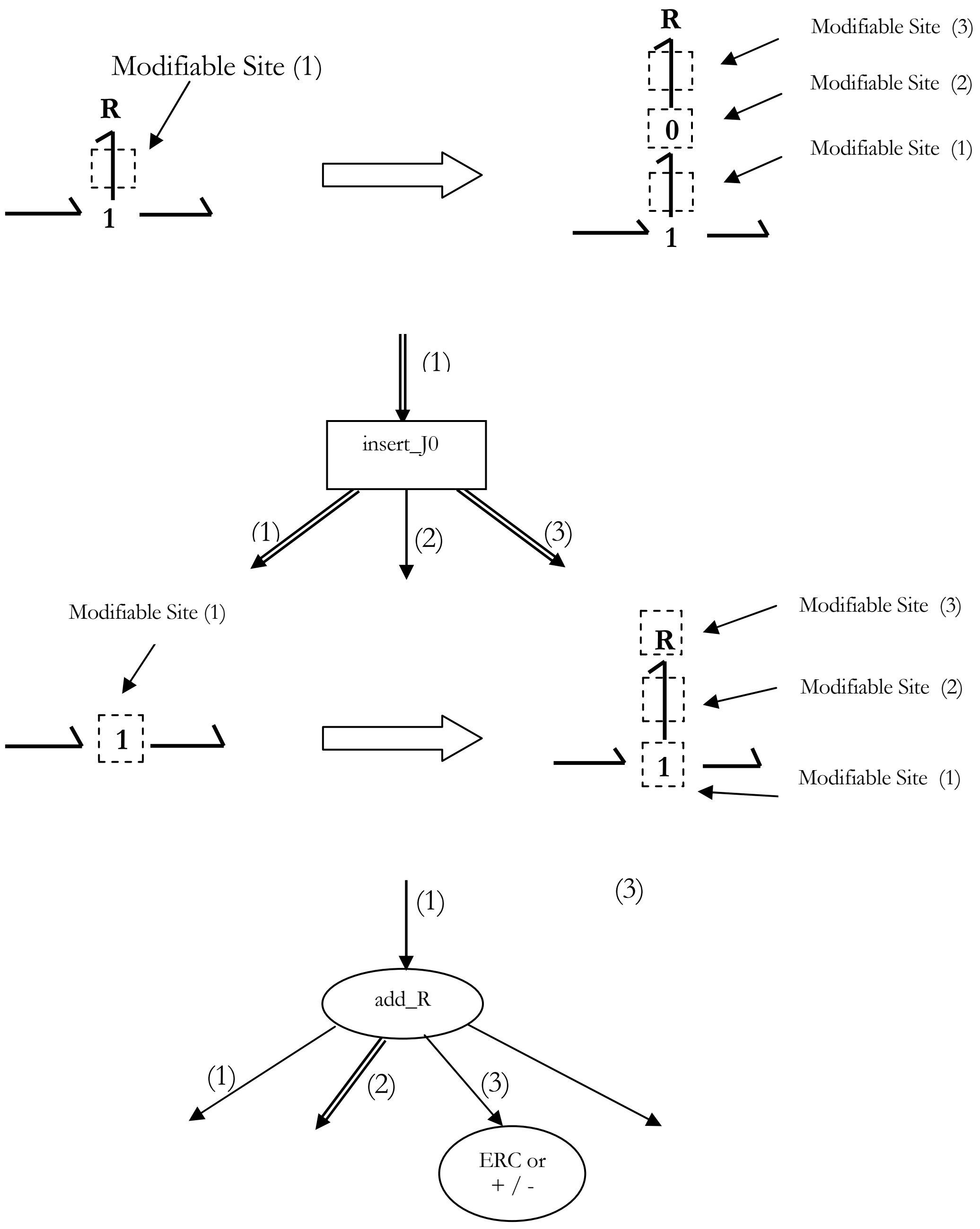


Figure 3.9 Illustration of insert_X and add_X functions

- **Terminals and functions for establishing parameters of components**

These functions and terminals are used to evolve numeric subtrees to establish the required parameters for different elements and/or modules. Typically, only arithmetic functions plus ephemeral random constants (i.e., numerical constants) are used

■ **Terminals and functions for regulating the topology growth process**

To regulate the topology growth process, such as telling it when to stop, when to jump, many different kinds of regulating terminals can be introduced. In the basic approach, three end_X terminals are used to stop the growth process. Since they do not have any branches, the growth process simply stops when encountering end_x at a branch.

Table 3.3 Definition of function set and terminal set

Name	#Args	Description
add_C	4	Add a C element to a junction
add_I	4	Add an I element to a junction
add_R	4	Add an R element to a junction
insert_J0	3	Insert a 0-junction in a bond
insert_J1	3	Insert a 1-junction in a bond
replace_C	2	Replace the current element with a C element
replace_ I	2	Replace the current element with an I element
replace_ R	2	Replace the current element with an R element
end_A	0	End terminal for add element
end_I	0	End terminal for insert junction
end_R	0	End terminal for replace element
+	2	Add two numeric nodes
-	2	Subtract two numeric nodes
*	2	Multiply two numeric nodes
/	2	Divide two numeric nodes
erc	0	Ephemeral random constant (ERC)

The whole set of terminals and functions are summarized in Table 3.3.

There are many possible sets of terminals and functions that are able to grow arbitrary bond graphs, theoretically. But they will have different biases when used in topological exploration, so different sets are appropriate for different types of problems. The question of how to choose an appropriate set will be investigated in Chapter 7.

3.4.5 Fitness Evaluation and Simulation Platform Selection

One of the major preparatory steps in applying genetic programming to a problem is to set up the simulation environment to evaluate the solutions. Models represented by bond graphs can be evaluated in several ways. One could use the commercial bond graph simulation systems, such as 20-Sim®, Symbol2000®, Dymola®, etc. The advantage is that these tools are very powerful and can evaluate bond graph models of very complex real-world problems. The difficulty is that all these tools are developed as interactive CAD tools, rather than for automated synthesis. They often require a compilation process for each new topology model, which is very time-consuming. However, these compilation steps can be reduced by keeping track of topologies. If only the parameters of a topology are modified, the same compiled binary code can be loaded for simulation.

In this dissertation, the simulation package used with bond graphs is a home-made C++ package, which can run "on the fly" without recompilation for each new topology.

3.5 Benchmark Problems

Designing a good benchmark problem for investigating open-ended system synthesis is difficult for several reasons. The generic electrical circuit synthesis problem of Koza et al. (1999), widely known as it is, requires a complicated simulation package such as SPICE, and the simulation complexity is fairly high. Other topology synthesis problems, such as molecular design, suffer from the lack of a "standard" and universally available simulation engine. Communication network design (Dengiz, 1997), neural network design (Stanley & Miikkulainen, 2002) and other pure graph-oriented topology synthesis procedures (Luke & Spector,1996), however, have quite different characteristics in terms of the search space and the fitness landscape. As a result, we introduce a set of bond-graph-based benchmark problems for evaluating methods and identifying issues in automated evolutionary synthesis of mechatronic systems.

3.5.1 Eigenvalue placement problem

In this problem, an analog circuit represented as a bond graph model is to be synthesized (including its topology and sizing of components) to achieve a specific behavior. The bond graph model domain to be used is composed of a set of inductors (I), resistors (R), capacitors (C), transformers (TF), gyrators (GY), and Sources of Effort (SE). Our synthesis task in the K-eigenvalue placement problem is to evolve a dynamic system with K eigenvalues that approximate a pre-specified set as closely as possible. By increasing the number of eigenvalues (K) (and, correspondingly, the order of the dynamic system), we can define a sequence of synthesis problems of increasing difficulty for use in evaluating the scalability of various techniques. The problem difficulty can also be varied by choosing different sets of primitive building blocks (modules) to be used in the search process. In this dissertation we use a set of causally well-posed primitives (CWP), in which each junction has a resistor attached, 1-junctions always have an inductor attached, 0-junctions have a capacitor attached, 0-junctions can only be adjacent to 1-junctions, and vice-versa. Such systems are readily realized in the form of mechanical systems as well as in the electrical domain. The space of all possible topologies generated using these primitives consists of connected graphs without loops (i.e., trees), and the conditions are strong enough to guarantee that the corresponding systems satisfy causality constraints.

Given a bond graph model, the fitness evaluation is conducted as follows:

1) Derive the A matrix for the candidate model
2) Compute the eigenvalues of the A matrix
3) Compute an error measure based on the target set of eigenvalues

Procedure 3.1 shows how to match the resulting eigenvalues to the target eigenvalue set to compute a/an (relative-distance-based) error.

4) Compute the fitness measure from the error
The fitness of a bond graph model is computed by normalizing that error measure into a fitness value (to be maximized) between 0 and 1 according the following scaling rule:
if distance/order <0.1 then fitness = 0.1 /(0.1+distance/order)
else fitness = 5.05 /(10+distance/order)
where order is the number of energy-storing components (capacitors and inductors).

```
Procedure 3.1 calculating distance between two sets of eigenvalues
Input: target eigenvalue set $(x_i^t, y_i^t), i = 1,\ldots,O_t$, where $O_t$ is the number of target eigenvalues;
    candidate eigenvalue set $(x_i, y_i), i = 1,\ldots,O_c$, where $O_c$ is the number of candidate eigenvalues.
Output: summed relative distances between two sets of eigenvalues

if order (cardinality) of candidate set $O_c$ is different from that of target set, $O_t$
  return $\infty$
otherwise
        calculate the average length $l_{ave}$ of target eigenvalues (distance from the origin),
            $l_{ave} = \sum_{i=1}^{O_t} l_i$ , $l_i$ is the length of $i_{th}$ target eigenvalue
        label all eigenvalues as unmatched
        $distance \leftarrow 0$
        for i = 1 to $O_t$
        find an unmatched candidate eigenvalue j, which is closest to target eigenvalue i
        label the candidate eigenvalue j as matched
        $distance \leftarrow distance + \sqrt{(x_j - x_i^t)^2 + (y_j - y_i^t)^2} / l_{ave}$
            where $(x_j, y_j)$ is the candidate eigenvalue, and $(x_i^t, y_i^t)$ is the current target eigenvalue i.
        end for
        return distance;
```

The fitness scaling measure in 4) above is chosen such that 50% of the fitness range is occupied by error measures greater than 0.1, and the remainder is used for error measures less than 0.1. It is continuous, with a single slope discontinuity, and approaches 1 asymptotically as the error measure approaches 0. Many evolutionary methods (using tournament or rank-based selection, for example) will not require or be sensitive to the scaling suggested in step 4, but some methods depend on an absolute fitness measure, and this is suggested for use in such cases, and also converts the fitness into the classical maximization form of evolutionary computation, without a singularity for an error of 0 (an exact match of candidates to the targets).

Some interesting variants of this problem formulation

It is possible to use this problem formulation in at least two interesting ways. First, a set of eigenvalues may be generated arbitrarily, without knowledge of a system that would have such a set of eigenvalues. This provides an open-ended problem with no known answer and unknown search properties. Second, it is possible, and often useful, to start from a known system, calculate its eigenvalues, and use that set as the target, thereby providing a target with at least one known optimal solution (error 0, fitness 1). This allows the user to readily assess the efficiency of the evolutionary design system as a global optimizer.

This problem formulation explicitly specifies the order of the system. Such a constraint is useful and often pre-specified in real-world design applications, as cost often correlates well with order. Alternative formulations could allow candidate systems of higher order than the target to receive finite error measures, but require that unmatched eigenvalues be outside the range of the target set, for example. Another interesting formulation of the problem, with many of the same properties, but requiring much less computational effort, would use the coefficients of the characteristic polynomial of the A matrix as the targets to be matched by those coefficients of candidate systems, thereby avoiding the cost of numerically calculating the eigenvalues. However, systems with widely differing eigenvalues can possess only small differences in characteristic polynomials, so the authors have chosen here to study the eigenvalues themselves, with the hope of better illuminating structure/function relationships in the systems evolved.

3.5.2 Analog filter synthesis

In this problem, the objective is to evolve a bond graph model equivalent to an analog circuit composed of passive elements including resistors, capacitors, inductors (and transformers and/or gyrators, sometimes) such that the circuit can be used as a high-pass, low-pass, or band-pass filter.

To calculate the fitness of a bond graph, first, the state equation of the bond graph is derived, which is then passed to a Matlab routine to calculate its frequency response. Within the frequency range of interest, 100 points are sampled uniformly and are compared to the magnitudes of the frequency response at the sample points with the target frequency response. Their differences are computed and the squared sum of differences is used as the raw fitness, defined as $Fitness_{raw}$. Finally, the normalized fitness is calculated according to:

$$Fitness_{norm} = 0.5 + 1 \big/ (1 + Fitness_{raw})$$

3.6 Summary

In this chapter, a brief history of topologically open-ended synthesis of dynamic systems is reviewed. Different graphical modeling schemes used for dynamic synthesis are compared with a through reexamination of how the characteristics of the graphical structure and its correspondence with physical systems and functions could affect the evolvability of the topology space in automated synthesis by genetic programming. An introduction of bond graph modeling approach for dynamic systems is presented in Section 3.1. A basic approach for topologically open-ended synthesis of bond graphs is proposed, following the framework in Section 2.3. This chapter concludes with two benchmark problems for bond graph synthesis, which are to be used in later chapters.

Chapter 4

HIERARCHICAL FAIR COMPETITION (HFC): A SUSTAINABLE EVOLUTIONARY COMPUTATION MODEL

In Chapters 2 and 3 decades of effort to pursue sustainable evolutionary search were discussed. Hundreds of techniques have been proposed to deal with premature convergence/stagnation, evolvability, and scalability problems in evolutionary algorithms (EAs). Unfortunately, evolutionary algorithms are still convergent algorithms and premature convergence still prevents us from solving many complex problems. We observed that one common point of most of the research is to keep the framework of canonical evolutionary algorithms intact, the one based on a simplified interpretation of the Darwinian principle of survival of the fittest for evolution. In this chapter we argue that the inherent convergent nature of the canonical evolutionary computation model contributes a lot to the difficulty of achieving sustainable search and we propose an alternative.

4.1 The Convergent Nature of the Prevailing Evolutionary Computation Model

4.1.1 Two Observations and Two Common Implicit Assumptions in Typical EA Frameworks

Current EA techniques carry with them implicit assumptions about evolutionary search. There are two important observations about and two implicit assumptions of most current EAs that can prove detrimental to their performance on many difficult problems, but which HFC can help to alleviate.

Observation I: The average absolute fitness of the population increases continually

One prominent observation about the typical EA framework is that as the evolutionary process goes on, the average absolute fitness of the population gets higher and higher. The population then seems to gradually lose its capability for exploration, until convergence to the global optimum or premature convergence (to some undesired region of the search space) finally occurs when the fitness difference between individuals become increasingly small. Many algorithms use rank-based or tournament-type selection methods to enable them to continue to distinguish between better and worse individuals in the population, but that does not solve this problem, as explained below.

Many people associate the loss of exploratory capability with the loss of diversity and try to increase the diversity of the population as a remedy. But actually, loss of diversity is just a symptom of this phenomenon –diversity per se is not sufficient: what is required is to maintain useful diversity. The more direct reason that premature convergence occurs is that with increasing average fitness of the population, only those new individuals with competitively high fitness tend to be allowed to survive. However, new "explorer" individuals sampling fairly different regions of the search space usually have low fitnesses, until enough local exploration and exploitation of their beneficial characteristics occur. Even for those with competitive fitnesses, their sparse distribution in distant regions still exposes them to the risk of being lost as the result of insufficient sampling or genetic drift. This explains why simply increasing the mutation rate or inserting random individuals doesn't work well: a highly evolved individual with co-adapted components, when mutated strongly or recombined with a random individual, almost always leads to inferior offspring. Random individuals or extensively mutated individuals usually have low fitness and cannot survive long enough to get their new genetic material exploited, as their loci have not yet been able to co-adapt to become instances of high-fitness schemata. This means that the higher the average fitness of the population, the less the capability it has to make and utilize large genetic modification, while small modification generally only mildly perturbs the high-fitness individuals and cannot move them out of local optima (or attractors). This gradual loss of the capability to incorporate new genetic material is one of the key factors leading to premature convergence.

Some mechanism is needed to reduce this effect of increasing average fitness of the population (or to reduce the domination of the population by the earliest-discovered individuals of the highest fitness), which makes it hard for new individuals (with less well co-adapted genetic material and usually with lower fitnesses) to survive. The Species Conserving GA (Li et al., 2002) uses similarity-based niching to explicitly keep those exploratory "not highly fit but different enough species seed" individuals. Fitness sharing and crowding use a similar strategy to keep "not highly fit but different enough" exploratory individuals by limiting the number of individuals at specific local search regions called niches. However, such techniques suffer from a property of the search spaces of many difficult problems, as described next.

Observation II: Many difficult problems have enormous and highly multimodal search spaces

Many difficult search problems (and in fact, many of those at which the HFC methods described here are aimed) are made difficult because their search spaces are enormous, with high dimensionality and high multimodality and the number of local optima is many larger than the population sizes typical used. (There are other types of difficult search problems, of course -- for example, needle-in-a-haystack problems -- that these methods may not address any better than other methods will). Distance or similarity-based niching and multi-population and search space division methods are widely used to maintain a desired level of population diversity. Crowding (De Jong, 1975) and deterministic crowding (Mahfoud, 1992) work by replacing similar individuals to allow space for diverse individuals. Fitness sharing uses a carrying capacity of niches to keep diverse individuals. The island parallel model uses multiple subpopulations in the hope that each accommodates different candidate solutions, with only infrequent migration to aid exploitation. SCGA (Li et al., 2002) uses similarity-based species protection to promote diversity. Tsutsui et al. (1998) use search space division to keep diversified populations. All these techniques work by spreading or pushing individuals “horizontally” (rather than "vertically" along the fitness dimension in Figure 4.1) to different areas in the search space -- for example, fitness sharing systematically prevents individuals from clustering at a few high-fitness peaks. They strive to maintain genotypic or phenotypic diversity. But they work well only with sufficiently large population sizes or sufficient numbers of subpopulations, which, in order to be sufficient, may sometimes be so large as to be impractical. Koza et al. (1999) worked with population sizes of 350,000 or even larger to achieve satisfactory results, confirming this capability, but at significant cost. However, as is shown in Section 4.6, increasing population size is not a scalable solution for difficult problems and may induce unnecessarily high computational cost.

It is clear that horizontal genotype or phenotype spreading techniques mentioned above don’t solve the scalability problem for addressing difficult problems, as they perform poorly as the size and multimodality of the search space increases. As illustrated in Figure. 4.1, even a one-dimensional search problem can be highly multimodal. For problems of high dimensionality (many engineering problems have 50-100 or more independent design variables) or for problems in which the number of local optima grows exponentially with problem size, the search space and number of local peaks (or attractors) becomes too large to allocate sufficient population size to accommodate representatives of most of them. Even though there are some techniques to reduce the number of effective niches to some extent (Goldberg et al., 1992), the issue of requiring a huge population size is still not resolved. This difficulty is caused by some deep design assumptions of the current EA framework about how an EA should work.

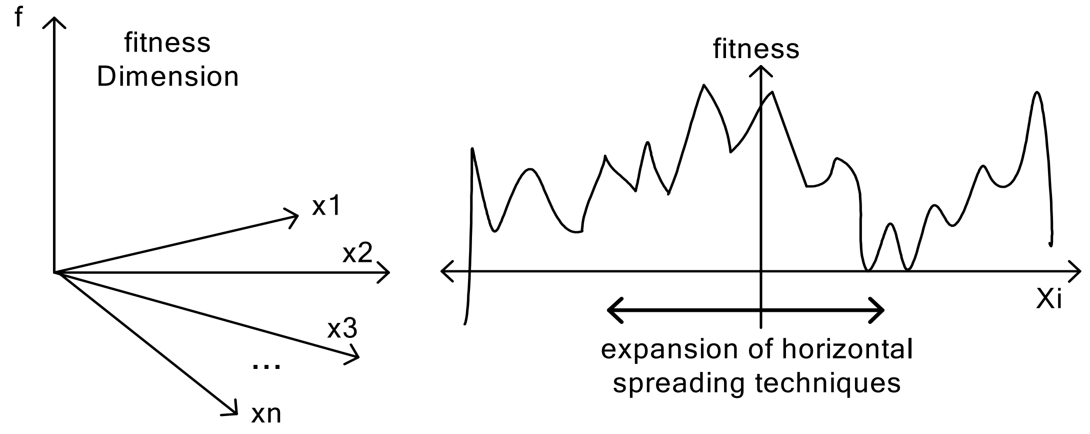


Figure 4.1 Enormity of the search space and of the number of local optima and expansion in horizontal spreading EAs

Common Implicit Assumption I: Problems can be reasonably addressed using an EA conducting a single-thrust search

As the motivation of using EAs is to find solutions with high fitness, EAs are usually taken as a single-thrust search process: starting from random populations, some lower-level building blocks are discovered and assembled into higher-level building blocks (or, as in the case of evolution strategies, some decision variables evolve decreasing magnitudes for their perturbations) and the best fitness and average fitness of the population increase until stagnating or reaching specified stopping criteria. For example, messy GAs are to be sized such that the initial population is sufficient to ensure an adequate supply of raw building blocks (Goldberg et al., 1993). Population-sizing models for GAs are derived (Goldberg, 1989; Goldberg et al., 1992), based on takeover times of building blocks in an (implicitly) single thrust evolution. Impractically large population sizes would be needed to make fitness sharing work for massively multi-modal problems (Goldberg et al., 1992). Huge population sizes are used to ensure sufficient diversity in multi-population parallel GP (Koza et al., 1999), still based on the single-thrust EA model.

For difficult problems with a huge number of local optima and a complex search space, it is inappropriate to assume that all building blocks or potentially important combinations of genetic material are present in the initialization stage, and even if they were, that they would be able to survive long enough to be exploited, due to unbalanced sampling resulting from their different degrees of salience (Goldberg, 1999) treats the building block aspect). Some type of "assembly line" continuing EA search process is needed. In such a process, a random individual generator continuously feeds new individuals into the EA population. While this mechanism would be completely ineffective in an ordinary GA (because the new individuals are grossly uncompetitive in the populations they enter and make poor mates for highly evolved solutions), this kind of continuous, probabilistically complete supply of building blocks can partially relieve messy GAs and other EA techniques from the burden of infeasibly large population size requirements for difficult problems. That is, no particular minimum population size is required at any one instant. However, simply introducing random individuals continuously into a population experiencing increasing average fitness is not sufficient to achieve the desired result.

Common Implicit Assumption II: Premature convergence can be adequately handled by maintaining the genotypic and/or phenotypic diversity of a population of a reasonable size throughout the run

Loss of genotypic diversity among intermediate-fitness individuals typically leads to loss of further exploratory power. The entities converging could be structures near the root of a GP tree, a group of loci in a GA, or a set of decision variables in ES. EAs work by accumulating a succession of improvements. High fitness individuals are usually evolved in this incremental way, in which lower-level building blocks are assembled into higher-level building blocks, or the structures of moderate-fitness individuals become fixed. The higher the fitness of the individual, the more tightly those loci become co-adapted to each other. In the traditional single-thrust EA framework, the evolutionary process cannot guarantee that all such intermediate–level structures grow at similar speeds, and as a result, only some of them get a chance to be sampled and recombined, while others are lost. At the same time, the "critical initial generations" effect says, "unless a schema (or its components) grows at the outset, its chances for success later are quite poor" (Goldberg, 2002). This is because single-thrust EAs do not tend to preserve the stepping stones that were used to make the initial climb up the fitness landscape, nor to maintain a "place" where those stepping stones may be regenerated. Thus, they lack the capability to incorporate new genetic material continually, with high probability. All the individuals are marching toward the fitness frontier or the highest-fitness regions of the search space discovered to date. So essentially, for single-thrust EAs, the search process converges rapidly at the outset with regard to the majority of the intermediate-fitness structures present in the population. The diversity-preservation mechanisms soon concentrate on preserving genotypic diversity among the highest-fitness individuals, neglecting the individuals that constituted the "path" to the top. The premature convergence of the best individuals is then to a large extent caused by the premature loss of useful lower-fitness individuals.

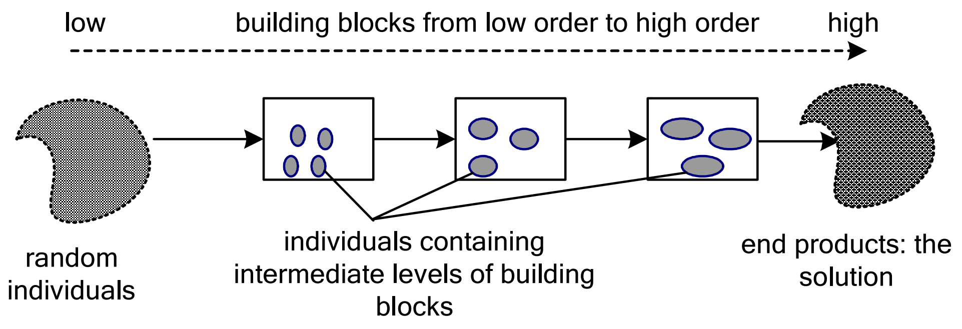

low
building blocks from low order to high order
high
random individuals
individuals containing intermediate levels of building blocks
end products: the solution

Figure 4.2 The assembly line structure of the continuing EA model

The importance of maintaining and continuously generating new "stepping stones" suggests a new model for an EA (Figure 4.2). This model arranges processing units in a series to form an "assembly line". Randomly generated individuals are fed into one end unit of this assembly line, in which individuals containing random "chunks" (although not necessarily adjacent on the chromosome) with different desirable characteristics are assembled into individuals with better-than-random fitness. The output of this first processing unit, the first-level stepping stones, is continually fed to the next processing unit for assembly of higher-performance stepping stones, and so on. The HFC model provides exactly such an assembly line processing structure for continuously supplying and assembling intermediate levels of stepping stones. For many problems, many alternative sets of stepping stones will emerge, providing many alternative paths from random-fitness individuals to individuals of very high fitness, and a continuing capability to generate additional such paths.

4.1.2 Premature Convergence and Stagnation: an Explanation

The HFC concept postulates that an important factor in the convergent nature of conventional EAs is their over-attention to the high-fitness individuals and neglecting of the importance of recombination and mutation among low-fitness individuals, even at later stages of evolution. The HFC concept is to sustain a supply of lower-fitness individuals that represent low levels of “organization” – possessing low-order building blocks that may even be created at random, but not yet (or still) be present in higher-level individuals. These individuals are "sheltered" – allowed to recombine and mutate, regardless of the fact that higher-fitness individuals are present. However, the goal is not to preserve the low-fitness offspring of bad matings among high-fitness individuals, so hierarchical elitism is employed to prevent that.

High-fitness individuals in conventional EAs usually have difficulty in modifying their basic "frameworks"—groups of coupled variables—without decreasing their fitnesses dramatically, especially in the case of genetic programming. As the average absolute fitness of the population goes up, it becomes increasingly difficult for the degraded offspring of high-fitness parents to survive. The individuals that do change their basic frameworks just cannot survive with their low fitnesses. So, the probability of framework changes becomes increasingly small as evolution goes on. In addition, the frameworks of current high-fitness individuals are established with very limited testing during the early stages of evolution, which certainly does not guarantee their global superiority. After the potential of these earliest-discovered frameworks is exhausted, no more framework innovation is likely, and genetic programming begins to exhibit stagnation. The scenario here is similar to the canalization concept of developmental biology.

This loss-of-explorative-capability (LEC) hypothesis to explain premature convergence can be understood more easily in genetic programming, where the early-stage evolution usually establishes the framework (or topological structure) of the individuals. However, LEC is also applicable to many GA problems with high epistasis, where disruption of the strongly coupled components of the genome usually leads to dramatic decrease of the fitness of the offspring. It is not the fitness diversity *per se* that contributes to the sustainable search capability of HFC or any other successful approach to sustainable evolution. In problems amenable to HFC, fitness levels are assumed to correspond to degrees of organization, or coupling among components, from unbiased random individuals to highly coupled, high-fitness individuals.

4.1.3 Requirements for Sustainable Evolutionary Algorithms

Given an understanding of how the above two observations about and two implicit assumptions of traditional EAs reflect impediments to their performance, the HFC model has been devised to meet two requirements described below that are believed to be important to assure continuing search capability for an EA. They may be seen as motivated from reflection on the continuing existence of a wealth of "vertically distributed" niches in the natural environment, in which survival of the fittest is a local concept, and the requirements for survival in some niches may be very different in some than in others. The requirements for insect survival can be very different from those for survival of mammals, for example, and nature continues to provide environmental niches in which each has superior survival ability.

Requirement I: The Continuous Supply and Incorporation of Low-Level Genetic Material

A continuing EA must assure that a continual supply of low-level novel genetic material is available for assembly or mutation to discover (or rediscover) individuals of intermediate fitness levels, in order to sustain the search indefinitely.

In HFC, the individuals of the whole population are organized into a hierarchy of fitness levels. A generator of random individuals continuously feeds raw genetic material into an assembly line of "processing units"—represented as distinct demes within each of which breeding is confined. This process does not rely on a huge initial population to ensure an adequate supply of the raw material, as in Goldberg (1989). Even if during some initial generations, some beneficial or critical low-level structures are not available or not selected, they can appear later with the continuing inflow of random individuals from the generator, and can survive and be incorporated into higher-fitness individuals that have a good chance of surviving to influence the continuing search. One implication of this is that a search process can be begun with a relatively small number of individuals evaluated, and that, if better solutions are desired, the process can be run longer, as opposed to the usual situation: if a better quality of solution is ever to be produced, the process must be started with a much larger number of evaluations, to assure sufficient "coverage" of essential structures in the initial population.

Previous attempts to introduce random individuals into nearly converged populations to partially reinitialize them (for example, Goldberg, 1989; Whitley et al., 1991), or to increase the mutation rate dramatically when the population begins to converge (for example, Cobb & Grefenstette, 1993), have difficulty incorporating lost building blocks into highly evolved individuals, as described above in Section 4.1, especially in the case of genetic programming. What does HFC do to change this situation? In a converged population with many high-fitness individuals, the individuals are usually composed of highly coupled or co-adapted subcomponents. When these individuals undergo crossover with a random individual or are mutated strongly, the operations almost always destroy the well-evolved closely coupled relationship, reducing the fitness dramatically. So it is difficult for the highly converged (evolved) population to jump out of local optima. This can also be explained based on Holland's hypothesis that, "Neutrality decreases with each successive level" (Holland, 2000), which means that for high-fitness populations, mutation doesn't work as effectively as it does with low fitness populations. Therefore, it is better to introduce random individuals or use high mutation rates only in subpopulations that do not already contain high-fitness individuals with highly co-adapted subcomponents. Accordingly, in HFC, the newly generated random individuals are always incorporated into the low-fitness-level subpopulations, so they can survive within the fair competition environment with low selection pressure and be naturally utilized, with their beneficial genetic material moving up the assembly line only when refined to enable competitive survival at a higher fitness level.

When one examines the fitness progress curve of most GP (and also GA) experiments, the most salient observation is that the largest fitness gains typically occur in the very early stages. The evolution in the later stages appears to be more like a refining process. In tree-based genetic programming, the initial stages of evolution usually establish the general framework of the topology of the GP trees of an individual. The nodes near the tree root converge relatively quickly. Actually, the higher the fitness, the more constraints the established topology puts on possible later modifications, and the less likely become major innovations. It is not wise to effectively terminate this phase after a very limited assembly process, as the fitness of individuals rises. The phenomenon described here is very similar to Waddington's canalization of development (Waddington, 1942). Since highly evolved individuals have less and less probability of changing their basic frameworks, it is preferable to maintain, somehow, repositories of representative intermediate individuals, from which further innovation may occur.

This analysis suggests introducing a paradigm shift in the strategy for avoiding premature convergence. Instead of trying to help a highly converged population escape local attractors, or to retard its convergence, a better strategy is to allow the continual emergence of individuals that may populate new attractors, in a bottom-up manner, as is seen in HFC. This process is kept unbiased by the high-fitness regions already found in the search process.

Requirement II: The Culturing and Protection of Intermediate-Fitness Levels of Individuals

Intermediate-fitness individuals (stepping stones) at a progression of advancing fitness levels must be maintained and protected against elimination by high-fitness individuals discovered early, thereby sustaining them sufficiently to allow for their exploitation.

Goldberg (2002) stresses the importance of ensuring building block growth via recombination for design of *competent* GAs. However, based on the analysis of Observation II, it is clear that competent GAs and other EAs may suffer from insufficient population size and premature loss of intermediate building blocks, for sufficiently difficult problems. While the HFC structure, unlike that of competent GAs, does not require the existence of identifiable building blocks, for problems that have them, the HFC model, with its assembly line structure of subpopulations in successive fitness levels, provides a mechanism for continually culturing lower-level building blocks into higher-level building blocks, so long as different fitness levels implicitly correspond to different levels of building blocks embedded in the individuals belonging to those levels. Of course, this is possible only when the genetic operators somehow are able to combine building blocks rather than disrupting building blocks, as occurs with two–point crossover in shuffled HIFF problem (Watson & Pollack, 1999). Fitness is already used as the signal for differentiation of good building blocks from bad ones or building blocks of different orders in messy GA schemes (Goldberg, 2002). This is justified by the fact that high-fitness individuals usually have more strongly coupled components (like real variables, binary genes, etc.). So these groups of strongly coupled components can be regarded as higher-level building blocks. This kind of implicit building block concept – *i.e.,* not needing to explicitly identify the building blocks, but maintaining them appropriately, nonetheless – distinguishes HFC from the *competent* GAs (Goldberg, 2002) and from ADF in GP (Koza, 1994).

To reduce the "founder effect" (Holland, 2000) – that early-discovered building blocks interfere with finding and nurturing of other competing building blocks – some kind of protection mechanism must be used. This idea can be explored jointly with the concept of diversity maintenance. Niching usually uses a distance measure to spread and keep genotypically or phenotypically diverse individuals. Elitism is another mechanism sometimes used to protect good building blocks. The Species Conserving GA explicitly preserves "different and good species seeds." The Cohort GA (Holland, 2000) uses delayed-but-guaranteed reproductions to reduce the loss of good schemata.

HFC can incorporate any or all of these techniques at each level. However, HFC provides an additional mechanism to keep multiple diverse high-fitness individuals, along with their intermediate genetic material. First, for each level, there is a continual inflow of new individuals from lower levels. In this sense, HFC can be looked on as a hierarchical version of elitism, in which subpopulations at each level are the "Halls of Fame" of all lower-level subpopulations. This kind of hierarchical elitism is especially important since the fractions of crossovers and mutations that produce better offspring decrease with increasing fitness. It ensures that the products of these increasingly rarely successful crossover and mutation operations get preserved and exploited. From a diversity point of view, the whole population is highly diversified, since we have individuals ranging from the random to the highly evolved. In fact, the ratio of high- to moderate- to low-fitness individuals may be adjusted arbitrarily in HFC to control the relative rates of exploration and exploitation. This additional diversity maintenance capability is especially useful where distance criteria for niching are not available or are hard to compute. It also allows stronger selection pressure at each level without risk of premature convergence of the population as a whole. While this requirement for protection of multiple fitness levels of individuals has been described above in the building block framework of Holland and Goldberg, it does not depend on the existence of classical building blocks to be used effectively, unlike the competent GA mechanism for explicit identification of building blocks.

4.2 The Hierarchical Fair Competition Model for Sustainable Evolution

4.2.1 The Metaphor: Fair Competition Principle in Biological and Societal Systems

The HFC model was initially inspired by competition mechanisms observed in biological, societal and economic systems. Competition is widespread in these systems, and selection is sometimes very strong, but diversity remains large. In the biological world, environmental niches are organized in different levels, occupied by many types of organisms with different sizes. As Bonner pointed out, "Size is a universal property of all organisms and size niches remain constant over geological time" (Bonner, 2000). The importance of hierarchical niches is also emphasized by Tiwari, Roy, et al. (2002):

> *Speciation* leads not only to horizontal diversification of species at any given trophic level but also to vertical bio-diversity that accounts for the emergence of complex species from simpler forms of life.

This means that competition in natural evolution is essentially organized into different levels and bacteria don't compete directly with elephants. In human society, competitions are also often organized into a hierarchy of levels. Unfair competition is typically avoided – for example, a young child will not normally compete with college students in a math competition. Young individuals with potentially outstanding capabilities in specialized areas do not have to face immediate competition with the most highly developed individuals in the population, but can rise rapidly as they become more able to compete; to play an important role in future advances. After close examination, we find there is a fundamental principle underlying many types of competition in both societal and biological systems: the Fair Competition Principle. We use the educational system to illustrate this principle in more detail.

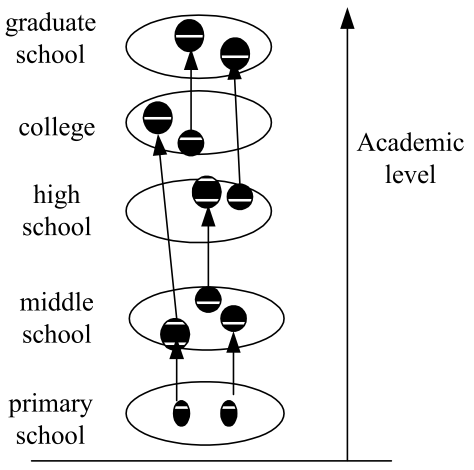


Figure 4.3 The fair competition model in educational systems of China. Low-level students compete to get admission to higher-level schools. Competition is designed to exist only among individuals of similar ability levels

In the educational system of China and many other developing countries, primary school students compete to get admission to middle schools and middle school students compete for spots in high schools. High school students compete to go to college and college students compete to go to graduate school (Figure 4.3) (in most Western countries, this competition starts at a later level, but is eventually present, nonetheless). In this hierarchically structured competition, at each level, only individuals of roughly equivalent ability will participate in any competition; *i.e.,* in such societal systems, only fair competition is allowed. This hierarchical competition system is an efficient mechanism to protect young, potentially promising individuals from unfair competition, by allowing them to survive, learn, and grow up before joining more intense levels of competition. If some individuals are "lost" in these fair competitions, they were selected against while competing fairly only against their peers. If we take the academic level as a fitness level, it means that only individuals with similar fitness can compete.

An interesting phenomenon sometimes found in societal competitions is the "child prodigy." A ten-year-old child may have some extraordinary academic ability. These prodigies may skip across several educational levels and begin to take college classes at a young age. An individual with sufficient ability (fitness) can join any level of competition. This also suggests that in subpopulation migration, we should migrate individuals according to their fitness levels, rather than according to "time in grade." To keep an individual of vastly superior fitness in a given level is to engender unfair competition.

With such a fair competition mechanism that exports high-fitness individuals to higher-levels, societal systems reduce the prevalence of unfair competition and the unhealthy dominance or disruption that might otherwise be caused by "over-achieving" individuals. It maintains relatively low selection pressure at each level while maintaining strong global selection pressure.

Similar "fair competition" is also enforced in the economic world, where a variety of regulations and laws (*e.g.,* antitrust laws) are set up by governments or international organizations to ensure fair competition and exclude domination. The system governing athletic competitions of various sorts is similar and well known. These strategies are necessary to promote healthy competition and allow new start-ups to have a chance to mature.

4.2.2 The HFC Model

Inspired by the fair competition principle and the hierarchical organization of competition within different levels in biological and societal systems, the Hierarchical Fair Competition model (HFC) is proposed for use in genetic algorithms, genetic programming, and other forms of evolutionary computation. The HFC model is composed of three interdependent components.

I. The Hierarchical Organization of Individuals to Establish a Fitness Gradient

In the HFC model (Figure 4.4), individuals are organized into a set of hierarchical fitness levels. These levels can be implemented as explicit discrete levels using separate subpopulations, as in multi-population EA, or by some speciation tags as in (Spear, 1992), or as implicit levels by some density control mechanisms as discussed in Section 4.4. In an HFC model with explicit levels, each level can have one or more subpopulations. Each fitness level accommodates individuals within a specified range of fitness, and forces emigration of individuals with fitness above that range. The entire range of possible fitnesses is spanned by the union of the fitness ranges of all levels. Each level has an admission threshold determined either initially (fixed) or adaptively. Each level also has an export fitness threshold, defined by the admission threshold of the next higher fitness level. In an HFC model with no explicit levels, some special mechanisms are needed to ensure good distribution of individuals ranging from random individuals to the best-so-far individuals, as is done in CHFC (Section 4.4).

Generally, only individuals whose fitnesses are above the admission threshold and below the export threshold of the fitness level of a given subpopulation are allowed enter, or stay, respectively, in a given level. Otherwise, they are exported to a subpopulation of the appropriate higher fitness level. This rule can be used to prevent the degradation of the individuals in higher levels potentially caused because unqualified individuals may occupy the levels. However, if other mechanisms are used to avoid the degradation issue, it is not necessary to strictly require that all individuals in a given level are qualified, as shown by QHFC in Section 4.6.

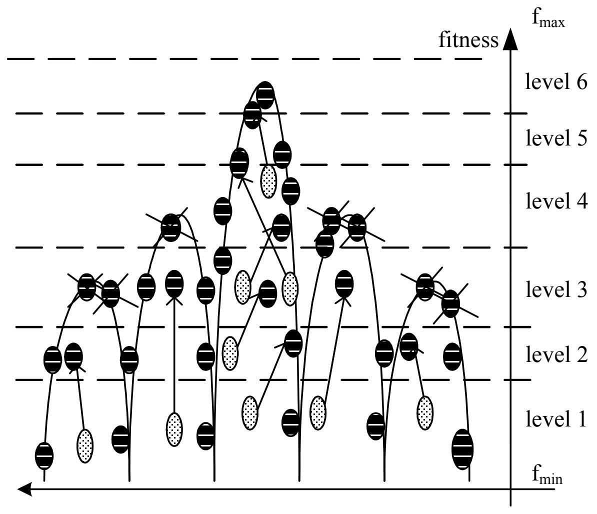


Figure 4.4 The streamlined structure of the HFC model. The HFC model extends the search horizontally in the search space and vertically in the fitness dimension and kills bad individuals at appropriate times while allowing promising young individuals to grow up continuously

A problem that can occur at any fitness level is that the children produced via mutation or crossover of individuals at a given fitness level have fitnesses below its admission threshold. This could allow the average fitness of individuals at that level to degrade below the admission threshold. This degradation problem can be dealt with in any of several ways:

1) these low-fitness offspring can be allowed to remain in the level -- reminiscent of the occasional backward step allowed in simulated annealing -- with the assumption that the selection mechanism and immigration replacement policy will act strongly enough that the average fitness remains satisfactorily above the admission level, or
2) the low-fitness offspring could be "down-migrated" to a lower-level population, but this is seen as undesirable -- it is not these low-fitness individuals that likely contain the building blocks we strive to preserve and recombine; or
3) offspring generated with fitnesses below the admission threshold could be discarded, and the operation generating them repeated (perhaps for a fixed maximum number of times). This would tend to help combine building blocks into higher-level ones; or

4) any offspring generated of lower fitness than their parent (or parents, in the case of crossover) could be discarded (and the parents kept, instead), or

5) offspring could be generated from a chosen parent (or pair, for crossover) until an individual with fitness at least as high as the parent (or the mean of the two parents) is generated (subject to a maximum number of trials).

It may appear that option 3) is the most desirable, and would not lead to systematic degradation of the fitness level of any population. However, some experiments with QHFC, discussed in Section 4.5, showed that policy 1) is computationally more efficient, so long as the lower levels can continue exporting individuals to higher levels. This kind of "fuzzy" segregation of fitness levels can also increase the diversity of the population.

II. Random Individual Generator: the Source of Genetic Material

As illustrated in Figure 4.5, at the bottom fitness level, there exists a random individual generator that feeds "raw" genetic material (in the form of individuals) continuously into the bottom processing level. It is beneficial if this generator is made to be unbiased such that it is able to supply a complete set of all possible low-level building blocks, unless there is prior knowledge about the search space that should be used to bias the generator. Of course, new building blocks may be discovered by recombination and other genetic operations and this unbiased condition is not necessary. The inflow of random individuals relieves HFC from depending on a large population size to provide and preserve enough different genetic material at the outset to allow thorough search of the problem space in a single "epoch", as is typical for existing GAs (Goldberg, 2002).

III. The Migration Policy from Lower to Higher Fitness Levels

HFC model requires that there exists a continuous flow of genetic materials from lower levels to higher levels, either according to some fixed breeding and migration strategies or some adaptive control mechanisms. Exchange of individuals between levels can be conducted synchronously at a certain interval or asynchronously and adaptively, as in many parallel EA models.

At each moment of exchange, or, if desired, as each new individual in a subpopulation is evaluated, any individual whose fitness qualifies it for a higher level is exported to the target higher level which has a fitness range that accommodates the individual. After the exporting process of a level, this level will import an appropriate number of candidates to fill those openings either from their admission buffers where qualified individuals are stored or from lower levels. If subpopulations at the base level find any open spaces left over from exporting, they fill those spots with random individuals. If subpopulations of higher levels find empty spaces after importing individuals from their admission buffers or if their admission buffers are empty, they can either mutate current members or select two members and do crossover to generate the needed number of new individuals, or they can import individuals in the subpopulation from the level below, even though they do not meet the admission criterion. (Note: the bottom level subpopulations import from the random individual generator).

In HFC methods, migration of individuals was allowed in only one direction, from lower-fitness levels to higher-fitness levels, but migration is not confined to only the immediately higher level. The motivation is to avoid domination in lower levels by degraded individuals whose “backbones” are the same with some individuals of higher levels. Also, any "degraded" individuals generated in a population of a given fitness level were not "exiled" to a lower level, even though they may not have met the admission criterion for migrating into the level—that is, method 1) above was used. This may allow for promising jumps with short-term fitness degradation in the fitness landscape. It can also be justified in that, despite the possible decrease of the fitness, many useful intermediate-level building blocks may still remain.

For HFC model without explicit levels, the migration policy is replaced by some other control mechanism to ensure the continuous supply of genetic material for lower levels to higher levels as discussed in CHFC of Section 4.4.

4.2.3 The HFC Principles for Sustainable Evolutionary Search

It is clear that there are many strategies to design a sustainable evolutionary algorithm based on HFC model. The major principles of HFC can be summarized as the following:

a) To ensure fair competition among individuals by segregating competition among different levels of individuals,

In this way, high-fitness individuals won't threaten the existence of those with lower-level fitness, as in most EAs. Extensive experimental study in Burke et al. (2002) suggested that a phenotypic diversity measure is superior to genotypic ones in predicting the run performance. This suggests that the straightforward and thorough phenotype-based protection of "inferior individuals" by HFC may be more reliable than other techniques such as fitness sharing or the species conserving algorithm SCGA (Li, et al., 2002), in the sense that it doesn't require a distance measure which is hard to find in genetic programming, and it supports the continuing search for new frameworks at low-levels.

b) To assure that exploration and exploitation occur at all fitness levels

Allocating computing cycles to demes of all fitness levels allows innovation to occur all the way from embryonic primitive individuals to highly evolved individuals. By cascading higher-level beyond lower-level fitness demes, a building block assembly line is established. In analogy to the Cambrian Age, HFC essentially keeps the emergence of new "species" operating, thereby enhancing the capability for sustainable evolution in genetic programming. It also greatly relieves the conventional demand for large population sizes to achieve good performance, as will be shown in the following experimental section.

c) To provide hierarchical elitism (one-way elitism) and hierarchical niching

In the multi-population HFC, when the fitness of an offspring is lower than the fitness admission threshold of the current HFC level, the offspring will be discarded. Otherwise, it continues to stay in the current fitness level or is exported to a higher level, if appropriate. This appears as a one-way migration from lower-level demes to higher-level demes. In the context of genetic programming, since higher-fitness individuals usually have higher complexity, the elitism here can effectively remove those individuals that have unnecessary complexity for their fitness level. This mechanism may help to control the bloating phenomenon in genetic programming. In HFC model, it is important to ensure that none of the levels get converged. Then it is beneficial to employ some kind of niching mechanism at each level to ensure the diversity and thus productiveness of all levels to make the flow of individuals from lower level to higher level go on continuingly.

d) To incorporate new genetic material continually to provide a constant influx of evolutionary potential

A random individual generator is configured at the bottom of the HFC deme hierarchy to provide inflow of unbiased genetic material to higher fitness levels. From the bottom level to the highest fitness level, this convergent evolutionary process will never deplete its evolutionary potential. Instead, it provides a mechanism to allow innovation to happen continually at all fitness levels.

4.2.4 Designing Sustainable Evolutionary Algorithms Based on HFC

Designing a sustainable evolutionary algorithm based on HFC principle is not constrained with any kind of structure mentioned above, including the multi-level hierarchy of the metaphor. As illustrated below, sustainable evolutionary search can be achieved if only the HFC principle can be guaranteed—the segregation of the competition of high-fitness individuals and low-fitness individuals and continuing breeding of both high-level and low-level individuals and the continuing promotion of new genetic material from the lower levels.

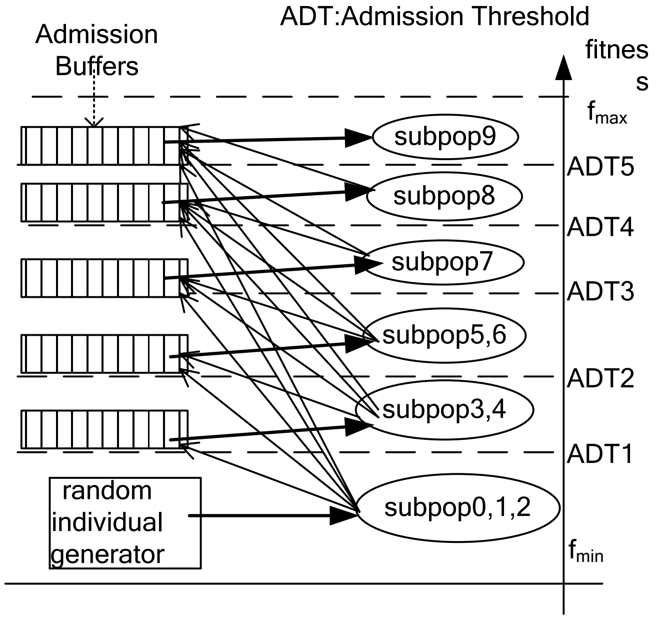


Figure 4.5 The organizational structure of the generational HFC model. In this model, subpopulations are organized in a hierarchy by ascending fitness level. Each level (with one or more subpopulations) accommodates individuals within a certain fitness range determined by its admission threshold. The admission buffers are volatile, existing only during migration. $f_{\min}$ is determined at first K_c generations, $f_{\max}$ is either specified by users or adjusted dynamically.

4.3 Generational HFC Algorithms: Static and Adaptive

4.3.1 The Motivation & Design Rationale

The first HFC algorithm is implemented straightforwardly as a generational multi-population evolutionary algorithm (Figure 4.5). According to its parameter setting and adjustment of the subpopulation structure, it can be classified as static HFC where all parameters and topology of sub-populations are fixed and adaptive HFC with dynamic parameters and topologies of subpopulations to let the algorithm adapt to the properties of problems.

In the generational HFC algorithm, the entire population is divided into several fitness levels. Each level contains one or more subpopulations. Each level has an admission fitness threshold and an export fitness threshold, which is equal to the admission fitness threshold of its next higher level. The admission threshold of the bottom level $f_{\min}$ is determined at the end of K_c generations and then fixed later on. Admission thresholds of higher levels are simply determined by evenly distributing the fitness range between the maximal possible fitness value and the minimal fitness $f_{\min}$. Since for many problems, we can map the absolute fitness range into standard fitness range [0, 1], so this static HFC is still applicable to many problems. In the case that this mapping is not available, the adaptive HFC algorithm is needed. In this algorithm, the admission thresholds of bottom level is determined as static HFC, but the admission thresholds of higher levels is first determined by evenly distributing the fitness range between the current maximal fitness plus a standard deviation of the top level and the minimal fitness $f_{\min}$, at the end of K_c generations. After that, the admission thresholds of all levels except the bottom level are adjusted with the increase of the best fitness.

The migration policy in generational HFC here is relatively simple. After every K_m generations, a synchronous exchange process is evoked, during which all super-qualified individuals at lower levels (whose fitness is larger than the export fitness of its current level) are exported to their corresponding qualified levels. Here a qualified level is a level whose fitness range accommodates the fitness of the individual. To facilitate the migration process, each level is allocated an admission buffer. At the starting of migration, all super-qualified individuals first migrate out to the corresponding admission buffers. Then starting from the top level to bottom level, each level will admit its candidates to first fill the openings left over by the emigrants, then replace the degraded unqualified individuals, then randomly replace one old individuals in the current level. If the current level still has some openings, a mutation or crossover operation is evoked to select parent from current level and generate the needed number of individuals.

Between the migration points, the degraded individuals of a given level are simply kept there, thus the policy 1) in Section 4.2.2 (I) is employed. So it does not discard any offspring or demote them to lower-fitness subpopulations.

One problem with static HFC algorithm here is that at the beginning, there is no or few qualified individuals at higher levels and we still want to use a simple one-way migration policy from lower levels to higher levels, so the we simply switch off the breeding at a higher level if it does not contain any qualified individuals. But this will reduce the effective population size for the early evolution stage. To solve this dilemma, one or more *floating subpopulations* are introduced whose dynamic admission thresholds are continually reset to the admission threshold of the level in which the current best individual has been found. Thus, these subpopulations provide additional search in the vicinity of the advancing frontier in the fitness hierarchy (see Figure 4.6). In this scheme, it is reasonable not to start the evolution process in higher-level subpopulations until some minimum number of immigrants above the admission threshold have entered them.

4.3.2 The Algorithm Framework

I. The static generational HFC algorithm

In this version of HFC algorithm, the user needs to determine the number of fitness level L, the number of subpopulations N_P, the population size of each subpopulation $|P_i|$, the belonging relationships of subpopulations to fitness levels, the admission thresholds of each fitness level f_{adm}^{l}, the migration intervals K_m. Note that the admission fitness of bottom level f_{adm}^{0} is always negative infinity to admit any random individual (we only discuss maximization problems in the thesis). All the admission thresholds are determined based on some initial exploration of the fitness landscape of the problem, such as the range of the fitness or distribution of early-discovered peaks.

There is a *Lazy Admission Setting* to simplify the parameter setting of static HFC. In this approach, the export threshold of bottom level or f_{adm}^{1} is simply calculated as f_{μ}, the average fitness of the whole population after K_c generations, which are called calibration stage. And as we know the adjusted maximal fitness $f_{\max}$ of this problem (usually 1), then the admission thresholds of other levels can be determined by equally distributing the fitness range between f_{μ} and $f_{\max}-\sigma_f$ to all levels except bottom levels as formula (1), where the σ_f is the standard deviation of the fitness of all individuals in the whole population at the end of calibration stage.

$$f_{adm}^{i} = f_u + (i-1).(f_{\max} - \sigma_f - f_u)/(L-2) \quad i = 1,...,L-1 \tag{4.1}$$

Table 4.1 gives the static generational HFC algorithm procedure

Table 4.1 Static generational HFC algorithm

```
Input parameters:
parameters for standard multi-population EA, including pCrossover, pMutation (We assume here using one
set of parameters for all subpopulations, though they can be different)
$N_P$, number of subpopulations
$|P_i|$, population size for each subpopulation
$K_m$, migration interval
$L$, Number of levels of the hierarchy
$f_{\max}$, the maximal fitness value
$f^l_{adm}$, $l = 1, ..., N-1$, the admission fitness of all fitness levels
( or $f^l_{adm}$ can be set by formula (4.1) and the procedure described above (4.1)
  $K_c$ calibration generations

1. Initialization
initialize the subpopulations, set the admission thresholds
$gen \leftarrow 1$:
2. Do
if Lazy Admission Setting approach is used to set admission threshod {
  if $gen < K_c$ (in calibration stage)
      run EA without exchange
  else if $gen = K_c$ (calibration stage ends)
      determine the admission thresholds for each level by formula (4.1)
}
breed all subpopulations
$gen \leftarrow gen + 1$ }
if($gen \% K_m$ =0) //do migration
    Do for each subpopulation from bottom level to top level
        Examine fitness of each individual, if over-qualifed, export to admission buffer of corresponding higher
        level whose fitness range accommodates this exported individual
    end do
    Do for each subpopulation from top level to bottom level
        while the admission buffer of current level is not empty
            pick out candidates from the admission buffer to fill the openings left over by emmigrants
        while the buffer is not empty
            continue to import candidates to replace the unqualified individuals
        while the buffer is not empty
            continue to import candidates to replace random old individuals in the subpopulation
    end do
}
until the stopping criterion is satisfied.
return the highest-fitness individual(s) from the highest-level subpopulation
end procedure
```

II. The generational HFC algorithm with adaptive admission threshold (HFC-ADM)

There are several difficulties with the above HFC algorithm with static admission thresholds. One is that we have to assume the maximal fitness is known to map the absolute fitness range into [0, 1], while for many problems this $f_{\max}$ is not known. Even it is known, to evenly divide the whole fitness range into L levels while most of high levels do not have any qualified individuals will lead to the failure to segregate the competition between high fitness individuals and low fitness individuals, since they are all compressed to the low levels before higher level individuals are found. This will break the fundamental principle of HFC for sustainable evolution. To solve this problem, we introduce a simple dynamic admission threshold adaptation mechanism (HFC-ADM) to set the admission thresholds continuously as the fitness of the best individual improve constantly.

In HFC-ADM, there is a calibration stage as the Lazy Admission Setting mentioned in the above section: at the end of K_c generations, the average fitness of the whole population f_μ , the standard deviation σ_f of fitnesses of all individuals, the best fitness of the whole population $f_{\max}$ are calculated. Then the initial admission thresholds are determined as follows:

$$f_{adm}^{0} = -\infty \qquad (4.2)$$

$$f_{adm}^{1} = f_\mu \qquad (4.3)$$

$$f_{adm}^{L-1} = f_{\max} - \sigma_f \qquad (4.4)$$

Admission thresholds of other L-2 fitness levels are determined by (4.1). Here, the basic idea is to start from maximum fitness value, first set $[f_{\max,}\ f_{\max} - \sigma_f]$ as the fitness range of the top level, set $[-\infty, f_\mu]$ as the fitness range of base level, and then allocate the fitness range of $[f_\mu, f_{\max} - \sigma_f - f_\mu]$ equally to the other $L-2$ levels.

However, it is clear that as the evolutionary search goes on, higher-fitness individuals are continually discovered that ruin the stratification developed by the admission thresholds determined at the initial calibration stage. So a dynamic admission threshold updating mechanism is needed. After each K_u generations, the maximal fitness, f_{max}, and the fitness standard deviation of the top-level subpopulations, σ_f, are recomputed to reset the admission thresholds of all the fitness levels except the base level and the first level, by (4.1) and (4.4). To maintain some momentum and to avoid dramatic variation of the best fitness, we use the *AdaptationRate* to decide by how much to change the current admission thresholds:

$$f_{adm}^{i,new} \leftarrow (1 - AdaptationRate) \times f_{adm}^{i} + AdaptationRate \times f_{adm}^{i,\exp} \qquad (4.5)$$

where $f_{adm}^{i,new}$ is the updated admission thresholds for level i, f_{adm}^{i} is the previous old admission threshold of level i, $f_{adm}^{i,\exp}$ is the expected new admission threshold for level i according to (4.1). The idea is that it is better to maintain smoothness of the admission update with respect to the increase of $f_{\max}$.

Table 4.2 is the algorithmic procedure of HFC-ADM

III. The generational HFC algorithm Adaptive Migration Topology (HFC-ATP)

In Section 4.3.1, we mention the difficulty of HFC with static admission thresholds and we have to use a somewhat ad hoc "floating subpopulation" to maintain sufficient effective population size during the early stage. Although the floating subpopulation can be used to enhance the search effort on the fitness frontier, the computational capability of higher-level subpopulations is largely wasted before their activation. This reduces the effective population size. One possible solution to this difficulty is to allow two-way migration. Then individuals of all levels could be evaluated and unqualified low-fitness individuals in high-level subpopulations could be moved to lower levels; however, this likely still yields ineffective search at the higher-level subpopulations, as at most a few distinct individuals at those levels survive and remain in those subpopulations early in the evolution process.

Table 4.2 The generational HFC algorithm with adaptive admission thresholds (HFC-ADM)

```
Input parameters:
parameters for standard multi-population EA, including  pCrossover, pMutation (We assume here
using one set of parameters for all subpopulations, though they can be different)
$N_p$, number of subpopulations
$|P_i|$, population size for each subpopulation
$K_m$, migration interval
$L$, Number of levels of the hierarchy
$K_c$ calibration generations
$K_u$ admission thresholds update interval (generations)

1. Initialization
initialize the subpopulations, set the admission thresholds
$gen \leftarrow 1$:
2. Do
if $gen < K_c$ (in calibration stage)
      run EA without exchange
else if $gen = K_c$ (calibration stage ends)
      determine the admission thresholds for each level by formula (4.1-4.4)
breed all subpopulations
$gen \leftarrow gen + 1$

if $gen \% K_u = 0$ {
      update admission thresholds of Level 2 to L-1 according to (4.1), (4.4), (4.5)
}

if $gen \% K_m = 0$ {//do migration
   Do for each subpopulation from bottom level to top level
       Examine fitness of each individual, if over-qualifed, export to admission buffer of corresponding
       higher level whose fitness range accommodates this exported individual
   end do
   Do for each subpopulation from top level to bottom level
        while the admission buffer of current level is not empty
              pick out candidates from the admission buffer to fill the openings left over by emmigrants
        while the buffer is not empty
              continue to import candidates to replace the unqualified individuals
        while the buffer is not empty
              continue to import candidates to replace random old individuals in the subpopulation
   end do
}
until the stopping criterion is satisfied.
return the highest-fitness individual(s) from the highest-level subpopulation
end procedure
```

Actually, we can use a smarter approach, namely the adaptive migration topology method to ensure the effective population size to maximally be utilized. In the beginning, all subpopulations are allocated to the bottom level. Later, once certain higher levels get some qualified individuals, then all intermediate levels are activated and appropriate number of subpopulations is allocated to those newly activated levels and are filled by admitting qualified individuals or importing lower level individuals from the next lower level or randomly generated if current level is the base level. This "HFC-ATP" algorithm works like a rubber band. At the initial stage, it is quite compressed, but gradually, the rubber band stretches to accommodate individuals with a larger range of fitnesses. In this paper, subpopulations are allocated as follows: firstly, all subpopulations are allocated to the bottom level. After the calibration stage, subpopulations are then evenly allocated to each activated levels. Extra subpopulations can be allocated from highest level to bottom level (if aggressive exploitation is desired) or from bottom level to top level (if intensive exploration is desired). The HFC-ATP and "floating subpopulation" are compared in Figure 4.6

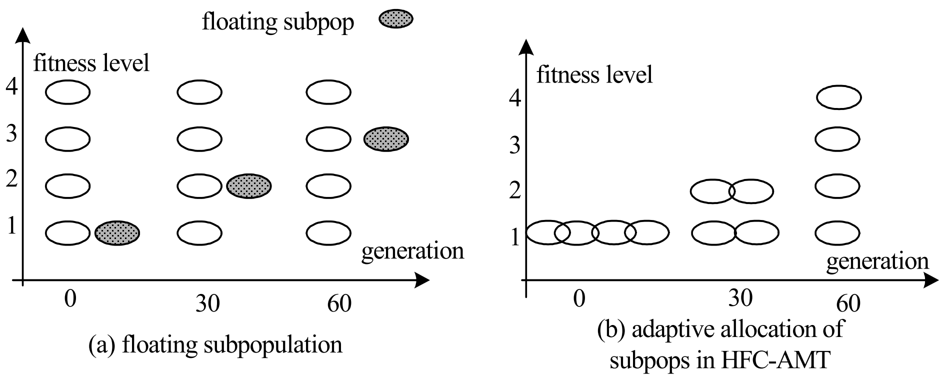


Figure 4.6 Floating subpopulation and adaptive allocation of subpopulations to fitness levels

4.3.3 Test Suites

To evaluate the performance of HFC algorithms and compare them to traditional techniques to sustain evolutionary search, a set of genetic programming problems are used here. One is a standard GP benchmark problem: the even-N-parity problem (Poli et al. 2000). Another is a real-world bond graph synthesis problem, the eigenvalue placement problem described in Section 3.3.1.

I. The even-10-parity genetic programming benchmark problem

As a Boolean function induction problem, the task of the even-10-parity problem is to evolve a function with 10 binary inputs and one binary output such that the output of the function is 1 (true) if and only if an even number of the inputs evaluated to be true. The difficulty of this problem depends on the function set exploited and the order (n) of the target function. Since even-n-parity problems with the standard function set {OR, AND, NOR, NAND} provide little gradient information for incremental search and are deemed an inappropriate benchmark problem for GP (Poli & Page, 2000), we prefer to use the following function set {OR, AND, NOT, XOR} here. The inclusion of the XOR function provides a means for GP to make a succession of improvements.

II. The eigenvalue placement problem

The eigenvalue placement problem defined in Section 3.5.1 is used here as a benchmark problem. One has to define the embryo bond graph, the function and terminal sets, and the target eigenvalues. The embryo bond graph used here is shown in Figure 4.7. The function set is the same as Table 3.3.

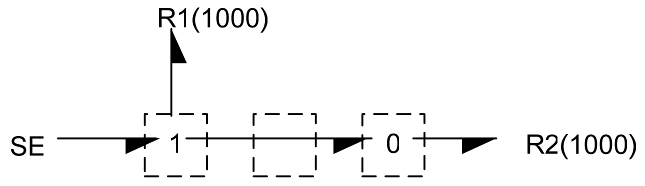


Figure 4.7 Embryo bond graph in eigenvalue placement problem

4.3.4 Experimental Results

I. Experimental setting for even-10-parity genetic programming benchmark problem

In this experiment, five algorithms are evaluated on the even-10-parity problem, all using the function set mentioned above. The algorithms are labeled single population (OnePop), multi-population (MulPop), HFC with floating subpopulation (HFC), HFC with adaptive admission threshold determination mechanism (HFC-ADM), and HFC with adaptive allocation of subpopulations (HFC-ATP). Four (total) population sizes (150, 250, 400, 800) are tested for all five algorithms, each with 60 runs to allow reaching statistical significance. The shared parameters for all the experiments are summarized in Table 4.3. Since there is a random individual generator at the bottom level to provide new genetic material, mutation was not used here. Note that a certain degree of elitism is enforced for OnePop and MulPop through the reproduction operator with best selection operator. This removes the possibility that HFC provides superior performance only by virtue of elitism. For the MulPop algorithm, the whole population is evenly divided into 10 sub-populations, arranged in a ring topology, as is commonly done. The migration interval is set as 5 generations. The number of individuals (K) to be migrated is about 1%, (K = 2, 3, 5, and 8 for population sizes of 150, 250, 400, and 800, respectively). The migration strategy for MulPop is to select the K best individuals from the donor subpopulation and copy them (without removal from the donor) to replace the K worst individuals in the receiving subpopulation. The parameters specific to the HFC technique and its adaptive variants are summarized in Table 4.4.

The determination of fitness admission thresholds for HFC and HFC-ATP was very straightforward. The maximum fitness of the even-10-parity problem is 1024, and the average fitness of the first 5 generations is typically around 550, so the fitness range of [550, 1024] was evenly allocated to all fitness levels. As will be shown by the performance of HFC-ADM with adaptive admission threshold determination, HFC techniques are not generally very sensitive to these parameters. The allocation of subpopulation sizes was decided based on a rough balance between exploring the fitness frontier and maintaining a supply of intermediate-fitness building blocks or stepping stones. Generally, more individuals were allocated to higher fitness levels. For HFC and HFC-ADM, the floating subpopulation size should be big enough for effective exploration at the highest currently activated fitness level. All of the parameters in Table 4.4 were set before these runs were made, without tuning them for each specific problem instance.

Table 4.3 Shared parameters for even-10-parity problem

max_evaluations	300,000
Init_method	half_and_half
Init_depth	4-7
max_nodes	300
max_depth	10
crossover with tournament selection (size=7)	pCrossover: 0.95
reproduction with best selection (elitism)	pReproduction: 0.05

Table 4.4 Parameter setting for HFC algorithms

	HFC	HFC-ADM	HFC-ATP
	Exchange frequency: 5 Fitness levels: 9		
	Admission thresholds from level 1 to 9: -100, 550, 600, 650, 700, 750, 800, 850, 900	Threshold adjustment Interval: 5 nCalibrateGen : 5 AdaptationRate: 0.95	Admission thresholds from level 1 to 9: -100, 550, 600, 650, 700, 750, 800, 850, 900
	subpopulation i belongs to fitness level i for i<10, subpopulation 10 is the floating subpop.		
Popsize=150	Subpop size for subpop 1 to subpop 9: 10 For floating subpop10: 60		Subpop size for subpops 1 to 10: 10,10,10,10,10,15,15,20,20,30

Popsize=250	Subpop size for subpops 1 to 6: 15
	for subpops 7 to 9: 20
	for floating subpop10: 100
			Subpop size for subpops 1 to 10: 15,15,15,20,20,25,25,25,40,50

Popsize=400	Subpop size for subpops 1 to 9: 20, 20, 20, 20, 30, 30, 30, 40, 40 for floating subpop10: 150		Subpop size for subpops 1 to 10: 20,20,30,30,35,35,40,40, 50, 100
Popsize=800	Subpop size for subpop 1 to subpop 9: 30, 30, 30, 30, 40, 40, 40, 60, 160 for floating subpop10: 400		Subpop size for subpop 1 to subpop 10: 40,40,40, 60, 70,70, 10,80,100,200

II. Results for even-10-parity problem

The average best raw fitness of run for each algorithm was tabulated for a given number of evaluations performed. In the case of the even-10-parity problem, since the perfect solution is known, the success rate of each algorithm within 300,000 evaluations was also measured. It is impressive that, according to Figure 4.8, all three HFC techniques consistently outperformed single population and simple multi-population GP for all the population sizes. The success rates almost doubled for population sizes 150 and 250 and were also much higher for population sizes 400 and 800. Considering the little additional computing effort of HFC to organize the individuals, this significant improvement might be surprising. The streamlined supply of intermediate building blocks provided by the assembly line of multiple fitness levels provided a mechanism for continuing search without getting stuck at local optima. The continuing framework also greatly reduced the requirement for large population sizes in GP.

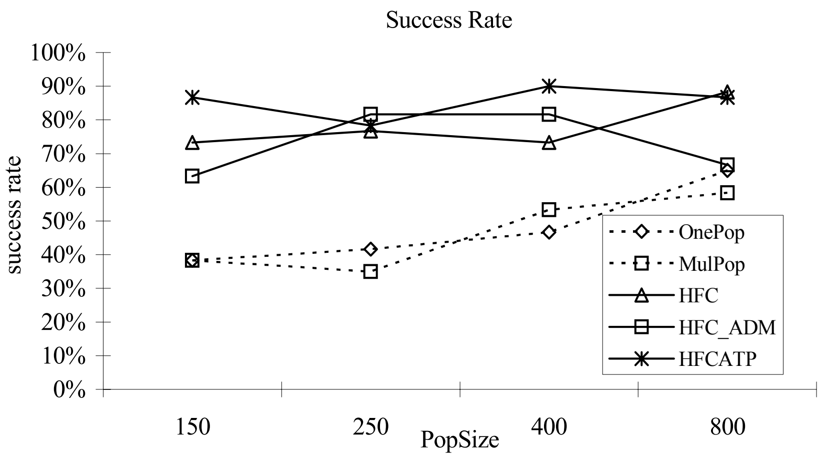


Figure 4.8 Comparison of success rates after 300,000 evaluations HFC techniques consistently outperform standard OnePop and MulPop GP and are essentially invariant with respect to population size.

The results in Figure 4.8 and 4.9 showed that the HFC techniques are very insensitive to the total population size, once a minimum size requirement is met. On the other hand, traditional EAs depend strongly on large population sizes to allow them to find good results before convergence occurs. This is just as predicted by the analysis in Section 4.1.1. In this respect, HFC changes the convergent nature of the existing EA framework into continuing search.

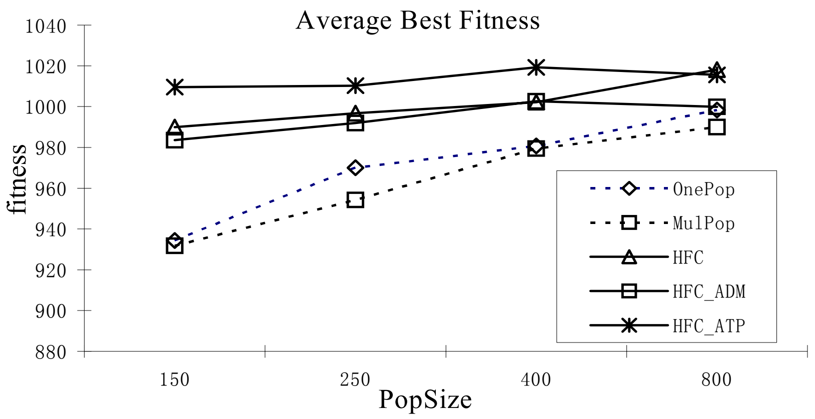


Figure 4.9 Comparison of average best-of-run fitness for HFC-GP and standard GP. With given population size and evaluations, HFC techniques consistently achieve better average best fitness, reflecting their robustness of search.

To compare search efficiency, the average number of evaluations used by each method in finding of the 20 earliest-found perfect solutions was computed, as shown in Figure 4.9. (The minimal number of perfect solutions found by all the methods was 20.). For this relatively easy even-parity problem, it shows that traditional EAs may get good results more quickly but taking a higher risk of premature convergence, while HFCs do not incur much of a penalty in terms of speed (especially for HFC-ATP, in this case) in finding the optimal solution and exhibit much more robustness in finding optimal solutions (Figure 4.8). For difficult problems, the "hasty" conventional EAs will have much less chance to make such quick progress.

III.Experimental setting for eigenvalue placement problem

The embryo bond graph used in the experiments is illustrated in Figure 4.7. It has three modifiable sites for further development. The 8-eigenvalue target is set in Table 4.5. The parameter settings for the five algorithms are the same as for the even-10-parity problem in above subsection I except for the changes noted in Table 4.5. For each experiment, 40 runs were conducted, with 300,000 evaluations in each.

Table 4.5 Parameter setting for 8-eigenvalue placement problem

8 eigenvalue targets:
{-0.1±5.0j, -1±2j, -2±j, -3±0.7j}

total population size=500
max_nodes = 400
max_depth = 12
Admission thresholds of 9 fitness levels for HFC and HFC-ATP:
-100, 0.55, 0.60, 0.65, 0.70, 0.75, 0.80, 0.85, 0.90
Subpopulation sizes for subpops 1 to 10 for HFC and HFC-ADM:
25,25,25,25, 30,30,30,30,30,250

IV. Results for eigenvalue placement problem

With this complex real-world synthesis problem, the continuing search capability of the HFC model is expected to enable obtaining much better results without the premature convergence usually found in experiments with standard GP techniques. The best-of-run distance errors from the target eigenvalues are averaged for 40 runs. The standard deviation of the best distance errors is also calculated (Table 4.6). The simulation results strongly support the claims of the preceding sections. The HFC-ATP algorithm again achieves almost half the average distance error with much smaller variance. This kind of robustness is enabled by the continuing search capability of HFC.

Table 4.6 Comparison of average best-of-run distance error for 8-eigenvalue problem

	OnePop	MulPop	HFC	HFC-ADM	HFC-ATP
Mean Distance Error	0.598	0.64	0.458	0.407	0.278
σ	0.287	0.363	0.19	0.228	0.15

To evaluate the relative performance of the five approaches, multiple two-tail t-tests are applied. The result is summarized in Table 4.7.

Table 4.7 The t-test results for four GP algorithms to an 8-eigenvalue placement problem: the difference between the standard GP and the HFC techniques is significant at the 1.2% level, while the difference between OnePop and MulPop is not significant.

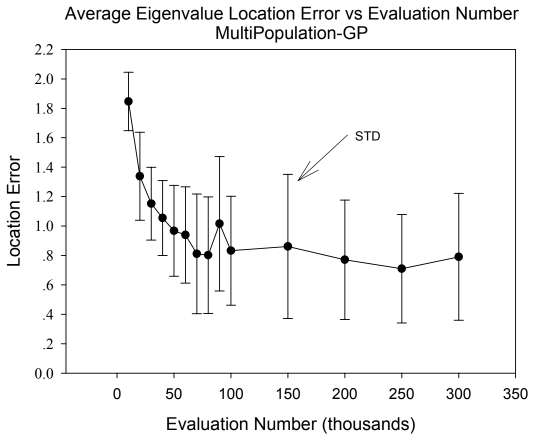
t-test	MulPop	HFC	HFC-ADM	HFC-ATP
OnePop	0.68	0.012	0.0015	5.8E-09
MulPop		0.0069	0.0010	3.8E-07
HFC			0.28	1.2E-05
HFC-ADM				0.0040

Table 4.8 Comparison of stagnation time for the 8-eigenvalue placement problem. In the HFC model, last progress is made much closer to the evaluation limit of 300,000, and it displays a much smaller tendency to converge.

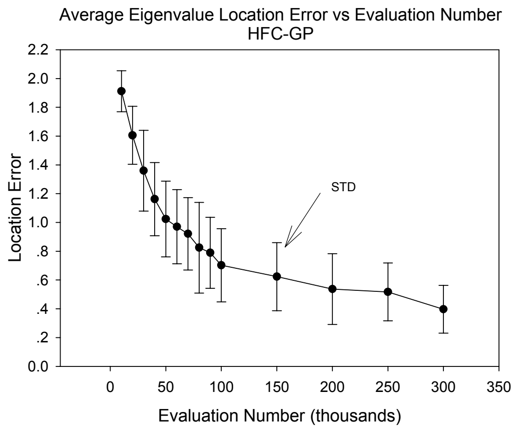
	OnePop	MulPop	HFC	HFC-ADM	HFC-ATP
Step at which last progress is made	232,300	204,300	262,500	283,500	283,100
σ	64,300	70,700	55,200	18,500	34,900

Another criterion useful to compare the continuing search capability with other methods is to compute the average evaluation number at which the last progress is made in runs of a fixed number (300,000) of evaluation steps. The time from this last progress step to evaluation 300,000, when that time becomes large, reflects likely "stagnation" of the search, and is expected to increase (*i.e.,* the last progress step is earlier) when premature convergence occurs. Table 4.8 shows that the HFC technique continues to make progress through evaluation numbers much closer to 300,000, which means that steady progress is achieved throughout the run. In fact, as the step at which last progress is made approaches 300,000, the evidence that the run has stagnated at all disappears. On the other hand, the standard GP approaches have stagnated (made no progress) for 70,000 to 100,000 evaluations before evaluation 300,000, on average.

Using the same parameter settings for a more difficult 10-eigenvalue problem {-0.1±5.0j, -1±2j, -2±j, -3±0.7j, -4±0.4j}, we calculate the evaluation numbers of last progress for 16 runs of MulPop and HFC-ATP, each run with 400,000 evaluations. The difference is even more striking – 263,400 for MulPop with standard deviation 106,639, but 392,320 with HFC-ATP with standard deviation 12,534. This result clearly demonstrates that for this more difficult problem, HFC techniques can achieve much more robust search and continue to make steady progress.



(a) Multi-Population GP



(b) HFC-ATP

Figure 4.10 Comparison of the average best-of-run location errors vs. number of evaluations for multi-population-GP and HFC-ATP. Error of 0 is the optimal value. HFC achieves much more robust search with continuous progress, reflected by its much smaller standard deviations. Multi-population-GP is less reliable, displaying large standard deviations of the location error. It also has the tendency that beyond 300 K evaluations, there won't be much progress, while for HFC-ATP, sustainable progress appears.

The robustness and the continuing search capability are also examined by investigating the relationship between the average of the best eigenvalue location errors over 40 runs and the maximum number of evaluations, ranging from 10,000 to 300,000. Here HFC-ATP is compared only to MulPop. Experiments for each maximum evaluation limit are run with different random seeds. The experimental parameters are the same as in the experiment described above except that for HFC, the subpopulation sizes for subpops 1 to 10 are {30,30,40,40,50,50,50,50,60,100}. For MulPop, the migration frequency is every 5 generations, with 10% migration of individuals at that interval, and with a copying-type migration (migrants appear in new population, but are not removed from the donor population). The results are presented in Figure 4.10. The robustness of HFC is demonstrated by its much lower standard deviation of location errors. The sustainable search capability is shown by its continuing progress given more evaluations, while for MulPop, after a certain threshold number of evaluations, no progress is made in any reasonable number of additional evaluations.

4.4 CHFC: HFC Algorithm with Single Population, the Continuous HFC

There are already three versions of HFC algorithms, namely the static generational HFC, HFC with adaptive admission thresholds, and HFC with adaptive migration topology. So why another one?

4.4.1 The Motivation & Design Rationale

While the generational HFC algorithms with multi-population can achieve sustainable search, it is complained that they are too complex to set parameters. And with the complex dynamics among levels, it is also hard to do theoretical analysis. In these algorithms, the whole fitness range is divided into discrete segments, each with a set of separate demes. It has the advantage of simplicity and efficiency. But the division is somehow arbitrary. In Section 4.3, the HFC model with adaptive admission fitness demonstrated that it is not necessary to find a set of exact admission thresholds to allow fair competition since the performance is not very sensitive to the admission thresholds. An extension of this idea is to completely remove the discrete segmentation of the demes and to use a single population to do hierarchical fair competition by introducing some special mechanisms to implement the HFC principle. This makes HFC easier to incorporate into many existing EA packages. As a complementary element, the continuous single-population HFC can also be applied to each subpopulation of the original HFC model, which is a naturally parallel model, or to any other parallel evolutionary algorithm.

The idea here is to extract the basic ideas of the fair competition principle of HFC and apply it to single population evolution algorithm, such that it becomes more widely and readily usable in existing GP/EA packages. Briefly, we introduce three mechanisms, called a) hierarchical elitism, b) explicit control of the breeding probability distribution over the fitness range, and c) explicit control of the distribution of the individuals over the fitness range, in order to achieve sustainable evolution for single population genetic programming and other EAs. It is also demonstrated that fair competition can also be achieved by other means other than the multi-population based segregation.

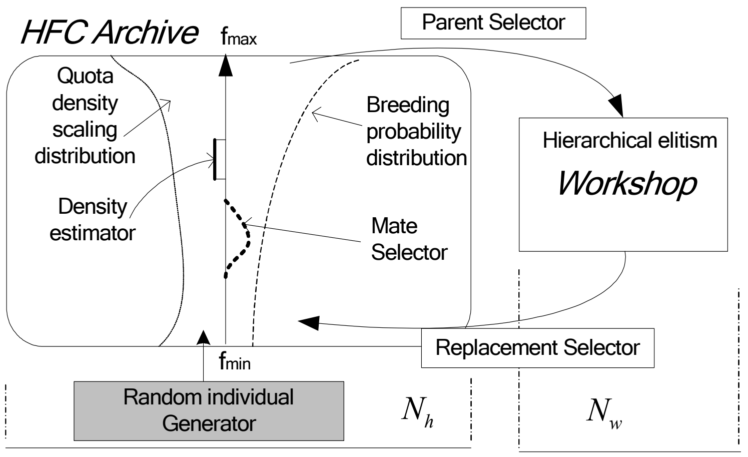


Figure 4.11 Structure of the continuous HFC model. N_h is the population size of the HFC archive. N_w is the workshop size

4.4.2 The Algorithm Framework

The Continuous HFC (CHFC) is a steady-state EA model. It consists of a set of components as illustrated in Figure 4.11, including an HFC archive population, a "workshop" population, a random individual generator, a parent selector, a replacement selector, a breeding probability distribution control mechanism, and a removal control mechanism with a density estimator (determining what individuals, of what fitnesses, should be removed when new individuals enter the archive population at their specified fitness levels). We first give an overview of the algorithm in Table 4.9, and then introduce details of each component.

Table 4.9 The continuous HFC (CHFC) algorithm

(1) Determine the following parameters besides other GP parameters:

HFC archive size, N_h

workshop size, N_w

Parameters for GP mutation and crossover operators

Breeding probability distribution and its parameter (here, based on an exponential distribution)

Density quota scaling distribution and its parameter (here we based it on an exponential distribution with the parameter λ and the maximum ratio K_m)

Pair selection method and its parameters (here, a random neighbor method with $p_{neighborhood} = 0.1$)

(2) Initialize the HFC archive randomly, finding the maximum and minimum fitnesses. Sort the individuals in the population by ascending fitness. The workshop deme is initially empty.

(3)Loop until termination

- According to breeding probability distribution $p_b(x)$, generate a value τ in [0, 1].
- Compute the target sampling fitness $f_t = f_{\min} + \tau\left(f_{\max} - f_{\min}\right)$ according the GP operator probabilities, decide on a current operation

If current operation is mutation, select an individual with index I in the population whose fitness is nearest to the sampling fitness; if current operation is crossover, select one parent as for mutation, then use the pair selector to select the second individual. Create one (if mutation) or two (if crossover) offspring and evaluate their fitnesses. If an offspring has a fitness value greater than one of the parents, save it in the workshop deme; else discard this offspring.

Check if the workshop deme is full. If full, update the HFC archive with the new offspring in the workshop deme; else continue at top of loop.

HFC Archive Update Procedure

Calculate the normalized density of each individual in the HFC archive according to Eq. 3.8 below. For each individual in the workshop deme, place it in the HFC archive according to its fitness. Also, determine the individual with largest normalized density value in the archive, the "victim," and remove it. The exception is that if the victim is among the top K_e (elite group size) fitness genotypes present in the population, it is kept, and another victim is removed.

End procedure

Explanation of the Continuous HFC model

The continuous HFC model is composed of the following components as shown in Figure 4.11:

a) Two populations for storage of individuals (only one of which is persistent)

One of them is the HFC "archive" used to accommodate the breeding candidates, and works like a "normal population" in an EA. The other, "workshop," deme is used to hold temporarily the offspring generated during one generation. By varying the size of workshop, we can control the generation gap to be any desired value.

b) A random individual generator

This generator continuously inserts some unbiased random genetic material into the HFC archive. It is the source of the fundamental genetic diversity.

c) A parent selector

This selection operator is used to select breeding parents according to the breeding probability distribution for recombination or mutation. The breeding probability is introduced to ensure sufficient selection pressure to exploit early-discovered high-fitness individuals, analogous to the allocation of larger subpopulation sizes to higher HFC levels (Hu & Goodman, 2002). Rather than using the fitness uniform selection operator (Hutter, 2002), a higher probability is typically allocated to higher fitness individuals, although that is under user control. However, these higher fitness individuals do not dominate the population as in a conventional EA, because of the individual allocation control mechanism used to prevent uncontrolled growth of high-fitness individuals.

The breeding probability distribution can be linear, exponential or any other distribution; typically, one is chosen that favors higher fitness individuals for breeding. Figure 4.12 lists some typical distributions; Fitness Uniform Selection (Hutter, 2002) uses the last, uniform, distribution.

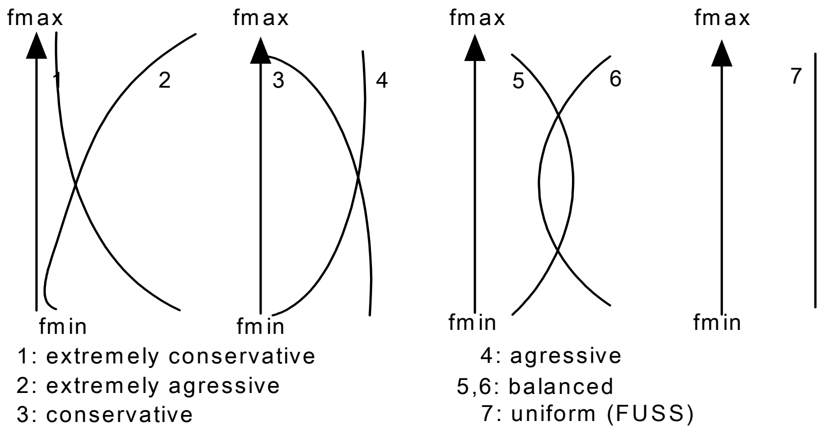


Figure 4.12 Seven types of distribution of the breeding probability of all fitness levels

In our experiments, the following (truncated, reversed exponential) distribution was used:

$$p_b(x) = \frac{\lambda e^{-\lambda(1-x)}}{1-e^{-\lambda}} \quad for \;\; x \in [0, 1] \qquad (4.6)$$

The parameter λ is a parameter to control the selection pressure or the aggressiveness of exploitation vs. exploration.

First a random value *r* between [0, 1] is generated according to the distribution $p_b(x)$; then the target sampling fitness is calculated as

$$f_t = f_{\min} + r * (f_{\max} - f_{\min}) \qquad (4.7)$$

After determining f_t, the individual *I* whose fitness is closest to f_t is determined.

The pair selection method for crossover can be implemented in the following ways:

According to the fair competition principle, the second selection is dependent on the first one and should be selected from the vicinity of the first selected individual. This is called the Random Neighbor approach. There are two ways to define the neighborhood of an individual (using the implicit level concept of HFC): 1) an individual (mate) with an index within *K/2* steps (offset of index number in the sorted population) away from the first individual may be randomly selected,

$$[I - \frac{K}{2} * \lambda(f_t),\ I + \frac{K}{2} * \lambda(f_t)] \qquad (3.3)$$

where K is neighborhood size, $\lambda(f_t)$ is a scaling parameter to allow different neighborhood sizes in different fitness levels. (In this paper, $\lambda(f_t)$ is set to 1 for simplicity.)

Or, 2) the entire fitness range may be divided into L levels, and the second individual (mate) may be chosen from the set of individuals with fitnesses in the range

$$[f_t - \frac{f_{\max} - f_{\min}}{2L} * \lambda(f_t),\ f_t + \frac{f_{\max} - f_{\min}}{2L} * \lambda(f_t)], \qquad (4.8)$$

using a percentage of the HFC archive size to decide the L value

$$L = p_{neighborsize} * N_h \qquad (4.9)$$

where N_h is the HFC archive size.

We have also tested the other two approaches used in FUSS (Hutter 2002) (to be reported elsewhere), the *Next Nearest* approach, where the second individual is the one whose fitness is second closest to the sampling fitness, and *Independent FUSS*, where we select the second individual independently, just as we selected the first individual.

d) A replacement selector

This selection operator decides which individuals in the HFC archive are to be replaced by the new offspring. This is based on the density estimator and the quota density scaling distribution.

e) A density estimator

This component is used to estimate the density around each individual, with additional scaling by the quota density scaling distribution. By adjusting the scaling distribution, we can achieve any desired proportion of individuals at the various fitness levels. Here, the relative density concept is used. That is, we evaluate by what percentage the current density surpasses its nominal (quota) density at any point on the quota density scaling distribution. The individual with the highest percentage becomes the next individual to be replaced.

The relative density in the neighborhood of an individual can be estimated with either of two approaches:

Approach 1: Select two individuals from the sorted (ascending fitness) HFC archive with indices within K/2 steps of current individual I. The raw density is estimated as

$$D_{raw}(I) = f(I + \frac{K}{2}) - f(I - \frac{K}{2}) \tag{4.10}$$

where $D_{raw}(I)$ is called the raw fitness density of an individual (the implicit denominator, K, is constant, so is ignored).

The raw nominal (quota) density of an individual $D_{rawq}(I)$ is computed as:

$$D_{rawq} = \frac{K}{N_h} * (f_{\max} - f_{\min}) \tag{4.11}$$

where N_h is the population size of the HFC archive. Finally, the normalized density is computed as the relative density:

$$D_{norm} = \frac{D_{raw}(I) - D_{rawq} * r_d}{D_{rawq} * r_d} \tag{4.12}$$

where r_d is a scaling parameter to be described below.

Approach 2: Assume the whole fitness range is divided into L levels, and count the number of individuals (M) whose fitnesses are in the range

$$[f(I) - \frac{f_{\max} - f_{\min}}{2L},\ f(I) + \frac{f_{\max} - f_{\min}}{2L}] \tag{4.13}$$

The raw density of an individual I is $D_{raw}(I) = M$

The raw nominal quota density is:

$$D_{rawq} = \frac{N_h}{L} \tag{4.14}$$

where N_h is population size of the HFC archive

The normalized density is then calculated as (4.12)

We use a percentage of the HFC archive size to decide the above L, where $L = p_{neighborsize} * N_h$, and N_h is the HFC archive size.

The parameter r_d in (3.12) is used to bias the allocation of individuals over different fitness levels. It is generated according to a user-specified quota density scaling distribution, which determines the relative size of high-fitness individuals and low-fitness ones. It is important to maintain sufficient high-fitness individuals for effective search. For different problems, this distribution can also have different types, as illustrated in Figure 4.12. In our experiments, the following distribution was used.

$$r_d(x) = \frac{a + \lambda e^{-\lambda x}}{a + 1 - e^{-\lambda}} \quad where \ a = \frac{(e^{-\lambda} - K_m)}{K_m - 1}\lambda \qquad (4.15)$$

where λ is used to control the shape of the exponential distribution. K_m is a user-specified maximum ratio of $r_d(1)/r_d(0)$. It is used to control the maximum ratio of individuals that are allowed between the highest fitness level and the lowest fitness level.

f) Hierarchical elitism

This mechanism is enforced during the insertion of offspring into the workshop. When the fitness of the offspring is less than that of both parents, the offspring is discarded (not used to update the HFC archive).

4.4.3 Test Suite

The basic motivation of HFC is to ensure sustainable search by avoiding premature convergence. It is most useful in the case of difficult problems where local optima are of great concern. Here the Santa Fe trail artificial ant problem (Koza, 1994) is used to illustrate how CHFC can improve the performance of genetic programming, dealing with the bloating problem and exerting control over the fitness distribution of the population.

The artificial ant problem involves evolving a sequence of movements in a two-dimensional grid to collect as much food (distributed on the grid) as possible. We used the Santa Fe Trail, with a size of 32*32. The maximum amount of food is 89. We defined the fitness as

$$fitness = \frac{1}{1 + \frac{\text{max food} - \text{collected food}}{\text{maxfood}}} \tag{4.16}$$

4.4.4 Experimental Results

We compared the performance of CHFC with both steady-state GP and generational GP. The parameters are listed in Table4.10.

Table 4.10 Parameters used in the various GP experiments on Santa Fe Trail

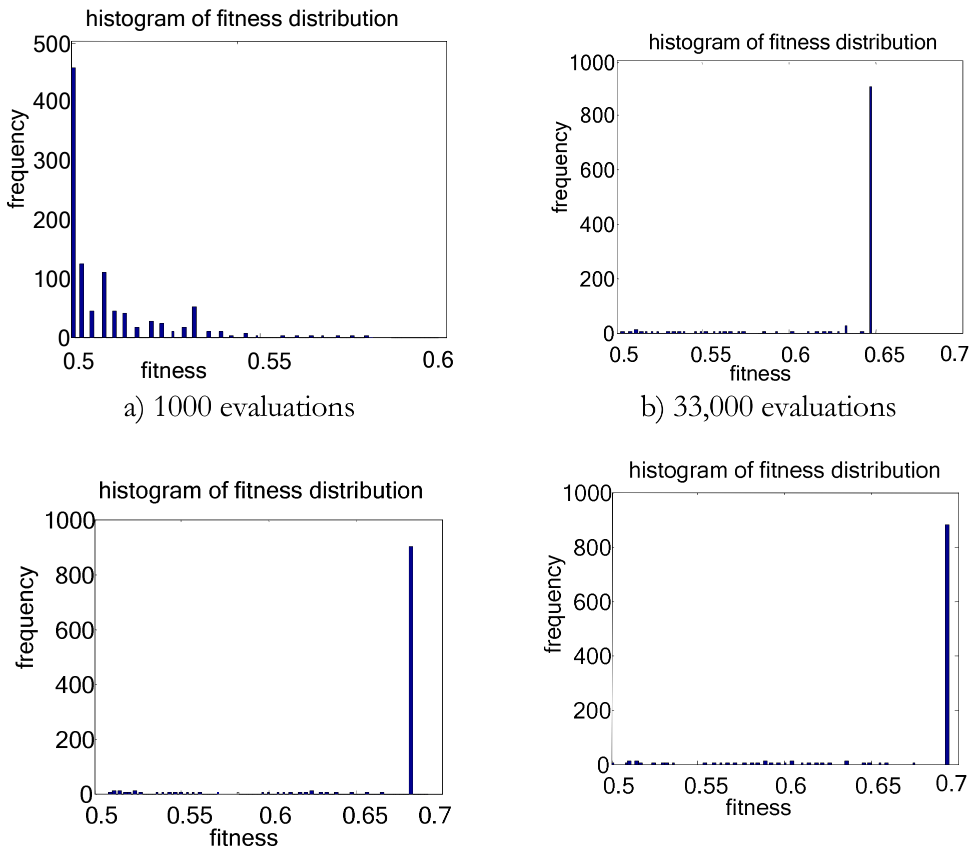
Common GP Parameters	Additional CHFC parameter
Function and terminal set	workshop size: 20
{Forward, Right, Left, Prog2, Prog3, IfFoodAhead}	Breeding probability distribution type:
Max evaluation: 100000	exponential (lambda=4.0)
Population size: 1000	Density scaling distribution:
Tournament size: 2 (for steady-state and generational GP)	exponential (lambda=4.0, maxRatio=10)
Crossover rate: 0.9	Elite group size K_e =1
Shrink mutation: 0.05	Probability of introducing random
Point mutation: 0.05	individuals each generation: 0.2
Maxtree depth: 17 mutate maxdepth: 5	Percentage generated randomly in that
Init max depth: 6 Init min depth: 2	event: 0.05

Table 4.11 Peformance comparison of CHFC and conventional GP in Santa Fe Trail artificial ant problem

Algorithm	Best mean fitness	Std. dev.
CHFC-GP	0.92	0.05
Steady state GP	0.76	0.073
Generational GP	0.81	0.089

Twenty experiments were run for each algorithm. The experimental results are summarized in Table 4.11. To gain insight into the evolutionary process, we drew four histograms showing the progression of the fitness distribution of the population during the run. (Since steady-state GP has similar behavior, only generational GP is compared to CHFC-GP). Figure 4.13 and

Figure 4.14 show the fitness distributions of generational-GP and CHFC-GP for the ant problem. It is clear that conventional genetic programming suffers a lot from the domination of early-discovered high-fitness individuals while CHFC, through its controlling mechanism based on fair competition, can always maintain a balanced distribution of both high-fitness individuals for exploitation and low-fitness individuals for "Cambrian innovation"—the innovation of the basic framework.



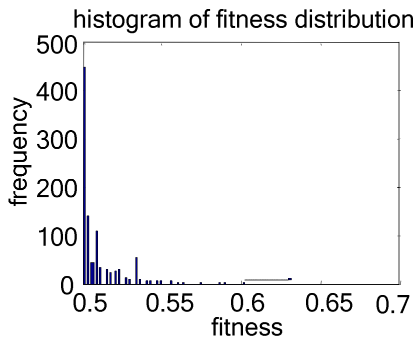
a) 1000 evaluations

b) 33,000 evaluations

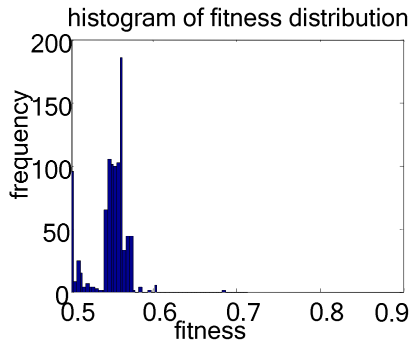
c) 66, 000 evaluations

d) 100,000 evaluations

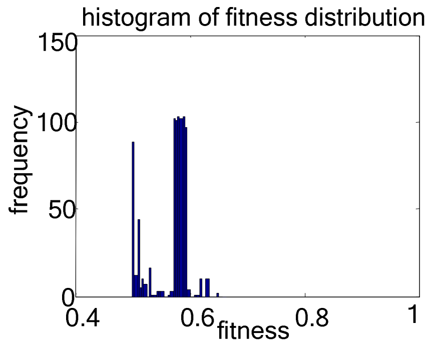
Figure 4.13 The convergent nature of conventional generational GP. The high-fitness individuals rapidly dominate the population of this typical run. The subplots were prepared by sampling at evaluation numbers 1000, 33000, 66000, and 100000.



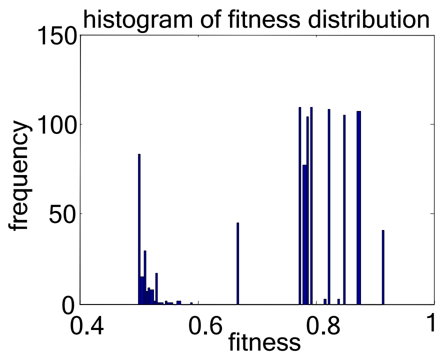
a) 1000 evaluations



b) 33,000 evaluations



c) 66,000 evaluations



d) 100,000 evaluations

Figure 4.14 Evolution of the individual distribution in the dimension of fitness of CHFC. Thanks to the breeding probability control and the density control mechanism, CHFC can maintain a balanced distribution of high-fitness and low fitness individuals without any risk of domination in this typical r un, evidently without sacrificing its exploration of the high-fitness range. Subplots are prepared at evaluation numbers 1000, 33000, 66000, 100000

Figure 4.15 plots the fitness growth process of the three algorithms. It is impressive that CHFC, by avoiding premature convergence, continues to generate innovative (and superior) solutions when the steady-state and generational GP have stagnated. These experiments demonstrated that HFC principle as a generic sustainable evolution model can be implemented in a variety of ways in different application contexts.

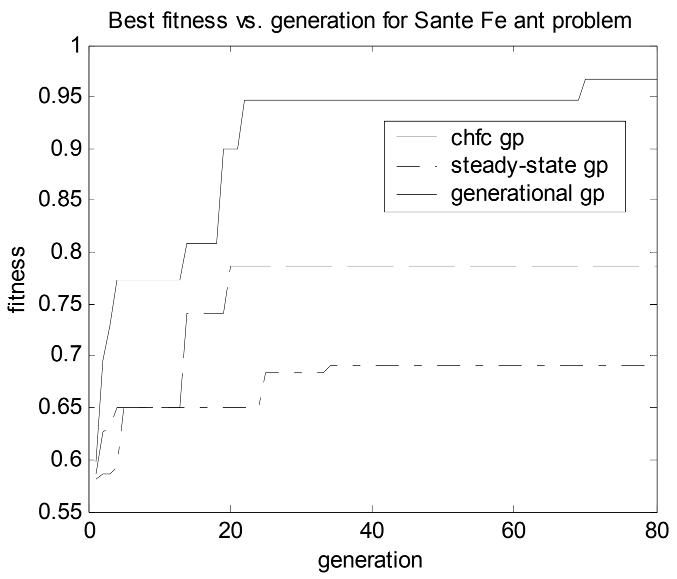


Figure 4.15 Performance comparison of CHFC with steady-state and generational GP. By avoiding the local optima that trapped the other runs, CHFC makes consistent progress and discovers improved solutions

4.5 HEMO: HFC Algorithm for Multi-objective Search

4.5.1 The Motivation & Design Rationale

Multi-objective evolutionary algorithms are very important in practice. As pointed out in (Tiwari et al., 2002), the three primary features of real-life engineering design optimization problems: multiple objectives, multiple interacting variables and constraints. These features usually make the search space hostile for evolutionary search and premature convergence is a prominent issue in practice. However, existing state-of-art evolutionary multi-objective optimization algorithms inherit the convergent nature from classical evolutionary computation models and there are few researches in the EMO community explicitly addressing this issue. In addition, some of the advanced EMO algorithms which can handle multi-modal problem to some extent invariably require the existence of distance function, while it is already demonstrated that genotypic or phenotypic distance function based techniques doesn't work effective in genetic programming (Burke & Gustafson, 2002). So how to improve the sustainability of multi-objective genetic programming poses a real challenge as we try to incorporate multi-objective GP into topology synthesis.

To achieve a sustainable multi-objective search for Pareto optimal solution set, two main issues need to be taken care of. One is the diversity of the Pareto front; another is the sustainable lateral diversity of the intermediate solutions. Figure 4.16 illustrates how a state-of-art EMO algorithm, NSGA-II fails to find the true Pareto front because of lack of the sustained lateral diversity. Although multi-objective evolutionary algorithms have some inherent tendency to increase the diversity because of the selection pressure of diversified multi-dimension objectives, the fact that the objective values in all dimensions increase constantly implies that it will gradually lose its exploratory capability in the long run, just as the traditional single objective EA does.

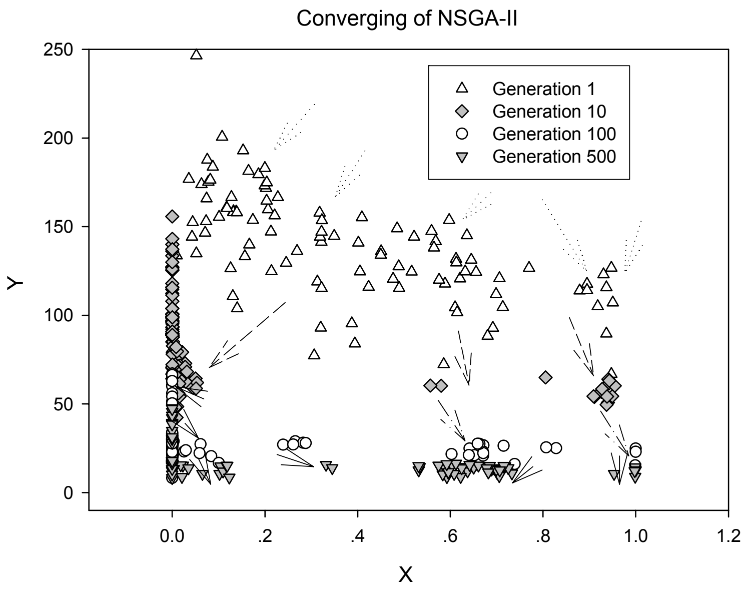


Figure 4.16 Evolution pattern of individual distribution in NSGA-II. The population of NSGA-II moves in clusters leaving the initial low objective value space and converging to the promising space. Even the maintenance of a predefined proportion of population into all fronts, but in the whole these fronts are converging to local areas, thus making it incapable to maintain the explorative capability in the long run.

To apply the sustainable evolutionary computational model to multi-objective problem, one of the most important decisions is how to segregate the competition among high fitness individuals and new lower fitness individuals as now the fitness is multi-dimensional. We can simply divide them by any single objective values. The solution proposed here is to use the average ranks of individuals. Since our previous experiments in single-objective EAs show that the HFC model is not sensitive to the exact segregation scheme, the coarse separation by ranks should be enough to ensure a sustainable multi-objective search.

By combining features from PESA and SPEA and extending the ideas in the NSGA-II with controlled elitism, and including the HFC organization, HEMO is proposed for sustainable evolutionary search for difficult multi-modal real-world multi-objective problems in which premature convergence is of great concern. We expect that HEMO will be especially advantageous in multi-objective genetic programming, where the highly multi-modal and discrete fitness landscape and the lack of good distance function often makes modern MOEAs such as PESA fail by converging prematurely to local Pareto fronts.

4.5.2 The Algorithm Framework

Based on the analysis of the fundamental cause of premature convergence and drawing ingenious ideas from previous successful MOGAs, we propose the HEMO framework for difficult multi-objective problems in which the avoidance of premature convergence is of great concern. Essentially, it is an extension of PESA enhanced with the continuing search capability of HFC. In addition to the Pareto archive and the Pareto workshop population, a succession of archives for maintaining individuals of different fitness levels is added to allow mixing of lower- and intermediate-level building blocks. A random individual generator is located at the bottom to feed raw genetic material into this building block mixing machine continually. The structure of HEMO is illustrated in Figure 4.17 and the algorithm described in Table 4.12.

Table 4.12 The HFC algorithm for multi-objective search (HEMO)

1) Initialization

- Determine the number of levels (*nLevel*) into which to divide the objective space for each objective dimension. Determine the grid divisions (*nGrid*) as in PESA. Note that nLevel is different from *nGrid*. The first one is used to organize intermediate individuals into the hierarchical archives, while the latter is used to estimate the density of individuals.
- Determine the population sizes of the Pareto archive, HFC archive and corresponding workshop demes. The distribution of population sizes among archives (workshop demes) can be determined separately or using some special distribution scheme like the geometric distribution in (Deb and Goel, 2000).
- Initialize the workshop demes with random individuals. The archives are empty at the beginning.
- Evaluate all individuals and calculate the crowding factor of each individual according to the hyper-grid approach in PESA.
- Calculate the fitness range of each objective dimension for all individuals in the whole population:
  $[f^i_{\min}, f^i_{\max}]$ where $i = 0,...,ObjDim - 1$
- Divide the fitness range into nLevel Levels. For all individuals, calculate the objective ranks for each objective dimension, $r_i, i = 0,...,ObjDim - 1$, $r_i \in [0, nLevel - 1]$;
- For each individual, calculate its fitness rank = the average rank over all objective dimensions of each individual. $r^f = \frac{1}{ObjDim} \sum_i^{ObjDim-1} r_i$
- Migrate (move out) individuals in the workshop demes to the corresponding HFC archives according to their fitness ranks r^f. Then add all non-dominated individuals of each workshop deme to the Pareto archive. There are two cases possible during these migrations. If the target archive is full, we will replace a selected individual according to the Pareto archive and HFC archive update procedures described below; else, we simply add the migrating individual into the target repository.

2) Loop until meeting stopping criterion

A steady state evolutionary model is used in the HEMO framework. First,

- Compute the breeding probability of each workshop deme of the HFC rank levels. This is calculated as follows:

$$pBreed^l = \frac{Popsize\ of\ workshop\ deme\ of\ level\ l}{\sum_{k=1}^{nLevel-1} Popsize\ of\ workshop\ deme\ of\ level\ k}$$

  These probabilities can instead be dynamically adjusted irrespective of the workshop deme sizes. These probabilities determine the allocation of search effort to each level, thus determining the greediness of the algorithm.
- Decide whether to do Pareto workshop breeding or HFC workshop deme breeding by probability $pParetoBreed$. If setting $pParetoBreed = 1$, then HFC-PESA reduces to an algorithm similar to PESA. This parameter is used to control the greediness of the Pareto search.

If Pareto workshop breeding is to be done:

- Decide whether or not to do crossover according to its probability. Mutate each gene of the offspring with probability pGeneMutate.

- Select parents from the Pareto archive using tournament selection based on the crowding factors of individuals. The less crowded, the more chance an individual will get selected. When selecting parents for crossover or mutation, the probability to select only from the Pareto archive is *pSelectFromPareto*. The probability to select a second parent from the rank 0 HFC Archive is 1- *pSelectFromPareto*. When there is only one individual in the Pareto archive, the second parent for crossover is selected from the highest HFC archive.
- Create an offspring (two in crossover) and add it to the Pareto workshop deme. If the Pareto workshop deme is not full, simply add the new candidate to it; else, trigger the **Pareto Archive Update** Procedure. Then a migration process will move individuals of each HFC archive to their new qualified HFC archives because of the update of the objective ranges.

If HFC workshop deme breeding is to be done:

- Decide at which level (L) breeding will occur according to the probability $pBreed^{l}$
- Decide whether or not to do crossover according to its probability. Mutate each gene of the offspring with probability *pGeneMutate*.
- Select parents from the HFC archive of level L by tournament selection based on the crowding factors. The lower the crowding factor, the higher the probability to be selected. If there is only one parent in the current HFC archive, then the second parent will be selected from the next lower archive.
- Create an offspring (two in crossover) and add it to the workshop deme. If the workshop deme is not full, simply add to the end; else, trigger the HFC Archive **Update Procedure** and the **Pareto Archive Update Procedure**.

▪ With low probability *pRandomImport*, update *perRandomIn* percent of the individuals of the lowest HFC archive with random individuals.

Pareto Archive Update Procedure ( )

▪ Screen out the non-dominated individuals in the workshop deme.
▪ Update the objective ranges of the whole population with the non-dominated individuals.
▪ Recalculate the crowding factors of all individuals of the selected non-dominated individuals and the individuals in the Pareto archive.
▪ Update the Pareto archive with the selected non-dominated individuals. If the Pareto archive is full, truncate it by removing individuals with higher crowding factors.
▪ Empty the Pareto workshop deme.

HFC Archive Update Procedure ( )

▪ Update the objective ranges of the whole population and recalculate the fitness ranks of all individuals in the workshop demes.
▪ Migrate individuals in the current HFC archives into their corresponding new levels. If the target HFC archive is full, replace an individual selected by tournament selection. The more offspring an individual produces, the higher the probability it will be replaced.
▪ Update the HFC archives with the individuals in the workshop deme. If the target HFC archive is full, replace an individual selected by tournament selection. The bigger the crowding factor is, the higher probability it will have to be replaced. Note that only higher archives are updated with the current workshop deme (uni-directional migration policy)

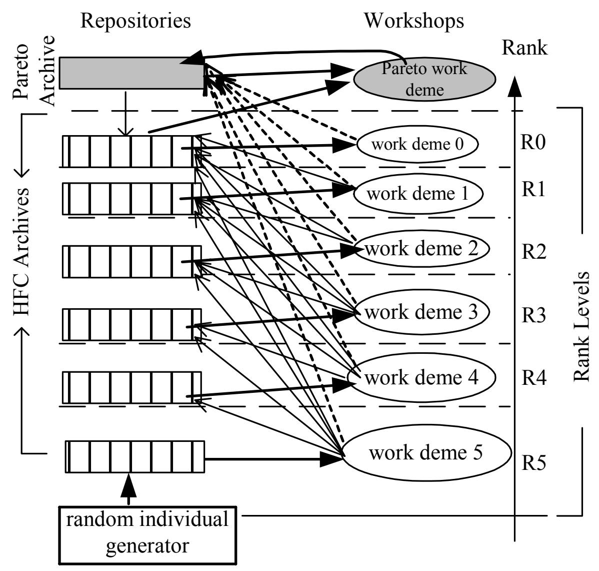


Figure 4.17 The assembly line structure of HEMO Framework. In HEMO, repositories are organized in a hierarchy with ascending fitness level (or rank level in the objective space as employed in this paper). Each level accommodates individuals within a certain fitness range (or belong to a given rank level) determined by the admission criteria.

4.5.3 Test Suites

Two test functions are selected to demonstrate the exploratory capability of HEMO to avoid premature convergence. Here, HEMO is only compared to PESA, since HEMO is most closely derived from PESA.

1) Multi-objective Rastrigin's problem (ZDT4)

$$ZDT4:\begin{cases} \text{Minimize } f_1(x) = x_1 \\ \text{Minimize } f_2(x) = g(x)[1-\sqrt{x_1 / g(x)}] \\ g(x) = 91 + \sum_{i=2}^{10}[x_i^2 - 10\cos(4\pi x_i)] \\ x_1 \in [0,1],\ x_i \in [-5,5],\ i = 2,...,10. \end{cases}$$

2) Multiobjective Griewangk Problem (GWK)

$$g(x) = 2 + \sum_{i=2}^{10} x_i^2 / 4000 - \prod_{i=2}^{10} \cos(x_i / \sqrt{i}),\ x_1 \in [0,1],$$

$$where \quad x_i \in [-512, 511]\ i = 2,...,10$$

GWK problem is constructed by replacing g (x) in 1) with Greiwangk's function, where

4.5.4 Experimental Results

Figure 4.18 illustrates the distribution of individuals of HEMO during the evolutionary process. It shows that HEMO works by trying to expand the individuals of its repositories evenly in the objective space, rather than by converging to the early-discovered high-fitness areas. This provides the necessary fitness gradient for new optima to emerge in a bottom-up way, from the bottom level HFC archive and workshop subpopulations.

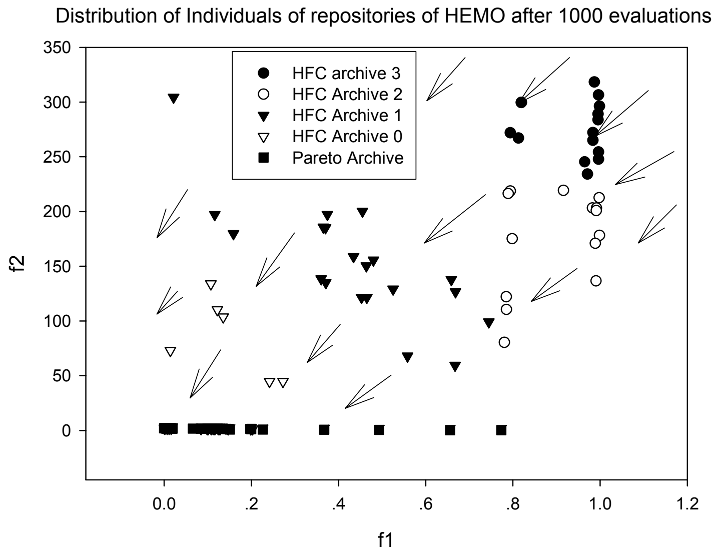


Figure 4.18 Distribution of individuals over objective space of GWK in HEMO after 1000 evaluations. Compared with Figure 4.18 of NSGA-II, the difference is that the archive population of NSGA-II is drifting and have the risk to converge to local area, while HEMO will never converge by trying to maintain representative individuals all around in the objective space and continuously introduce new genetic materials, thus providing the fitness gradient for new optima emerge in a bottom-up way

The robustness of PESA and that of HEMO are compared by examining the relationship of performance and the mutation rates applied to each type double gene after crossover. We use the statistical comparison method of (Corne et al., 2000) to compare the Pareto front obtained with different mutation rates for both PESA and HEMO (Table 4.13). The upright triangle holds comparison results of different mutation rates for HEMO, while the bottom left (shaded cells) are for PESA. From (Deb & Goel, 2001), we know that for test function ZDT4, NSGA-II fails to find the true Pareto front. This is also the case for PESA, as illustrated in the first column. PESA without mutation is worse than any PESA configuration with mutation. It is also suggestive that for PESA, the performance varies greatly with different mutation rates, achieving best performance here with a mutation rate of 0.12. It is very sensitive to the mutation rate, of which the optimal value is hard to know in advance. In contrast, HEMO is more robust over mutation rates. The performance difference with no mutation is not much different from that of mutation rate 0.16 since it doesn't depend on the mutation to maintain the explorative capability. The continual introduction of random individuals from the bottom archive level provides the explorative capability.

Table 4.13 Comparison of the robustness of PESA and HEMO with test function ZDT4. The first row of each cell represents the percentage of unbeaten Pareto front of the compared mutation rate pair, the first one for the smaller mutation rate, and the latter one for the larger mutation rate.

Mutation Rate	0.00	0.04	0.08	0.12	0.16
0.00		99 100 0 1	99.5 100 0 0.5	99.7 100 0 0.3	97.5 100 0 2.5
0.04	50 100 0 50		99.8 100 0 0.2	99.9 100 0 0.1	0.5 100 0 99.5
0.8	7.3 100 0 92.7	99.7 100 0 0.3		100 100 0 0	2.4 99.7 0.3 7.6
0.12	3.2 100 0 96.8	50.0 100 0 50.0	70.6 100 0 29.4		2.4 99.9 0.1 97.6
0.16	7.3 100 0 92.7	50.1 100 0 49.9	100 8.7 1.3 0	100 100 0 0	

We also compared the best performance of PESA (mutation rate 0.12) with that of HEMO (mutation rate 0.16) for the same number of 10,000 evaluations (Table 4.13). For ZDT4, the Pareto front found by HEMO was much better than PESA found. In the case of GWK, HEMO had limited advantage over PESA. The reason is that the statistical comparison procedure used here compares the merged Pareto fronts found during 20 runs. PESA with different random seeds may converge to different points in the objective space, which on the whole comprise a good Pareto front. However, the PESA is a poor opportunist in the sense that for both the ZDT4 and GWK functions, PESA converges to only one or two Pareto solutions in 6 or 7 runs of a total of 20. In contrast, HEMO always obtains diversified solutions in the Pareto archive.

Table 4.14 Opportunistic PESA and robust HEMO. HEMO obtains a much better Pareto front for ZDT4 and a small advantage for GWK. However, for each independent run, the probability for PESA to converge to one or two Pareto solutions is around 33%, while HEMO seldom converges to local Pareto fronts

Test Function	ZDT4		GWK	
	PESA	HEMO	PESA	HEMO
Test results	0.3 **0**	**100** 9.7	47.3 **46.3**	**53.7** 52.7
Premature Convergence Probability	6/20	0.0	7/20	0.0

4.6 QHFC: Quick HFC Algorithm with Adaptive Breeding Mechanisms

As a generic framework for continuing evolutionary computation, HFC is representation-independent and does not rely on the building block hypothesis as stated in Holland's GA theory (2000). Actually, there is no established schema theory or even consensus on the definition of building blocks in the GP community (O'Reilly & Oppacher, 1995; Rosca, 1997). The sustainable search capability enabled by HFC on the GP problems above with variable-length, highly nonlinear interaction and coupling of program chunks demonstrates the effectiveness of HFC to ensure continuing search. The question that naturally arises is whether HFC will work well in the (typically) simpler GA domain—particularly with a binary representation, where extensive theoretical studies and effective techniques exist and clear comparisons can be made. For example, how does HFC compare with other diversity-maintenance techniques in GA? Is it possible to combine HFC with other methods to produce a better sustainable search algorithm?

This section describes how the hierarchical fair competition principle can be used to design a surprisingly simple, effective and robust genetic algorithm, named Quick HFC (QHFC), for binary GA problems. This is achieved by combining the ideas of the HFC and

AHFC algorithms described in Section 4.3 with a simple but effective diversity-maintenance technique: the deterministic crowding (DC) algorithm (Mahfoud, 1992). A difficult standard GA test problem, HIFF, is used to show how the often-used DC diversity-maintenance technique fails, how its performance depends strongly on the population size and how HFC can be used to make it more robust in terms of both population size and search capability. The performance of QHFC is compared with a standard GA enhanced by deterministic crowding (DCGA) and DCGA with a multi-partial-reinitialization method (DCGA+MR), illustrating that the HFC principle can transform the convergent DCGA into a much more robust and capable search algorithm. It turns out that QHFC has enabled using much smaller population sizes and many fewer evaluations to solve rather large instances of the HIFF problem.

4.6.1 The Motivation & Design Rationale

A straightforward implementation of the HFC principle for a binary genetic algorithm problem simply requires transforming HFC-GP and AHFC-GP in Section 4.3 into HFC-GA or AHFC-GA by replacing the tree representation with a binary representation. However, preliminary experiments showed that these two algorithms don't work in that form for binary GA problems. Close examination reveals the reason and provides further insight into the HFC principle for sustainable evolution. It is well known that one of the significant differences between GA and GP is that the loss of diversity in a binary GA with crossover and small mutation rates is much faster than that in GP. Crossover of two identical binary chromosomes produces two identical offspring, while in GP, crossover of two identical trees usually produces distinct offspring. As a result, the levels of HFC-GA converge very quickly for binary problems because of the fixed allocation of breeding probabilities to levels used in HFC-GA, thereby violating the HFC principle that the lower level should be able to ensure sustainable export of individuals (thus maintaining diversity) into higher levels; otherwise the higher level will just "die out," deprived of its supply of new and viable genetic material, as traditional GA and GP do.

So the question reduces to how to ensure sustained and useful diversity for all intermediate levels. One solution is to allocate much more breeding opportunity to lower levels. But this will reduce the relative effort of exploitation of higher levels. A better strategy is to do more breeding in lower levels to improve their diversity only when the higher levels detect that lower levels have converged to some extent, and then allocate them more breeding opportunities to let them catch up. This requires an effective mechanism to detect the convergence of a subpopulation. There are many mechanisms to measure the diversity of the population (Wineberg & Oppacher, 2003) and some are used in guiding the balance of exploration and exploitation (Ursem 2002). Most of these methods require calculation of genotypic distances between all possible pairs in the population. Although this calculation can be reduced to O(n) time complexity (Wineberg & Oppacher 2003), it is hard to set the diversity thresholds for all levels in HFC, and diversity measured in this sense may not be the useful diversity needed for good evolution (remembering that adding random individuals to intermediate-fitness populations, for example, increases diversity but hardly improves search performance).

Based on the above analysis, a new, practical mechanism is introduced into the original HFC framework to support the continuing potency of each level in the HFC framework. *Potency* here is defined as the capability of a fitness level in HFC to produce offspring with fitness high enough for export to higher HFC levels. This mechanism for maintaining the potency of each level except the top level is works as follows: starting from the level below the top level, breeding is conducted successively in each level, using steady-state breeding methods, while tracking the number of offspring produced that are eligible for *promotion* (migration to the next-higher fitness level). If a given number of promotable offspring are not produced within a specified number of evaluations at a given level, then a "catch-up" procedure is conducted: a specified fraction of that level's individuals is replaced by individuals taken from (i.e., removed from) the next lower level, and *popsize* genetic operations and evaluations are performed. Then, in turn, the openings created at the next lower level are immediately filled with individuals removed from the level below that, etc., until, at the lowest level, the openings are filled by new randomly generated individuals (however, further genetic operations and evaluations are not performed as part of this "ripple down" filling of openings). This "double loop" procedure assures that each level, before it next breeds, has either recently produced individuals worthy of promotion to the next level or has received new individuals from the next lower level, thus ensuring its potency to export higher-level individuals.

Table 4.15 Quick HFC (QHFC) algorithm

```
Parameters:
Total population size $|P_t|$, bit mutation rate $p_m$
L: number of subpopulations (levels) of QHFC
$\gamma$ : size factor parameter, the ratio of higher level archive size w.r.t next lower level archive size $|P_{k-1}| = |P_k|.\gamma$
breedTopFreq: number of generations to breed top level between potency testing of lower
        levels (via breeding)
detectExportNo: number of individuals from a level that must be promoted for the level to be
        considered potent
catchupGen: maximum evaluations in any but top level, normalized by level's popsize, for
        potency test
percentRefill: percentage of this level's popsize to import from next lower level when there is
        no progress in the top level, or when lower levels fail potency test (do not furnish
        detectExportNo qualified immigrants within specified number of evaluations)
noprogressGen: maximum number of generations without any fitness progress in top level
        before triggering importing of percentRefill individuals from next lower level
Procedure
    1. initialization
        rancomly initialize and evaluate the HFC subpopulations
        calculate the average fitness of the whole population and set it as the admission fitness of the bottom level, $f_{\min}$,
        which is fixed thereafter
        remove individuals with fitness less than $f_{\min}$, and equally distribute the rest of the individuals among the levels,
        according to fitness, thereby determining the admission threshold of each level
        generate random individuals to fill the openings in each archive
    2. while termination_condition is false
```

```
        breed the top level for breedTopFreq generations using generational deterministic crowding and applying mutation
        after each crossover
        if no progress on best fitness of the whole population for noprogressGen generations,
                call import_from_below, but ensuring the best individual is not replaced
        if average fitness of top level > 2 $f_{adm}^{L-1}$ - $f_{adm}^{L-2}$, adjust admission thresholds by evenly allocating fitness range to
        each level:
                $f_{adm}^{k} = f_{\min} + k(f_{\max} - f_{\min}) / L$   for k=0 to L-1
                where $f_{adm}^{k}$ is the admission fitness of level k, $f_{\max}$ is the maximum fitness of the whole population
        //potency testing
        for each level from L-2 to 0
            call do_potency_testing
            if not succeed
                call import_from_below to replace (at random) percentRefill percentage of the current level
                breed one generation at this level
            endif
        end for
    end while

Procedure do_potency_testing ( $l$ )
        $l$ is the level for potency testing
        catchup_evaluation←0
        exportedIndividual ← 0
        while catchup_evaluation < catchupGen*| $P_l$ | and exportedIndividual< detectExportNo
        randomly pick two individuals from level $l$

                if fitness of offspring > $f_{adm}^{l+1}$,
                        promote it (them) to level $l+1$ (replacing randomly any but the best individual or other
                        individuals just promoted) and call import_from_below to replace its (their) closest parent(s)
                        exportedIndividual ← exportedIndividual +1
                else
                do deterministic crowding with the 4-member family
                endif
        end while
        if fail to promote detectExportNo individuals
                return not success
        else
                return success
 Procedure end
 Procedure import_from_below ( l, nImport, victimList )
        $l$ : the level into which to import new individuals from next lower level
        nImport: the number of individuals to import from next lower level
        victimList: a list of indices of individuals which will be replaced by the imported new individuals
        if $l$ =0
                randomly generate nImport new individuals and import into (lowest) level $l$
        else
                randomly choose nImport individuals from level $l-1$ to replace individuals in victimList . If victimList is
                empty, randomly choose victim individual from current level. Put the indices of the new immigrant
                individuals from level $l-1$ into the level $l-1$ newVictimList, whose openings will eventually be filled
                with individuals from level $l-2$ (this assures the replacement of individuals removed from level $l-1$ ).
                call import_from_below ( l -1, nImport, newVictimList)
 Procedure end
```

This mechanism for sustaining the potency of search does not require evaluating any measure of the distance among genotypes or phenotypes, and could also be applied to GP and other sorts of problems. However, the particular QHFC applied here to a GA, a "good" distance function was available, so was used in order to demonstrate how QHFC can transform a "classical" GA with deterministic crowding into a more robust and efficient algorithm. The other difference introduced, in order to facilitate the QHFC application, was to use a generational GA at the top level and a steady-state GA at the remaining levels for breeding.

4.6.2 The Algorithm Framework

The QHFC algorithm is summarized in Table 4.15. Compared with HFC-GP and AHFC-GP, QHFC has many fewer parameters to specify, and the admission thresholds are automatically adjusted.

4.6.3 Test Suites

We test QHFC with a well-known standard test problem in GA: the hierarchical if-and-only-if (HIFF) problem (Watson et al., 1998), an instance of hierarchically decomposable functions (HDFs). In this problem, a binary string of length 2^K is to be evolved, where K is the number of levels in the hierarchy. The fitness of a binary string is defined by the recursive function shown below

$$f(B)=\begin{cases} 1 \qquad\qquad\qquad\qquad , & if\,|B|=1, else \\ |B|+f(B_L)+f(B_R), & if\,(\forall i\,\{b_i=0\}or\{b_i=1\} \\ \quad f(B_L)+f(B_R), & otherwise \end{cases}$$

where B is a block of bits $\{b_1, b_2, ..., b_n\}$, $|\mathrm{B}|$ is the length of the block (n), b_i is the ith element in block. B_L and B_R are the left and right half substrings.

Using HIFF as the test problem is motivated by the fact that diversity maintenance is critical to prevent premature convergence and to solve it successfully (Watson & Pollack, 1999). It is not a separable problem because of the strong non-linear interactions of its building blocks at all levels. Several approaches have been applied to this problem including deterministic crowding, fitness sharing (Watson & Pollack,1999, 2000) and several "competent GAs" with linkage learning mechanisms (Pelikan & Goldberg, 2001), providing a good basis for comparison.

In the experimental study here, QHFC is compared with the generational deterministic crowding GA (DCGA) and its multi-partial-reinitialization version (DCGA+MR) (Table 4.16). DCGA (without multi-partial-reinitialization) is implemented simply by setting *percentRefill*=0.

Table 4.16 DCGA (-MR): Deterministic crowding GA with/out multi-partial-reinitializations

```
percentRefill: the percentage of the population that will be replaced by random individuals.
maxNoProgressGen: maximum number of generations with no progress on the maximum fitness of the population  before
        we partially reinitialize the population
1. Intialize the population randomly
2. Repeat until stopping condition
        randomly group individuals into pairs
        do crossover and mutation for each pair of individuals p1, p2 to produce two offspring c1, c2
        according to pairing rule, each offspring is paired with and compete against one parent
        if the fitness of the offspring is better than that of its corresponding parent, the parent gets replaced
        check if maximum fitness has not been updated after maxNoProgressGen generations, we replace
        percentRefill percent of the population with randomly generated individuals.
Pairing rule: if H(p1,c1)+H(p2,c2) <H(p1,c2)+H(p2, c1) then pair p1 with c1, and p2 with c2, otherwise pair p1 with c2 and p2 with c1, where H. returns the Hamming distance between two individuals.
```

4.6.4 Experimental Results

The following experiments compared the performance of QHFC, DCGA, DCGA with multi-partial-reinitialization for the 128-bit and 256-bit non-shuffled HIFF problems. The 64-bit HIFF problem was not addressed because it is so easy that any conclusion from it may be misleading. In all three algorithms compared, simple two-point crossover with mutation was used, where mutation is applied on offspring immediately after crossover. All methods used a bit mutation rate of 0.0075 for both 128-bit HIFF and 256-bit HIFF. For the DCGA-MR, the *percentRefill*=0.25, *maxNoProgressGen*=10.

For QHFC, in all the experiments, with different population sizes and different problem sizes, a single set of parameters was used.

L: 5 γ: 0.7 *breedTopFreq*: 2 *detectExportNo*: 2

percentRefill: 0.25 *catchupGen*: 20 *noprogressGen*: 10

All three algorithms were studied using a series of population sizes from 100 to 4000, all with maximum number of evaluations 1,000,000 and all experiments with 30 runs. The results are illustrated in Figure 4.19 and Figure 4.20.

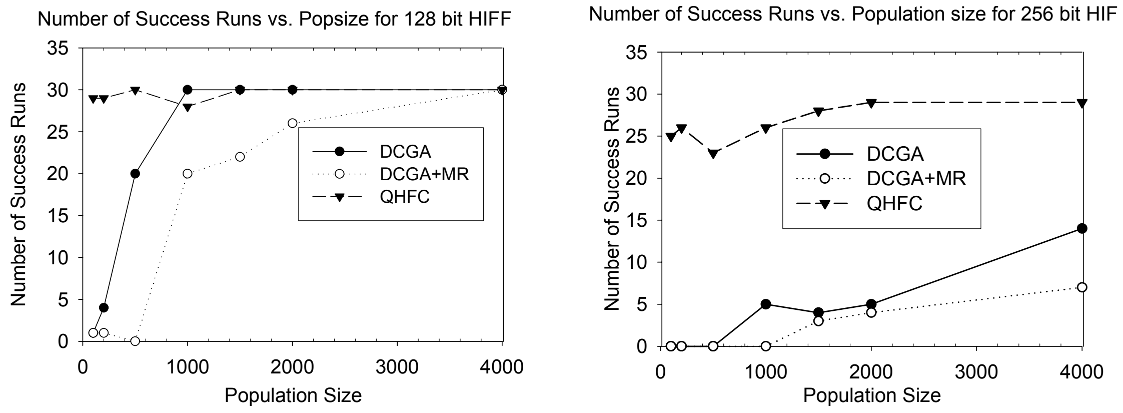


Figure 4.19 Comparison of QHFC with DCGA with/without multiple partial reinitializations in terms of robustness of the search capability and robustness in terms of population size. For the simple 128-bit HIFF problem, DCGA can achieve robust search if the population size is sufficiently large (>=1000). QHFC, instead, can do very robust search from population sizes 100 to 4000. For the difficult 256-bit HIFF problem, a population size of 4000 for DCGA can only find the solution in half the 30 runs, while QHFC can solve it very robustly still using population sizes from 100 to 4000. DCGA+MR apparently suffers from the disturbance of imported random individuals and works worse in both case than DCGA.

Figure 4.19 clearly shows that for the simpler 128-bit HIFF problem in which diversity maintenance is not a big issue, using a simple GA with deterministic crowding could solve the problems quite well given a sufficient population size. However, for real-world problems, correctly estimating the required population size is not easy and population sizing theory can be unreliable and difficult to apply (Goldberg, 2003). Also, the idea of using large populations to maintain diversity is not a scalable approach, as illustrated in the 256-bit HIFF problem, where a GA with deterministic crowding fails in half of 30 runs even with a population size of 4000, while QHFC can still achieve 25 successful runs out of 30 with a population size 100. This robustness is derived from the sustainable diversity promoting mechanisms in HFC, where genetic material from lower levels provides the non-exhaustible diversity. This is a key feature that distinguishes HFC from other diversity maintenance techniques such as niching, restricted mating, island or multi-population models, pygmies and civil servants, and species-conserving GA, as mentioned in Chapter 2. As regards the well-known partial-reinitialization approach, Figure 4.19 shows that DCGA+MR performed the worst among the three methods, simply by wasting evaluations on hybrids of high-fitness individuals with random individuals, where high-fitness individuals with strongly coupled components can not typically be improved by inserting junk alleles.

The advantage of HFC is not limited to robustness of search even with small population sizes; it also allows solving many problems with fewer evaluations, as illustrated in Figure 4.20. For simple problems like 128-bit HIFF, a GA with deterministic crowding wins by using fewer evaluations. This is achieved by allocating all breeding opportunities to high-fitness individuals maintained in the population, while QHFC loses by allocating a portion of its evaluations to lower levels to ensure sustainable diversity. However, one can see that this preventive measurement does not incur a very large penalty in terms of number of evaluations. The strategy of allocating a fraction of the evaluations to lower fitness levels is rewarded by increased robustness and the capability to solve more difficult problems, as shown for the 256-bit HIFF problems in Figure 4.20. With a population size of 4000, a traditional DCGA with/without multi-partial-reinitialization found the optimal solutions in fewer than 15 of 30 runs (Figure 4.20). The successful runs were obtained using three times the number of evaluations used by QHFC with a population size of 2000, which also succeeded in all 30 runs. The influence of population size in QHFC is interesting. With an extremely small population size of 100, QHFC can still achieve quite robust search and find the optimal solution in 25 runs out of 30, but with much larger variation in number of evaluations. With a larger population size (1000-2000), QHFC can find the optimal solutions within more consistent numbers of evolutions.

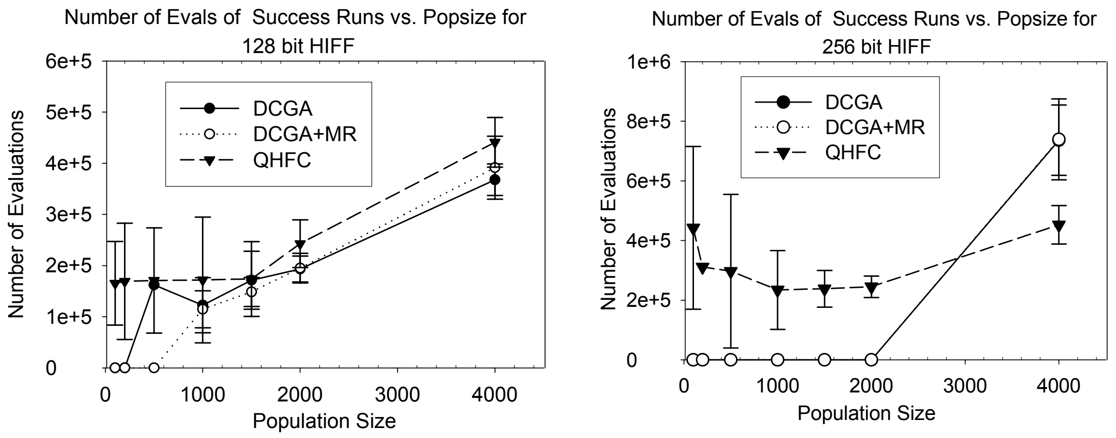


Figure 4.20 Comparing QHFC with DCGA with/without partial reinitializations in terms of average number of evaluations needed to find the optimal solutions. For the simple 128-bit HIFF, DCGA wins with slightly fewer evaluations, but HIFF wins by being able to use much smaller population sizes. For the 256-bit HIFF, QHFC wins in terms of both robustness of search (finding the optimal solutions in 25-29 runs out of 30 for the population sizes of 100 or higher) and number of evaluations (with population size 2000, QHFC succeeded 29 times, using only about one-third the evaluations of DCGA (with a population size 4000 and with only half the runs successful).

When population size is set larger than necessary, QHFC takes more evaluations to find the optimal solutions than with smaller population sizes, but still many fewer than DCGA needs. This increased average number of evaluations to find optimal solutions with overly large population size is caused by spending too many evaluations in lower levels, since the same set of parameters was used for all experiments, and thus many evaluations were done in lower levels. One important lesson to draw here is that QHFC is quite scalable for the HIFF problem, in that the required population size and the number of evaluations needed do not vary much for population sizes from 100 to 4000. QHFC is also seen to be quite scalable in terms of problem size, with about 160,000 evaluations needed for 128-bit HIFF and about 234,000 for 256-bit HIFF.

Results on HIFF problems have also been reported in (Watson & Pollack, 1999), where a GA with domain-knowledge-based fitness sharing and a population size of 1000 were used. It showed that even this fitness sharing method with considerable domain knowledge fails on the 128-bit and 256-bit problems, while QHFC with population size 100 and 200 consistently and reliably solved the 128-bit and 256-bit HIFF problems.

The results of QHFC were also compared with those of the somewhat complicated hierarchical Bayesian Optimization Algorithm (Pelikan & Goldberg, 2001, 2003), one of the *competent* GAs with explicit linkage learning mechanisms. The optimal result of hBOA on the 128-bit HIFF was about 26,000 evaluations, with a standard deviation of about 12,000, and on the 256-bit HIFF, was about 88,000 evaluations, with a standard deviation of about 10,000. QHFC, without any linkage learning mechanism, achieved competitive results with much smaller population sizes (down to 100), reliably. For the 128-bit HIFF, QHFC used about 160,000 evaluations, and used about 234,000 for 256-bit HIFF (averaged over 30 runs). Clearly QHFC, as a linkage-blind approach, uses many more evaluations. But about half of 30 QHFC runs succeeded within 85,000 evaluations for 128-bit HIFF and within 153,000 evaluations for 256-bit HIFF. The discrepancy in number of evaluations needed compared with hBOA is less with larger HIFF problems – the QHFC appears to be scaling with a lower slope than the hBOA on this problem. And as the results reported in (Pelikan & Goldberg, 2001) are the best results after tuning of population sizes of hBOA empirically to minimize the number of evaluations for each problem instance, while the same set of parameters was used for all

QHFC experiments here, independent of the population size and problem size, it is hard to do precise comparison. QHFC may, in fact, be tunable to require fewer evaluations than reported here. QHFC has also been tried on some other hierarchical deceptive functions (to be reported elsewhere), again obtaining very good results.

However, one fact to be pointed out here is that the performance of QHFC here does not apply to shuffled HIFF problems, while hBOA can solve both shuffled and non-shuffled HIFF reliably. As a shuffled HIFF problem is not solvable by GAs with a simple crossover operator (Watson & Pollack, 2000), i.e., the type used with QHFC here, the conclusion is that QHFC may greatly improve the robustness and scalability of various EAs, so long as the operators used are appropriate to the given problem. Based on the scalability and robustness of QHFC on the hierarchical HIFF problem with tight linkage, it seems that QHFC can achieve a reasonable level of hierarchical problem solving capability. Since the simple crossover operator is the bottleneck for QHFC to solve shuffled HIFF problems, one interesting idea is to combine QHFC with BOA and to see if this hybrid can compete with hBOA (Pelikan & Goldberg, 2003). To do that, one only needs to replace the crossover operator in QHFC here with the Bayesian model building method in BOA. The combination of QHFC with hBOA also appears extremely promising.

Compared to previous HFC algorithms, there is an important conceptual progress in QHFC. In the original formulation of the HFC model, we require that only qualified individuals can be migrated into corresponding fitness levels. In QHFC, this criterion is relaxed: individuals of lower levels are to be migrated into higher levels when these levels stagnate without progress for a certain number of generations. The experimental results in this section and for other three benchmark problems (Hu et al., 2004) confirmed the validity of this refinement over original HFC model.

4.7 Why and How to Use HFC for Sustainable Evolutionary Search

Given that there is a bunch of HFC algorithm available, it is a good strategy to compare their properties and relate them to traditional techniques to sustain the evolutionary search. This will further deepen our understanding of the nature of HFC and how to exploit the principle further.

4.7.1 The Advantages of HFC-Based Evolutionary Algorithms

Based on the experiments and performance evaluations of five sustainable evolutionary algorithms designed based on the hierarchical fair competition principle, it is good to outline the essential capabilities of HFC to show us why we should prefer to use HFC based evolutionary algorithms and in which situation to use them.

I. Sustainability of evolutionary search

By ensuring a continuous supply and incorporation of genetic material in a hierarchical manner, and by culturing and maintaining, but continually renewing, populations of individuals of intermediate fitness levels, a continuing search capability can be achieved by HFC based EAs. HFC employs an assembly-line structure in which subpopulations are hierarchically organized into different fitness levels, reducing the selection pressure within each subpopulation while maintaining the global selection pressure to help ensure exploitation of good genetic material found. The competition segregation mechanism of HFC ensures that no convergence of the population exists to the vicinity of any set of optimal or locally optimal solutions and thus no premature convergence is allowed. The exploration capability is always maintained by the continuing search at all levels sustained eventually by the constant import of random individuals. HFC essentially transforms the convergent nature of the current EA framework into a *non-convergent* search process.

It is evident that the search sustainability of HFC algorithms is not achieved by increasing the population size. As demonstrated by both the generational HFC and the QHFC, HFC-based evolutionary algorithms can use smaller populations to achieve better results, thereby challenging current population-sizing theory.

Also note how HFC avoids the detrimental "founder effect", i.e., that the early established "coupled structures" of the genomes prevent further modification. This is achieved by constantly creating brand new such high fitness individuals from lower levels. Thus, the tendency of "standard" EA's to narrow their search fairly rapidly to the earliest-discovered regions of relatively high fitness is countered. This allows more thorough exploration around new individuals (usually with low-fitness) that may contain ultimately valuable genetic material that might be discarded by standard EA's. Because low-fitness individuals are not forced out of the lower levels by competition from higher-fitness individuals, they continue to explore the space widely, feeding promising new search regions to higher-fitness subpopulations as they are found. Another good property of HFC to do sustainable search is that it doesn't require the distance function though this information may be beneficial in some cases, while most existing techniques to sustain search ask this information and many applications just do not have a good appropriate distance function.

II. Reliability of evolutionary search

Stochastic search techniques like evolutionary algorithms are often criticized for their low guarantee of their confidence of the search. These algorithms are usually have low reliability and are very sensitive to the population size and other parameters. However, in many real-world problems, the reliability of an algorithm to give reasonably good solutions is very important. Multiple experiments of HFC clearly demonstrate the outstanding reliability of HFC based evolutionary algorithms. For examples, the QHFC can find the optimal solutions of 256 HIFF very reliably even with a population size 100 (25 out 30 runs), while the traditional state-of-art diversity maintaining technique can only succeed for 14 out of 30 runs even with a population size 4000.

III.Robustness of the search parameters

Evolutionary algorithms have also been widely criticized by their many parameters and the need to tune those parameters for good performance. For some problems, such as space exploration applications, there is no chance to tune such a tuning process. In other problems, such tuning is too expensive to do. HFC algorithms instead have shown an extraordinary capability to be insensitive to their tuning parameters. Especially in the experiments with QHFC a single set of parameters without any tuning was used for all experiments. Nevertheless, all results are better than those of traditional techniques.

IV.Efficiency of the search

Any efficient search algorithm needs to address the exploitation and exploration problem and achieve good balance between these two. It is well established that, to speed up the search process, the selection pressure should be sufficiently high to exploit the high-fitness individuals. Ideally, using extreme elitism to always keep the best K (the population size) individuals would have the strongest selection pressure. However, to ensure the search is sustainable and does not suffer from premature convergence, one needs to lower the selection pressure to keep sufficient diversity. Traditional diversity-maintaining techniques usually achieve diversity by sacrificing the convergence speed, and even this diversity is not productive diversity for sustainable search because of the "founder effect".

HFC provides another solution to this dilemma. By segregating the high-fitness individuals from the worse ones and promoting superior offspring quickly to a place where they are free to compete with, and be recombined with, each other, HFC can quickly exploit high-fitness individuals as they are discovered. And the breeding between high-fitness individuals are usually more productive than breeding high-fitness individuals with bad ones. The selection pressure of higher levels can be set as high as appropriate as along as the progress is supported by lower levels and is sustainable. The potency testing mechanism in QHFC is a very effective mechanism to ensure this sustainable fast progress. If the top level converges too quickly, this mechanism will automatically allocate more breeding to lower levels to bring in diversity. Thus in QHFC algorithm, the search can run as fast as possible, but not too fast. The result is that very efficient search as demonstrated in QHFC experiments. With double the reliability of deterministic crowding, we also achieve results with one third of the evaluations.

V. Wide applicability

Since HFC model is representation independent, it is widely applicable to a wide areas including but limited to all kinds of evolutionary algorithms, directional artificial evolution, evolvable hardware, etc. It has already been applied to genetic programming, genetic algorithms, and multi-objective evolutionary algorithms. We expect its wide use in improving other search algorithms, including the evolution strategies.

VI. Scalability of the search

It is observed that in HFC algorithms, there is no requirement on global information. Thus, it has a good scalability with respect to increasing number of processors. As more processors are available, they can be distributed to different fitness levels – either to low-level subpopulations for more extensive exploration, or to the higher-level ones for intensive exploitation of high-fitness individuals. The migration can be conducted horizontally among similar levels and vertically between high and low levels.

VII. Solving multi-modal problems

The HFC model maintains a large number of high-fitness individuals in high-fitness-level subpopulations without threatening lower-fitness (but perhaps promising) individuals. And this maintenance does not strongly depend on the population size as in the case of fitness sharing. The capability to solve the 256 non-shuffled HIFF reliably shows that QHFC can maintain a good diversity in the high levels. By introducing the age-based speciation, this capability can be further extended.

VIII. Compatibility with existing techniques

Goldberg (2002) describes two types of approaches to improve the scalability of an EA. The first is the explicit building block discovery and exploitation mechanisms, as used, for example, in his messy GA and in the automatically defined function ("ADF") of Koza (1994), which may transform intractable problems into tractable ones. The second type includes the family of parallel GA models, including both coarse- and fine-grained variants, the reduction of evaluation effort through use of more rapidly calculated approximations to the fitness function or fitness inheritance, and the hybridization approach.

As a generic framework for continuing search, HFC is compatible with all of these approaches, and may be applied in addition to any of them. HFC is a natural extension of the island (coarse-grain) model for parallel EAs, although it may also be implemented as a specialized selection scheme for breeding and survival within a single population (Hu, Goodman et al., 2003). It is also clear that the continuing search enabled by the assembly line structure of HFC, when coupled at each level with existing techniques like fitness sharing, species conserving, and elitism, can yield yet more efficient and effective EAs, as is demonstrated in QHFC and more studies to be reported in the future.

It is already demonstrated that if the genetic operator is not appropriate, HFC as a general model can't help. It cannot solve the shuffled 256 HIFF problems easily because single crossover operators do not work on this problem. But HFC can be convenient combined with better operators such as Bayesian Optimization Algorithm, linkage-learning crossover to improve the search capability. Since HFC is not a selection operator, but a generic framework, it can easily hybridize with many other techniques in evolutionary algorithms in the same way as many techniques are applied to multi-population EA.

4.7.2 Relationship of HFC with Other Techniques to Sustain Evolution

I. Relationships with Other Techniques in Terms of Diversity Maintenance and Incremental Evolution

Although HFC does *not manage* diversity *per se* as an approach to ensuring continuing evolution, HFC essentially provides several effective ways that do maintain diversity. First, the continuous introduction of random individuals into the lowest level subpopulations ensures the supply of diversified genetic material, which can then be exploited in the "fair competition environment" of low-fitness-level subpopulations. Thus, HFC implements the equivalent of an *effective* multi-start or re-initialization mechanism on a continual basis. HFC also maintains diversity by decreasing the risk of dominance by the earliest-discovered high-fitness individuals. At each fitness level, HFC ensures that the competition of individuals is fair. Only individuals with comparable fitnesses are allowed to stay in a certain fitness level, and new competitive immigrants from lower levels are continuously incorporated. If a new offspring turns out to be extraordinarily fit, it will immediately emigrate to a yet-higher fitness level. So, essentially, the local selection pressure at each specific level is low while the global selection pressure is maintained by the stratified structure of fitness levels.

The structure of HFC is especially suited for incremental evolution, where the solutions usually need a steady-state developmental process for refinement. In HFC, this process is realized by the climb of developing individuals from lower fitness levels to higher fitness levels. Along the way, they compete only with developmentally similar individuals, assuring them sufficient time to "mature."

II. HFC's Stepping Stone Supply vs. Explicit Linkage Learning or Modular Approach

HFC works by ensuring continuous availability of "stepping stones" – individuals at a hierarchy of fitness levels and accompanying levels of co-adaptation of genetic material, populating the range from randomly generated individuals to the best yet discovered. It assures the descendants of newly generated individual has time for "maturation" in the presence of other individuals of similar fitness, throughout the fitness hierarchy, avoiding strong selection pressure or competition with more fit individuals. HFC does not require the existence of building blocks to work well, but takes advantage of them implicitly if they are present. This is in contrast with the explicit building block exploitation in messy GA and other linkage-learning GAs. It is also different from the half-blind building block exploitation in the ADF mechanism of GP. However, the apparent effectiveness of HFC justifies its role as a generic framework for continuing evolutionary search, which is not guaranteed by the linkage learning and other modular approaches alone. Coupling the continuing search capability of HFC with the explicit building-block-discovery capability of linkage learning is expected to achieve a synergistic effect for suitable problems as discussed in Section 4.5.

III. HFC, Restarting, and the Short-Run-or-Long-Run Dilemma

HFC provides a good solution to the short-run-or-long-run dilemma (Luke, 2001; Goldberg, 1999). With a given number of total evaluations available, it is hard to decide whether it is better to run multiple (independent or serial) epochs with small population sizes or to run a single epoch with a large population size. Running with large population sizes may lead to convergence too quickly due to unbalanced scaling of building blocks (Goldberg, 1999) while small population sizes may not provide sufficient building blocks. Use of a rejuvenating operator still suffers from lack of intermediate building blocks, as discussed in Chapter 2. However, HFC, which is not very sensitive to the population size, allows to the use of much smaller population sizes, while preserving capability for continuing search. It is a natural mixing of implicit parallel and serial processing (Goldberg, 1999). Compared with multi-epoch EAs or other restarting techniques, HFC saves a huge amount of computational effort by reusing building blocks.

IV. HFC and Other Schemes with Hierarchical Organization of Subpopulations

Some other EAs also employ a hierarchical organization of subpopulations. The Pyramid GA (Aickelin, 2000) used lower-level subpopulations to optimize single aspects (of the objective function) of the problem while higher levels imported individuals from lower levels for further recombination. Hsu et al. (2002) used similar ideas (LLGP) for agent evolution. The injection island GA (or iiGA) (Lin et al., 1994) used a hierarchical organization to implement multi-resolution or other heterogeneous, layered search. As a continuing search framework, HFC is conceptually different from these approaches, which still suffer from the convergent nature of the traditional EA framework. In HFC, the hierarchical organization of subpopulations by fitness provides a mechanism for maintaining the intermediate stepping stones necessary for continuing search.

4.7.3 Which One to Use: Comparison of HFC Family EAs

We classify the existing five HFC algorithms according to the following criteria:

1) Generational or steady-state model

Generational HFC model is easy to implement and understand. Steady-state HFC is more similar to the natural evolutionary process and is a good model for implementing local update mechanisms and is more responsive to the dynamics of evolution since it doesn't need to wait until one generation of evaluation. This is the reason why the QHFC uses the steady-state model for potency test.

2) Competition segregation mechanism

   Vertical niches and hierarchical competitions in nature is segregated by the physical environment while in evolutionary algorithms there doesn't exist such and diversified environment. In Section 4.2.3, we outlined several mechanisms to segregate competition including multi-population, tags, and density control, all allowing the lower level young individuals to have the chance to grow up. Multi-population mechanism

3) Admission threshold adaptation

   It is desirable to design a mechanism to automatically set the admission thresholds to make the algorithm more autonomous. In HFC-ADM and QHFC, such kinds of adaptive thresholds mechanisms are proposed. CHFC goes even further. It completely removes the admission thresholds and still enjoy the benefit of segregation of competitions.

4) Breeding strategy for different levels

   Currently, there are three mechanisms to control the amount and order of breeding of individuals at different levels. One is the synchronous sequential breeding for all levels as in the generational HFC, where the amount of breeding of each level is proportionate to their population size. However, it is not necessary to relate the breeding opportunity with the population size of different levels and it is also not necessary to breeding all levels synchronously. Then the second breeding strategy is introduced in Continuous HFC, which can separate the breeding opportunity of a level from its population size. The breeding sequence also deviates away from the synchronous sequential breeding. Instead, the level to breed at a specific instant is probabilistically chosen according to the probability distribution. A third adaptive breeding strategy is proposed in QHFC, where the breeding of lower supporting levels are called only when it is necessary. This breeding-on-request mechanism can greatly reduce the unnecessary breeding of lower levels. While conventional EAs have much difficulty to make a balance between strong selection pressure to make quick progress and lower selection pressure to maintain diversity, QHFC provides a promising solution to this problem: an EA can now run as fast as possible and slow down only when it is necessary. The comparison of current five HFC algorithms is summarized in Table 4.17.

Table 4.17 Comparison of features of HFC family algorithms

features	algorithm property	Generational HFC, HFC-ATP	HFC-ADM	CHFC	QHFC	HEMO
EC model	generational	√	√		hybrid	
	steady-state			√		√
competition Segregation mechanism	multi-population	√	√		√	√
	density control			√		
admission threshold setting	Fixed	√		n/a		
	adaptive		√	n/a	√	√
breeding strategy for different levels	fixed	√	√			
	probabilistic			√		√
	adaptive				√	

Based on the above analysis it is clear that QHFC will be best for multi-population type of HFC algorithms and CHFC can be used when only one population is used and theoretical study is needed. It should be noted here that many of these features are not exclusive and better algorithms can be devised by combining some desirable features. For example, combine the breed-on-request breeding with the continuous HFC would lead to even better evolutionary algorithms.

4.8 Summary

In this chapter, the Hierarchical Fair Competition (HFC) model is proposed as a generic framework for sustainable evolutionary search for solving complex problems. This continuing search capability is achieved by ensuring sustained access to "stepping stones" of intermediate fitness levels in a sequence of fitness-stratified breeding pools. For problems exhibiting building block structure, these pools act as repositories containing a succession of increasingly large and fit assemblies of such building blocks. Understanding the dynamics of HFC provides a new insight into the premature convergence of EAs as well as the structure of an EA itself. Instead of trying to escape local attractors from relatively highly converged/evolved populations containing individuals having strongly coupled components, a better strategy is to ensure continuing search by allowing the emergence of new optima in a bottom-up manner.

The convergent nature of the extant EA framework is examined in terms of the effect of the continuous increase of the average fitness of the population. This inherent property makes the horizontal-spreading diversity maintenance techniques, such as niching, inefficient in the case of complex, highly multimodal problems, as is common in the program spaces of GP. An argument is made that it is important that, at any time, the population contain some number of “stepping stones” – intermediate-fitness individuals, continually updated by synthesis from lower-level stepping stones, and managed so as to assure that they are capable of producing higher-fitness individuals.

Inspired by the fair competition principle in societal and economic systems, the HFC model consists of three components: subpopulations structured hierarchically according to fitness, a random individual generator active throughout the evolutionary process, and a mandatory, universal migration policy from lower fitness levels to higher fitness levels. The assembly-line structure of HFC enables it to search the problem space with steady-state progress by maintaining a hierarchy of stepping stones, spanning the space from randomly generated individuals to the fittest individuals yet discovered. HFC is representation independent and thus applicable to binary-coded GAs, integer- or real-valued GAs, tree-based genetic programming, etc. This model shows good characteristics in terms of diversity maintenance, incremental evolution and building block exploitation.

Five HFC based evolutionary algorithms were evaluated with a set of standard GP/GA benchmark problems and a complex analog circuit synthesis problem. In all of these experiments HFC achieved much better robustness, found better solutions in terms of average best fitness, and used fewer evaluations than other GA and GP techniques. These results demonstrated that HFC constitutes a useful technique to improve existing EAs.

Chapter 5

SCALABLE EVOLUTIONARY SYNTHESIS

In previous chapters, a sustainable evolutionary computation framework is proposed to solve more challenging problems. However, the scalability of evolutionary synthesis of systems also depends on many other factors involved in designing artificial evolution systems, such as genotype representation, genotype-phenotype mapping, genetic variation operators, etc. The second part of this dissertation concerns the principles and techniques for achieving scalable evolutionary synthesis.

In this chapter, four fundamental principles for handling complexity in both artificially designed systems and biologically evolved systems are examined. Given this context, some critical limitations of standard genetic programming based approaches for evolutionary synthesis will be discussed and a brief survey of the state-of-the-art of research towards scalable evolutionary synthesis will be presented. Several promising research directions are also discussed at the end of this chapter.

5.1 The Scalability Issue of Evolutionary Synthesis

At the "end of the beginning" of evolutionary synthesis by genetic programming, the most central issue is how this paradigm can be scaled up to solve large complex synthesis problems (Lipson et al., 2003). At the time of writing, in the domain of automated synthesis of analog electrical circuits where GP has been most successfully applied, the maximum number of components is about 100-300. Koza (Koza et al., 2004) suggested that there are unexploited potentials in the current genetic programming approach that may enable us to solve industrial-strength analog circuit synthesis problems, including increase of computer power, domain knowledge, problem-specific knowledge, and faster simulators. However, from the experience of our group in bond graph synthesis by genetic programming (Seo et al., 2002; 2003a, 2003b) and many others (Luke, 2003), it is clear that there are some fundamental limitations of current evolutionary synthesis approaches including GP that prevent us from achieving the desirable scalability.

5.2 The Fundamental Principles for Handling Complexity

Interestingly enough, handling complexity has been a central topic in both engineering (for artificially designed systems) and biology (for naturally evolved systems). How complex are our engineering systems? The Windows XP system of 2002 is composed of 40 million lines of source code (Wheeler, 2004); the open-source Linux system, 30 million physical lines of code. In the semiconductor industry a typical LSI circuit is composed of 10,000 to 100,000 elements; a VLSI circuit is composed of more than 100,000 elements. A Pentium 4 CPU for personal computers of 2000 has 42,000,000 elements. To design or evolve such large-scale systems some fundamental engineering principles are needed to manage the complexity involved, of course explicitly enforced by human engineers. However, the complexity of these artificial systems is dwarfed by the staggering complexity of the biological systems that result from biological evolution. A human brain contains about 100 billion neurons and a human body contains about 65 trillion cells. There must be some fundamental principles to manage this complexity, enforced by the evolutionary process itself. By examining the mechanisms behind both artificial and biological systems to handle complexity, we can identify some interesting similar principles.

The most important strategy of human beings to handle complexity is the decomposition or divide-and-conquer strategy. Based on the understanding of a high-level design objective, human engineers divide the big problem into several sub-problems based on their design experience and domain knowledge; these sub-problems can be decomposed into even smaller sub-sub-problems until the complexity is manageable. In software engineering, this is called the module decomposition stage. This decomposition principle is enforced extensively in the objected-oriented paradigm of software design, and is usually labeled as the principle of modularity (Baldwin & Clark, 2003). A modular system is one that has been decomposed into a set of cohesive and loosely coupled modules, connected by pre-specified interfaces. One of the benefits of modularity is that individual components of a system can be examined, modified and maintained independently of the remainder of the system, thus greatly improving the maintainability of the whole system as it evolves according to the changing demands of its function. Clearly, this is a top-down decomposition approach to handle complexity. Interestingly, the same decomposition principle is invented by biological evolution in a bottom-up way. It is argued that "modularity is one of the most important concepts in all biology, and it is fundamental to any discussion of development and evolution" (Gilbert, 2003). The modularity of the brain is another indication of how decomposition is employed in brain evolution to handle complexity.

The second important principle for handling complexity is reuse. A key motivation of the object-oriented paradigm in software engineering is to improve the reusability of software modules, as implied by the modularity principle. Indeed, one purpose of the encapsulation principle of objected-oriented design is to build a set of interchangeable, expandable, and tailorable software building blocks, thus greatly promoting the reusability of the components for a variety of systems. So, decomposition is not just to make tasks manageable; it is also aimed at allowing future reuse of the modules as building blocks without "reinventing the wheel." In the biological world, the reuse strategy is exploited to its extreme. All of the trillion cells of a human body have an identical genome. The housekeeping genes are reused again and again by all cells. And most of the genes responsible for development of left hands are reused in the developmental process for right hands. There are two types of reuse strategies. One is to simply duplicate an existent module and apply it elsewhere – a repetition approach. Another, subtler and more critical, reuse strategy in both object-oriented programming (OOP) and biological evolution is reuse by inheritance. This corresponds to the principle of inheritance in the OOP paradigm in software engineering and also, surprisingly, corresponds to the cell specialization mechanisms in biology. It is somewhat amazing how similar the class hierarchy within a lineage in OOP is with the cell specialization type hierarchy in biology. One great advantage of reuse by inheritance is that modification/improvement of a base class leads to modification/improvement of all downstream classes. Similarly, the optimization of genes of upstream cell types automatically optimizes the behaviors of cells of downstream cell types.

The third important principle for handling complexity is context. This principle is less obvious than the previous two, but is very important. In software engineering, there are two mechanisms to ensure the appropriate reuse of modules. One is the syntactic definition of the interaction interface, which specifies the conditions for use of a module. The other mechanism, however, depends on software engineers, to ensure the consistency of semantics of the module for its purpose. By ensuring the consistency of the both syntactic and semantic contexts, specific components can be modified or exchanged with new compatible functionally improved components without deteriorating the function of the whole system. This greatly increases the evolvability and maintainability of software systems.

In contrast with artificial systems, living organisms are evolved by the blind evolutionary process through recombination and mutation of genomes. There is no such thing like a conscious brain to help ensure compatible contexts such that during the exchange of genetic material between two genomes no essential genes are lost during this process, which may lead to unviable or defective offspring. Actually, this process is well controlled in biological evolution by the homologous recombination mechanism (Weaver, 2002). One definition of homologous recombination is: *breakage and reunion between homologous lengths of DNA mediated in Escherichia coli by products of the genes RecA and RecBCD, and functionally equivalent genes in other organisms* (www.biochem.northwestern.edu/holmgren/Glossary). The purpose of homologous recombination is to bias the genetic exchange toward exchanging very similar segments of the genomes that perform similar functions. This will ensure that segments encoding the eye color won't be exchanged with segments encoding the hair color. The detailed mechanism of homologous recombination is not clear yet. But apparently it is implemented through the physical and chemical environments and their interactions with the DNA molecule during mitosis. It is the binding specificity of molecules that serves as the mechanism to detect corresponding homologous DNA segments. This binding specificity also guides the developmental process in which cells depend on physical and chemical environmental information to determine what to do at specific developmental stages. For example, morphogen gradients are used as the position clue for development of the body plans of many animals. So even though these kinds of physical media cannot tell whether it is good or bad to apply a specific module in this context, it at least can always make sure that modules with a specific type of binding properties should be developed at this context. The environment can then evaluate the goodness of the developed phenotypes.

Another important principle for building complex systems is the generative representation embodied in the evolution of the development programs of living organisms. Instead of evolving the detailed specifications of phenotypes, a generative representation proposes to evolve the building plans for the phenotypes. It is shown that a generative representation has unique advantages in scalability and evolvability (Hornbey & Pollack, 2002; Hornbey et al., 2003). Another advantage of a generative representation is the natural way to exploit self-organization mechanisms in the unfolding process from genotype building plans to phenotypes. Self-organization has been regarded as one of the most important factors leading to the emergent order in life (Kauffmann, 1993). According to this theory, much of the order we see from living organisms is the result of the self-organization process arising from the interaction among physical entities.

5.3 The Limitations of the Current GP Approach for Scalable Synthesis

Genetic programming was invented as a simple extension of classical genetic algorithms without much consideration of scalability issues in mind. Naturally, there are several fundamental limitations with respect to the four major principles for handling complexity in design.

The standard genetic programming (Koza, 1988; Cramer, 1985) uses the syntax of trees as the genome representation scheme. This scheme, in its basic form, does not have explicit support for evolving modular systems and thus no support for effective reuse. To solve that, Koza (1994) proposed Automatically Defined Functions (ADF) to support reuse of code. In this approach, a chunk of code is evolved along with the main program; multiple references in the main program lead to reuse of the same code chunk.

There are two fundamental limitations of the ADF mechanism. One is that there is only weak support for specification of the context for module reuse, usually only through specification of the type of the root-node of the reused subtree/module. The second limitation is the lack of support for inherited reuse similar to the specialization of cells. Although gene duplication and ADF duplication with subsequent invocation provide a mechanism to derive a new module from an existent one, there is no more relationship between these two modules after this duplication. The result is that improvement of the parent module will not affect the function of the offspring module. Another limitation of ADF is that it has limited capability to support hierarchical composition of modules by evolving stable relationships among modules. Developmental mechanisms have also been introduced into standard genetic programming by many people (Gurau, 1994; Koza, 1994; Miller & Thomson, 2003; Kumar, 2004) and are used in most GP-based evolutionary synthesis systems. However, compared to biological developmental processes, there are many sharp differences. For example, there is no two-way interaction between developing phenotypes and the genomes in GP, which may be essential to support the biological mechanisms of reuse, modularity, hierarchy, and context. Because of these limitations, it is not expected that current genetic programming can achieve the desired scalability.

Lack of modularity support in genetic programming also limits the capability of the parameter evolution mechanism in evolutionary synthesis. It is well known that current parameter optimization techniques can only solve problems with at most several hundred variables, so long as there are non-additive interactions among variables. For large synthesis problems like VLSI design, the number of parameters can easily reach thousands of variables. Clearly, decomposing the synthesis problem of a complex system into evolving modules and their connections can greatly reduce the burden of search in a huge parameter space. Koza et.

al.'s work (Koza et al., 2003) on evolving parameterized topologies represents the direction of achieving scalable evolutionary synthesis in terms of parameter search. But their work can only work on explicitly defined modules rather than automatically defined modules.

5.4 The State –of –the Art in Scalable Evolutionary Synthesis

There are many research efforts toward improving the scalability of the evolutionary synthesis paradigm, almost touching upon each of the four principles discussed above. In this section, I would like to discuss the representative work in each of these genres and propose some additional research directions that may be promising.

One of the earliest instances of scalability research comes from Koza and his colleagues with the introduction of the ADF (1994). A good discussion of bottom-up problem solving by automatic decomposition is provided in the first chapter of this book (1994). Later work to improve reuse support of genetic programming includes the module acquisition (Angeline, 1994) approach, which, however, tends to collect useless code because of its blind extraction of partial codes in the main programs and subsequent release of these code chunks. Thus it has no way to evolve stable and useful modules. Rosca and Ballard (1996) extended the ADF approach by enhancing it with a learning scheme to figure out the fitness of each module. Koza's ADF, since no separate fitness criterion is applied to it, also tends to accumulate useless chunks of code. Rosca's ARL introduces a fitness for each module, thus improving GP's support for evolving modular systems. But the performance improvement of ARL is not significant compared to the ADF. In the genetic algorithm area, Watson (2000) proposed a genetic algorithm called the symbiogenetic evolutionary adaptation model (SEAM), in which modules are explicitly represented in the population and symbiotic composition is used as the major operator. However, this approach is a kind of constructional modularity rather than decompositional modularity (Reisinger, 2004). Reisinger et al. (2004) investigated how to evolve modular neural networks, but achieved limited progress.

Recently, two workshops were organized toward understanding mechanisms of hierarchy, modularity, and regularity for scalable evolutionary synthesis (Lipson, Koza et al. 2003; 2004). Clearly, the issue of how to reinvent genetic programming to support hierarchical modularity and improve automatic decomposition and reuse is attracting strong attention from the GP community.

The second major research direction toward scalable evolutionary synthesis is inspired by the biological developmental mechanism, including developmental GP (Gruau, 1993), computational embryology (Kumar, 2004), artificial ontogeny (Bongard, 2002; Eggenberger, 1997; Miller, 2003). Most of them are only applied to evolutionary synthesis of neural networks, and none of them has achieved competitive results with respect to the standard developmental GP approach. This implies that emulating the developmental mechanism is not the whole story of scalability. Better understanding of the fundamental principles of biological evolution is still critical to design of scalable artificial evolutionary systems.

Two important investigations with regard to understanding the role of context in genetic programming and the role of self-organization or emergent order in artificial evolution are (Lones, 2004) and (Bentley, 2004).

Lones proposed an innovative implicit context concept in his enzyme genetic programming (Lones, 2004; Lones & Tyrrell, 2004). Lones classified contexts into three classes. The standard GP uses an explicit context for each component, which is the type of the root nodes of the exchanged subtrees. This position-dependent context cannot be preserved by the crossover operation, which usually results in the production of offspring whose behavior is quite different from their parents. In other words, the crossover in GP cannot do homologous crossover to ensure that the exchanged subtrees have similar functions. This violates the inheritance principle of evolutionary computation and leads to low performance. Actually, most of the offspring of crossover operations in GP have much lower fitness than their parents. The second type of context is indirect context, used in graph-like genetic programming (Poli, 1997, Miller & Thomson, 2000), which uses arbitrary reference to other components as the context. This has the same limitation as the explicit position-dependent context in standard GP, though it introduces some desirable effects of redundancy and implicit reuse.

In Lones's enzyme GP, he introduces a position-independent implicit context for each component, represented as a shape signature called *functionality*. This shape signature summarizes the identification of the functional properties of the component and all its input shape signatures. The mapping from the linear genome to programs is accomplished through a self-organization process based on shape matching. The purpose is to make sure that similar shape signatures of components imply similar functional behaviors. However, there are several detrimental limitations of this approach. One is that the simple self-organization construction method used by Lones is too sensitive to the relative shape signatures among the components such that a small mutation of the functionality of the components may lead to dramatically different and strongly degraded programs, thus violating the inheritance principle. In contrast, living organisms have evolved many mechanisms to achieve robustness self-organizing developmental process. This limitation also occurs in grammatical programming where a single change of the early bits dramatically changes the interpretation of a later part of the genome, which is even worse than the case of Lones' emzyme GP as it changes the semantic of the coding. The second limitation of enzyme GP is the lack of sufficient support for modular reuse, which accounts for its failure to solve even-n-parity problem when n>3. Another limitation is that the functionality (shape of signature) of a component in enzyme GP essentially has no connection with the actual function of the component: the output of the component will never affect its exported context sensed by other interacting components. Instead, in biological development, the function of the cells regulates the physiological, chemical and physical environment that serves as the implicit context. However, Lones's work represents an important research direction toward deep understanding of how and why biological evolution works so well.

Another important work toward scalable evolutionary synthesis comes from Bentley (2003). He proposes a fractal protein approach to improve evolvability, scalability, exploitability. The motivation is to provide a rich medium for evolutionary computation to exploit. In this approach, each gene is expressed as a highly complex fractal form and the interaction of these fractal proteins implements some kind of virtual chemistry, which is expected to provide the richness and diversity for the evolutionary process to exploit. The question is whether artificial evolution can exploit any kind of complexity such as the Mandebrot set for its own synthesis purpose. Unfortunately, the fact that this fractal protein approach did not bring any significant improvement in terms of scalability since 2002 implies that there are some fundamental limitations in this approach. However, this does appear to represent one promising direction for achieving scalable evolutionary synthesis. Although he did not mention the discussion of Kauffmann (1993) on the critical role of self-organization in evolution, Bentley implicitly tried to let the artificial evolutionary process exploit existing orders in the fractal proteins he introduced into the evolutionary and developmental process in his system. This is to some extent inspired by evolvable hardware in which the evolutionary process can exploit a variety of physical properties of the evolution environment (Thompson et al., 1999). Another similar effort is from Julian Miller, who used the liquid crystal as the rich physical medium for the evolutionary process to exploit (Harding & Miller, 2004). With similar motivation of how to exploit self-organization and emergent order, Eggenberger (2003) demonstrated how genome-physics interaction can greatly reduce the number of parameters needed to specify the phenotypes.

5.5 Future Directions

From previous works on scalable evolutionary synthesis, it is observed that the fundamental principles for handling complexity must be considered at the same time to realize scalable evolutionary synthesis. There are still many aspects of biological evolutionary mechanisms that we can learn from. This section presents some critical issues or promising research topics in scalable evolutionary synthesis.

One fundamental guideline for genome representation in scalable evolutionary synthesis is the linear organization of genes, as in the DNAof living organisms. This simple structure strongly facilitates the implementation of homologous crossover, which seems to me to be one of the critical mechanisms for the scalability of biological evolution. Homologous crossover ensures that the offspring generated will resemble their parents, thus leading to high-inheritability. Actually, there is some existing work related to linear GP (Nordin, 1994; Neill & Ryan, 2001) and homologous crossover (Hansen, 2003), but it is not implemented in a developmental framework that considers support for modular reuse, and thus has not yet, I believe, demonstrated its full potential.

The second interesting topic is investigating how to introduce the modular gene organization structure into linear GP. This is inspired by the idea that genes in living organisms are all delimited by start and end codons. It is also thought provoking that each gene contains several transcription-control regions. These fundamental structural features of genes may provide ideas for artificial evolution concerning how to design a representation scheme to support modularity and a developmental process.

The third research topic is on context evolution. In both standard genetic programming and enzyme genetic programming, the structures of contexts are strongly constrained. The former, standard GP uses a simple root node type to represent the context of applying a complex module, while the latter uses a pre-defined vector to represent the context. In biological developmental processes, the whole physico-chemical environment in which the development is occurring is used as the context, including many of the chemical signal molecules such as the morphogens. Based on the critical role of morphogens in controlling the precise development of patterns in living organisms (Day & Lawrence, 2000; Wolpert, 1994; Lawrence, 2001; Patel & Lall, 2002), one interesting project to attempt would be to introduce virtual chemical morphogens to organize the developmental process in the genotype-phenotype mapping of linear GP. These morphogens would be regulated by the functions of components rather than arbitrary connections, thus implementing some kind of embodied implicit contexts. The importance of the evolution of context can also be considered from the point of view of DNA structures. It is well known that 90% of DNA does not encode precursors of mRNA or any other RNAs (Lodish et. al., 2003). Since there are rich interactions between developing phenotypes and the genomes, my hypothesis is that these non-encoding regions may serve as context information to determine which segments of the DNA should be transcribed based on current context.

Current genetic programming does not have a good way to support reuse by inheritance –deriving new component types inheriting and modifying extant component type, while cells still retain the ability to specialize into different cell types during their development. This type of reuse may be more important than simple repetitive reuse. Although gene duplication can implement a coarse form of reuse by inheritance, the trouble is that the duplicates will evolve separately. One implementation idea is to introduce an interpretation layer during the mapping from genome to phenotypes. By adding additional layers, the same genome can develop into specialized modules.

Parameter evolution remains a huge obstacle to scalable evolutionary synthesis. In standard genetic programming, each time a crossover operator moves a module from one place to another, a whole new parameter optimization process is needed to tune the parameters. A more clever way is to evolve a context-dependent parameter setting mechanism rather than treating the parameters independently and identically for each module. In this way, after crossover, modules only need a small amount of adjustment to adapt to new contexts. Koza's (Koza et al., 2003) approach for parameterized topologies can be used here. Ideas from biological shape (Day and Lawrence, 2000; Lawrence, 2001; Patel & Lall, 2002) and size control could serve as great sources of ideas. I call this mechanism the generative mechanism for parameter optimization in evolutionary synthesis.

Another promising research topic is to investigate how a self-organization process of components can be exploited to form complex patterns. It is especially interesting how to introduce some auxiliary scaffolding components into the evolutionary developmental system. This includes many housekeeping genes, signaling molecules, morphogens, etc. Most existing artificial evolutionary developmental systems do not include these organizational entities, which may be essential to build complex patterns.

Finally, I believe that the HFC sustainable evolution model can greatly improve our capability for achieving scalable evolutionary synthesis. Firstly, HFC can help to avoid premature convergence, which is much severe in topologically open-ended evolutionary synthesis than parameter optimization problems. Second, HFC can achieve some synergetic effect with evolutionary developmental systems that support modularity and reuse. One big motivation of HFC evolution model is that new modules evolved from lower levels can be used to update old modules in the solutions of higher levels. This will help to support implementation of hierarchical modularity mechanisms.

Chapter 6

STRUCTURE FITNESS SHARING FOR SCALABLE EVOLUTIONARY SYNTHESIS

Chapter 5 presented a brief examination of the principles for handling complexity, seeking to provide a better understanding of the limitations of current genetic programming for scalable synthesis. However, as of this writing, there has been no highly competitive technique among those GP methods that aim to improve scalability by introducing developmental mechanisms, implicit contexts, or linear genome representations. At the same time, human-competitive results are continually produced with current genetic programming techniques. It is thus worthwhile to study issues involved in improving current GP-based evolutionary synthesis paradigms, even if they do not promise to yield all of the features desired for scalable synthesis. Techniques useful for improving the existing systems may also apply to future evolutionary development-based synthesis systems. This chapter investigates the issue of maintaining balanced simultaneous topology and parameter search in evolutionary synthesis to achieve sustainable topological search capability and proposes an effective technique named Structure Fitness Sharing (SFS) to address it. The issue of representation or encoding will be investigated in the next chapter in the context of evolutionary synthesis of bond graph-based dynamic systems.

6.1 Balanced Topology and Parameter Search in Evolutionary Synthesis

The objective of topologically open-ended synthesis is to find a solution system with a pre-specified desired functionality. To achieve this, both the topology and the parameters of the system must be searched, which leads to the problem of simultaneous topology and parameter search. One unique feature of this process is that rather than depending on domain knowledge to determine the goodness of a topology, we can only evaluate its performance by simulating a system with this topology configured with a set of appropriate parameters, which are determined by calling a parameter search procedure. In the case of GP-based evolutionary synthesis, the population is composed of individuals with different parameters and different or identical topologies. If a parameter operation is applied to parent individuals, it will generate offspring with identical topology but different parameters. If a topology operation is applied to parents, offspring with different topologies will be created.

Now comes the dilemma. To find better topologies, we want to sweep more topology space and evaluate more topologies, which means that we need to modify the topological structures of the individuals in the population more frequently. However, this runs the risk of discarding potentially good topologies simply because insufficient parameter search has been applied to them to expose their true value of performance. Since for each topology, a certain amount of parameter search is needed to evaluate its fitness, discarding a promising topology usually means wasting much of the parameter and topology search effort applied to reach that particular topology. On the other hand, if too much parameter search is applied to a topology, it may turn out to be another kind of waste of computing – unnecessary evaluations are applied to mediocre topologies and we suffer from insufficient topological exploration. Another risk is that if too much parameter search is applied to some individuals with big progress in fitness, these individuals tend to generate more offspring with identical topologies in the population and impede the existence of individuals with alternative topologies whose potentials are yet to be discovered. The effect of this phenomenon is what we have called *premature structural convergence*.

Premature convergence of topologies in GP-based evolutionary synthesis is partially caused by neglecting the difference between topology and parameter search. In standard GP, nodes at which to perform crossover and mutation are selected randomly from the entire set of nodes, treating those specifying topology modifications identically with those specifying numerical modifications (provided that numerical subtrees are used to define the parameters of components, as is often done). This means that a new circuit topology is often discarded by the selection process if its fitness is low with its initial set of parameters. The result is that often, topologies of moderate quality with slightly better-fitted parameters proliferate and dominate the population, while inherently better topologies with bad parameters are discarded. Ideally, a topology should be discarded only when it is demonstrated to be bad after a sufficient effort to adjust its parameters. In addition, since there are often many more numeric nodes than topology modifying nodes, premature structural convergence is accentuated, since there is a lower probability of choosing a topology modifying node that will generate a new topology than of choosing a node that changes only parameters. This sounds reasonable since it is desirable to allocate more parameter search than topology search. The problem is that as the allocation of parameter search is not controlled, usually a few topologies with sufficiently good parameters tend to dominate the population.

The balanced topology and parameter search issue has received limited attention in the past years. Koza and his colleagues handle this problem by a deliberate parameter setting in their experiments. For example, in (Keane et al., 2002), they set 63% one-offspring crossover on internal points and 7% on terminals, 1% subtree mutation and 20% mutation on numerical constant terminals and 9% reproduction. This parameter setting implicitly tries to ensure a balanced numeric search and topology search. However, this coarse scheme cannot control how parameter search effort is distributed to different topologies. It thus still has high risk of premature structural convergence when some topologies receive too much parameter searching while others receive too little. This approach cannot ensure both topology diversity and parameter diversity for healthy evolution. Another serious concern of the balanced topology and parameter search issue is discussed in the NEAT (Neural network Evolution with Augmented Topologies) approach by Stanley and Miikkulainen (2002). Similar to the method proposed in this chapter, NEAT also tries to apply fitness sharing to topologies to encourage and maintain topological innovation capability and avoid premature structural convergence. NEAT defines a clever way to avoid the difficulty of measuring the distance between two complex topologies by measuring the distance of their linear genotypic representations. It first aligns two linear genomes using historical marking of genes and then counts the discrepancies of corresponding genes. NEAT has shown great success in neural network evolution. However, the effectiveness of NEAT depends on the assumption that there is strong correlation of genotypic similarity and phenotypic (topological or functional) similarity, which holds in the domain of neural networks. In other systems, even small genotypic modification usually leads to large change of behaviors and NEAT may have trouble. This is usually the case in analog circuits or bond graph-based dynamic systems, where the components are not homogeneous like the neurons used in neural networks.

There has been some research in standard genetic programming related to the premature structural convergence issue of GP-based evolutionary synthesis. The premature convergence of program tree structures is a well-known issue in the GP research community (Burke et al., 2004; Daida et al., 2004) and has been analyzed through theoretical analysis and visualization techniques (McPhee and Hopper, 1999; Daida, 2004). Several approaches are proposed to address this premature convergence problem. Most of them are derived from GA, but with some specific consideration of the GP context. Rodriguez (Rodriguez-Vazquez,

1997) uses a multi-objective GP approach with fitness sharing being performed in the fitness space to maintain the diversity of the population. Though easier to implement, it remains an open question whether diversity of fitness values is generally a true indicator of the diversity of a population – a measure that should actually be based on the phenotype space. De Jong et al. (2001) use a multi-objective method to explicitly promote diversity by adding a diversity objective. In their method, a distance measure defined as follows is used in the diversity objective. The distance between two corresponding nodes is zero if they are identical and one if they are not. The distance between two trees is the sum of the distances of the corresponding nodes – i.e., distances between nodes that overlap when the two trees are overlaid, starting from the root. The distance between two trees is normalized by dividing by the size of the smaller of the two trees. The diversity objective is defined as the average squared distance to other members of the population. An improved version of the above distance metric between two trees is proposed in (Ekart & Nemeth, 2000) and used to do fitness sharing in GP. Their method includes the following three steps:

1) The two GP trees to be compared are brought to the same tree-structure (only the contents of the nodes remain different).
2) The distance between each pair of symbols situated at the same position in the two trees is computed.
3) The distances computed in the previous step are combined in a weighted sum to form the distance of the two trees.

The major improvement of this method is that it differentiates the types of nodes when calculating the distance between two nodes. It first divides the GP functions and terminals into several subsets. For nodes with types belonging to the same subset, it calculates the relative distance. For nodes with types belonging to different subsets, it uses a defined function to make sure that the distance between nodes from different subsets is larger than that between nodes of the same subset. It also considers the fact that a difference at some node closer to the root could be more significant than a difference at some node farther from the root, using a multiplier K to distinguish them. Edit distance and phenotypic distance for fitness sharing for GP are also tested in their experiment. The former gets slightly better accuracy but with relatively high computational cost. The latter doesn't provide much improvement over the original GP without fitness sharing. Implicit fitness sharing (McKay, 2000) has also been applied to GP. Instead of calculating the distance between the structures of GP trees, it is a kind of phenotypic (behavior-based) fitness sharing method. The fitness is "shared" based on the number of other individuals who have similar behaviors, capabilities or functions. Implicit fitness sharing provides selection pressure for each individual to make different predictions from those made by other individuals.

However it is clear that the program tree structure as the genotypic is somewhat different from the phenotypic structures, namely the topologies of candidate design solutions, that we talk about in evolutionary synthesis. Here in a GP population, topology diversity is needed to enable efficient topology exploration, which is the main objective, in most case, for discovery of innovative designs. At the same time, the goodness (or fitness) of a topology can only be evaluated after sufficient parameter exploration within the same topology. Thus, the parameter diversity of each topology also needs to be maintained. In the context of variable structure and parameter design by GP, the population diversity has some significant differences from that of a GA, in the following respects:

- **Number of peaks**

  When applying fitness sharing in GA, two assumptions are made. One is that the number of peaks is known or can be estimated. The second is that the peaks are almost evenly distributed. In many problems of GA, a relatively limited number of peaks is expected to enable efficient use of fitness sharing. However, in TYPE III problems, each structure may have a huge number of peaks with respect to its parameter space, while in the topology space, each topology is a distinct peak, since the topology space is not a continuous space, but rather a highly nonlinear discrete space.

- **Continuity of search space**

  In GA, many problems can be considered as defined in an approximately continuous space, although sometimes certain aspects have distinctly discrete behavior. However, in TYPE III problems, GP deals with a highly discrete topology space that also has a huge continuous space (of parameter values), since for each topology, the search for appropriate parameters can be regarded as an instance of GA search.

- **Constraints**

  In GA, only parameter constraints exist. However, in TYPE III problems, GP must deal with both topology constraints and parameter constraints.

The demand for topology diversity as well as parameter diversity makes the existing fitness sharing methods inefficient for effective evolutionary synthesis. The fitness-space-based fitness sharing (Rodriguez-Vazquez, 1997) and the implicit fitness sharing (McKay, 2000) methods fail to maintain sufficient parameter diversity since they do not promote coexistence of individuals with the same topology but with different parameters in order to enable efficient parameter search. Fitness sharing with the distance metric, as in (Ekart, 2000; DeJong, 2001), is also inefficient in this case. First, the computational cost is still demanding, since a complex topology and its parameters often require a big tree – perhaps 1000 - 2000 nodes in most of our experiments – especially when parameters are normally represented by a numeric subtree such as Koza uses (Koza, 1999). Second, but more importantly, the underlying assumption of the above distance metrics is that structural dissimilarity measured between two GP trees meaningfully reflects the dissimilarity in function between the two topologies. However, as the topology space represented by a GP tree is a highly non-linear space, in most cases, a change of a single (non-parameter) node changes the behavior of the GP tree dramatically. This phenomenon can be traced to the weak causality of GP (Rosca, 1995), which means that small alterations in the underlying structure of a GP tree cause big changes in the behavior of the GP tree. So measuring a sort of "Hamming" distance between the structures of two GP trees to predict the difference of the behavior/function is not well founded, and thus inefficient. This makes a useful definition of a sharing radius hard to determine. It seems that distance metrics in the topology space and the parameter space and the association of a set of parameters with the topology to which they apply must be faithfully captured in order to most effectively maintain both topology diversity and parameter diversity and thereby to achieve efficient search. Therefore, given the inherent difficulty of topology/function mapping, perhaps it is counterproductive to use any structural similarity measure beyond the most basic and completely faithful one – the identity mapping: two topologies are either identical, or they are not. That is the structural distance measure used here. While it is possible to define a broader relationship that still captures identity of function (for example, if swapping of the order of two children of a node has no effect on the function computed), such definitions depend on the semantics of the functions, and were not implemented here.

From above analysis, it becomes evident that in design problems involving both variable structures and variable parameters, search must be balanced between the topology and parameters. On one hand, each topology needs sufficient exploration of its parameters to develop its potential to some extent. On the other hand, no topology should dominate the population, or it would prevent sufficient future exploration of the topology space.

In order to address this problem, topology and parameter search must be controlled explicitly. In the beginning of this research, a probabilistic method was first devised to decide whether GP would do a topology modification (crossover or mutation on a topology modifying node) or a parameter modification (crossover or mutation on a parameter modifying node). Since topology changes have a more fundamental effect than parameter changes on the performance of the system, we introduced a bias toward more parameter modifications than topology modifications by controlling the probability of selecting these types of nodes for crossover and mutation sites. The following example probabilities were defined to facilitate keeping the topology and its function stable and to allow parameters to be adjusted well enough to demonstrate the potential of a topology.

$$p_{structure\,modification} = 0.1$$
$$p_{parameter\ modification} = 0.9$$

We also used explicit control of the node selection process to achieve balanced parameter evolution for all parameters in a topology. During the parameter modification stage, we first established a list of all variables whose values needed to be established during evolution, then we randomly selected a variable as the current variable to be changed. We then selected a node in the numeric subtree of this variable and did a crossover or mutation operation. In this way, each variable had an equal opportunity to be changed during evolution. This improvement sped up the evolution of balanced numeric subtrees. All variables tended to have numeric subtrees with similar depths. Even with the methods above, premature structural convergence still often occurred as topologies with well-fitted parameters quickly dominated the whole population. In this context, the Structure Fitness Sharing (SFS) method was proposed to control the reproduction of high-fitness topologies. Our assumptions are that fitness sharing can profitably be based on the number of individuals with identical topologies, and that distance between the structures of two GP trees with distinct structures is not generally an adequate predictor of the differences between their behaviors. Thus, any "counting of positions where the trees differ" distance metric is not well founded. Instead, a simple labeling technique is used to distinguish structures.

6.2 Structure Fitness Sharing (SFS)

Structure Fitness Sharing is the application of fitness sharing to structures in GP. In contrast to the GA fitness sharing using a distance measure to identify peaks, in SFS, fitness sharing is based on the tree structures, treating each tree structure in GP as a species in the space of parameters and structures.. Some GP trees labeled as distinct topology species may turn out to encode identical topologies. But the overall density of trees not recognized as encoding for identical phenotypes is low and is expected to be insignificant to the effectiveness of SFS, as is corroborated in our experiments.

In SFS, each topology is uniquely labeled, whenever it is first created. So long as GP operations on an individual do not change its topology, but only its parameters, the topology label of this individual is not changed. Parameter crossover and mutation, or replication of the individual, simply increase the number of individuals with this topology (label) in the population. If topology modifications are conducted on an individual that change the topology – for example, we change a Rep_C (a GP function node replacing a resistor or inductor of the circuit with a capacitor) to a Rep_I (a GP function replacing a resistor or capacitor of the circuit with an inductor) node – then a new topology label (structureID) is created and is attached to this new individual. Our assumption is that the possibility that any particular topology-altering operation produces exactly the same topology possessed by other individuals in the current population is relatively low, so it is not necessary (or worthwhile) to check a new topology against all other existing topologies to see if it is identical with one of them (and so could use its label), although a hashing technique might make this relatively easy to do. Furthermore, even if some newly created individual shares the same topology with another individual but is labeled with a different topology label, the algorithm is not strongly affected, so long as it occurs infrequently.

In standard GP, individuals with certain topologies will prosper while others will disappear because of their low fitnesses. If this process is allowed to continue without control, some good topologies (usually one) tend to dominate the population and premature convergence occurs. To maintain diversity of topologies, fitness sharing is applied to individuals of each topology. SFS decreases the fitness of the individual as follows: SFS penalizes only those topologies having too many individuals, according to the following fitness adjustment rule used for the experiments in this paper:

N_s: Number of topologies to be searched simultaneously

$N_{\exp}$: Expected number of search points (individuals) for each structure in the whole population

$N_{Ind_j \in s_i}^{s_i}$: Number of individuals with topology s_i (of which individual ind_j is one member)

For each individual Ind_i if $N_{Ind_j \in s_i}^{s_i} > 0.8 * N_{\exp}$, then the fitness penalty coefficient is

$$k = \left(\frac{N_{Ind_j \in s_i}^{s_i}}{N_{\exp}} \right)^{-\alpha} \text{ where } \alpha = 1.5 \qquad (5.1)$$

and if $k > 1$ then $k = 1.0$

The adjusted fitness is defined as:

$$f_{adj} = f_{old} \cdot k \qquad (5.2)$$

With this method, each topology has a chance to do parameter search. Premature convergence of topologies is limited, and we can still devote more effort to high-fitness topology search.

6.2.1 Labeling Technique in SFS

A labeling technique is used in SFS to distinguish different topologies efficiently. A similar technique has been used by Spears (1994), in which tag bits are used to identify different subpopulations. Spears's result suggests that in crowding and fitness sharing in GA, we only need to decide whether or not two individuals have the same label. The added precision of the distance metric for maintaining the diverse state of a subpopulation is often unnecessary. In SFS, the label is used only to decide whether or not two individuals have the same label (i.e., topology). We use simple integer numbers as labels rather than more complicated tag bits. Ryan (1994, 1995) also uses similar labeling ideas to decide which race an individual belongs to in his racial GA (RGA).

6.2.2 Hash Table Technique in SFS

In order to keep track of all individuals with each particular label, SFS uses a hash table -- this speeds up the access to the topology by each individual when a crossover, mutation, or reproduction is conducted. Each time a new topology is created, an entry with the *structureID* (next integer) as the key is created in the hash table. The size of the hash table is controlled to accommodate at most 500 topologies in our experiments. Whenever the number of topology entries in the hash table exceeds 500, those topology entries with no individuals in the current population or with a low fitness of its best individuals and with old ages (generations since their labels were created) are removed from the hash table. The corresponding individuals of these topologies are removed from the current population at the same time.

6.2.3 The Structure Fitness Sharing Algorithm

Table 6.1shows the outline of the SFS algorithm for GP-based evolutionary synthesis:

Table 6.1 The structure fitness sharing algorithm

```
Step 1: Initialize the population with randomly generated individuals. Initialize the structure hash table.
Step 2: Assign each individual a unique label. Here a label is just an unassigned integer number incremented by one at each assignment.
Step 3: Loop over generations
    3.1 Select the parents for a genetic operation according to their standard fitness
    3.2: If current operation is an operation that changes the structure from that of the parent(s), (including crossover and mutation at structure operator nodes of GP trees)
        Create a new label for each new structure created and add the new structure item to the structure hash table.
    3.3: If the current operation is a parameter modification (mutating the parameter nodes or crossing over at a parameter node) or only replication of an existing individual, do not create a new label. New individuals inherit the labels from their parents. Update information about the structure items in the hash table, including the best fitness of this structure, number of individuals, age, etc.
    3.4: If the maximum number of structures in the hash table is reached, first purge those structures that have no individuals in the current population. If there are still too many structures, then delete those structures whose best individuals have lower fitness and high age (>10 generation, in our case) and delete their individuals in the population and replace them with new individuals formed by crossover or mutation until the maximum number of structures in hash table is kept.
     3.5: Adjust the fitness of each individual according to equation (1).
Step 4:  If stopping criterion is satisfied, stop; else go to step 3.
```

6.3 Experiments and Results

6.3.1 Problem Definition and Experiment Setting

In this section, genetic programming with the SFS technique is applied to an analog circuit synthesis problem – the eigenvalue placement problem as defined in Section 3.5. This problem was previously approached using GP without SFS (Rosenberg, 2001). In this problem, an analog circuit is represented by a bond graph model (Fan, 2001) and is composed of inductors (I), resistors (R), capacitors (C), transformers (TF), gyrators (GY), and sources of effort (SE). The developmental evolution method similar to (Koza, 1999) is used to evolve both the topology and parameters of a bond graph representation of a circuit that achieves the user-specified behavior. With this method, a set of topology modifying operators (e.g. Rep_C, Rep_I) is provided to operate on the embryo bond graph. A set of numeric operators (such as +, -, *, /) is provided to modify the parameters of the topology.

The design objective of the eigenvalue placement problem is to evolve an analog circuit with response properties characterized by a pre-specified set of eigenvalues of the system equation. By increasing the number of eigenvalues specified, we can define a series of synthesis problems of increasing difficulty, in which premature convergence problems become more and more significant when traditional GP methods are used. Since both the topology and the parameters of a circuit affect its performance, it is easy to get stuck in the evolution process, so this is a good benchmark problem for the SFS technique.

Table 6.2 Target eigenvalues

Problem 1: 6-eigenvalue problem
$-0.1 \pm 5.0j, -1.0 \pm 2.0j, -2.0 \pm 1.0j$
Problem 2: 8-eigenvalue problem
$-0.1 \pm 5.0j, -1.0 \pm 2.0j, -2.0 \pm 1.0j, -3.0 \pm 0.7j$
Problem 3: 10-eigenvalue problem
$-0.1 \pm 5.0j, -1.0 \pm 2.0j, -2.0 \pm 1.0j, -3.0 \pm 0.7j$
$-4.0 \pm 0.4j$

In the experiments below, a set of target eigenvalues was given and a bond graph model with those eigenvalues was generated. Table 6.2 shows the three sets of 6, 8, and 10 target eigenvalues used as targets for example genetic programming runs. We applied single population GP with and without SFS and multi-population GP with and without SFS to all three problem instances. Each experiment was repeated 10 times with different random seeds.

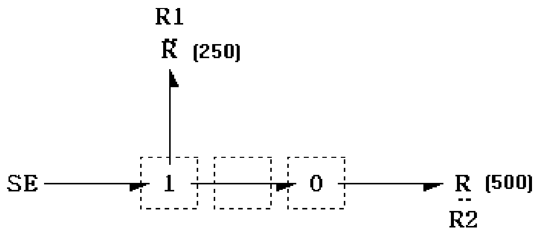


Figure 6.1 The embryo bond graph model

The embryo model used is shown in Figure 6.1. It represents an embryo bond graph with three initial modifiable sites (represented as dotted boxes). In each case, the fixed components of the embryo are sufficient to allow definition of the system input and output, yielding a system for which the eigenvalues can be evaluated, including appropriate impedances. The construction steps specified in the GP tree are executed beginning from this embryo. The numbers in parentheses represent the parameter values of the elements.

Table 6.3 Parameter settings for experiments

Parameters of Single Population GP	Popsize: 1000
	init.method = half_and_half
	init.depth = 3-6
	max_nodes = 1000
	max_depth = 15
	crossover rate = 0.9
	mutation rate = 0.1
	max_generation = 1000

Additional Parameters of Multi-Population GP	Number of subpopulations = 10; Size of subpop = 100 migration interval = 10 generations migration strategy: ring topology, migrate 10 best individuals to the next subpopulation in the ring to replace its 10 worst individuals
SFS Parameters	N_s : 50 N_{exp} : 20 = popsize/ N_s

Three circuits of increasing complexity are to be synthesized, with eigenvalue sets as specified above. The GP parameter tableau for the single population method is shown in Table 6.3 below. These problems exhibit a very high degree of epistasis, as a change in the placement of any pair of eigenvalues has a strong effect on the location of the remaining eigenvalues. Eigenvalue placement is very different from "one-max" or additively decomposable optimization problems and these problems become increasingly difficult with the problem order.

6.3.2 Experimental Results

The performance of each of the three GP approaches is reported in Figure 6.2 to Figure 6.4 for 6/8/10 –eigenvalue placement problems. Four GP methods are compared, including:

OneGP: single population GP, no SFS

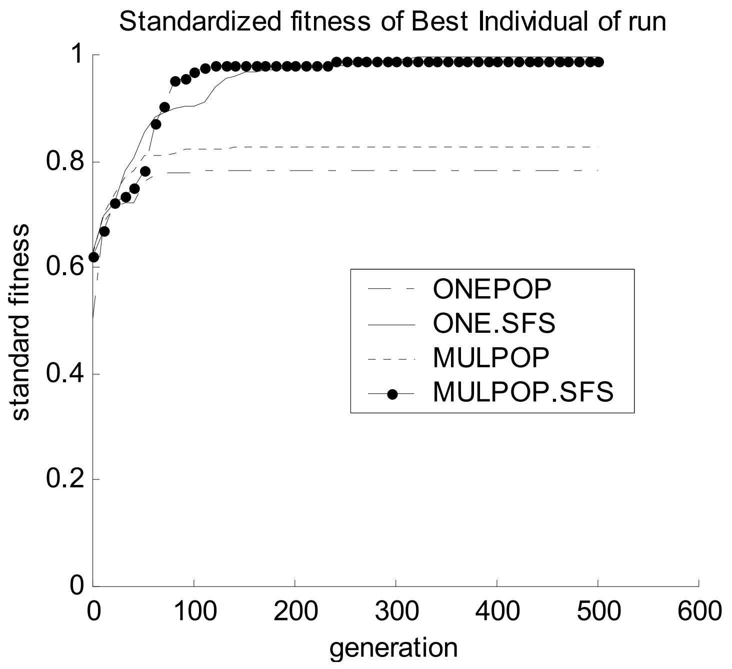
MulGP: multi-population GP, no SFS

ONE.SFS: single population GP with SFS

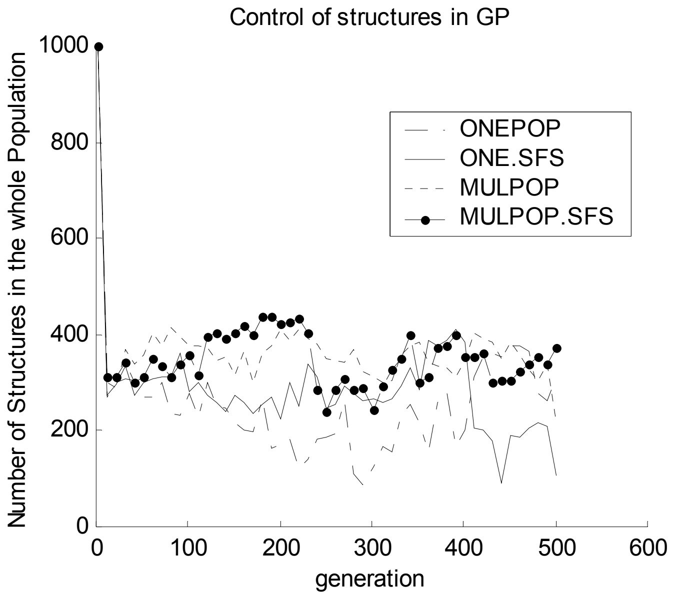
MULPOP.SFS: multi-population GP with SFS

Figure 6.2 (a) compares the four algorithms in terms of the best-run fitness of the population for 6-eigenvalue placement problem. In this figure, both single population GP and multi-population GP converge quickly in the beginning of the evolution and stagnate without any fitness progress due to the convergence of topologies. In contrast, standard GP with SFS technique maintains well the capability of topological innovation and thus achieve sustainable fitness progress for a long time. SFS thus greatly improve the fitness as indicated by the significant fitness gains between these two approaches. Figure 6.2 (b) illustrates the evolution of number of topologies (two topologies may be coincidently generated as identical) in the population. It shows that the diversity drops dramatically in the very beginning and then keeps fluctuating up and down. However, GP with SFS maintains much higher number of topologies in the population for a long time while standard GP tends to lose topology diversity later on. It is this sustained topological diversity that contributes to the discovery of higher fitness individuals from time to time along the evolution.

The experiment result for 8-eigenvalue is shown in Figure 6.3. Similar to the 6-eigenvalue problem, GP with SFS still achieves significant performance improvement compared to standard single-population GP, while doing a little better than the multi-population GP. Figure 6.3 (b) shows that SFS works by maintaining higher topology diversity in the population. The reason for the smaller performance gain for multi-population GP is that the latter has some inherent capability to maintain diversity, so also topology diversity. However, this capability is limited, as shown in Figure 6.4 (a) for the 10-eigenvalue placement problem. Here, in the case of multi-population GP, SFS is able to greatly improve the best-fitness of the population, actually dominating all the solutions of standard GP with a significant gap. What is interesting in this case is that the number of topologies in the GP with SFS is even smaller than that of standard multi-population GP. Our explanation is that multi-population GP can maintain diverse topologies by topological operations, but fails to exploit these topologies and to keep promising topologies for sufficient time to "mature" until they gain high fitness. Thus it lacks the capability of balanced topology and parameter search. The idea is somewhat similar to the concept of "healthy diversity" in genetic algorithms, in which high diversity resulting from high mutation rate doesn't lead to high performance. In this case, high topology diversity doesn't mean efficient topology search. It is the capability to explore and exploit a variety of topologies that contributes to efficient evolutionary synthesis.

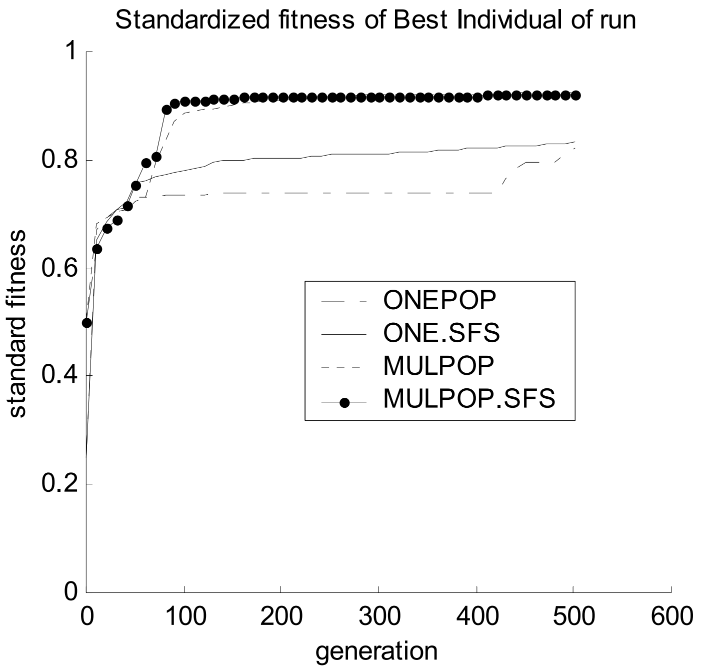


a) Comparison of best-of-run fitness

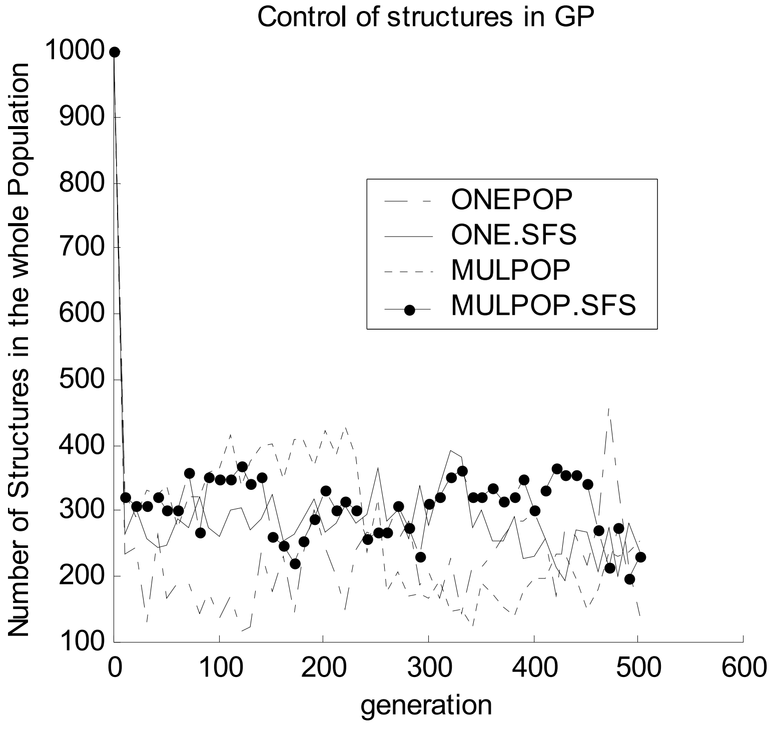


b) Number of topologies in the population

Figure 6.2 Experimental results for 6-eigenvalue problem. Comparison of SFS and standard GP in terms of the best-of-run fitness and the capability of maintaining multiple topologies in the population. SFS can greatly improve the performance of standard GP in both single population and multi-population GP model.

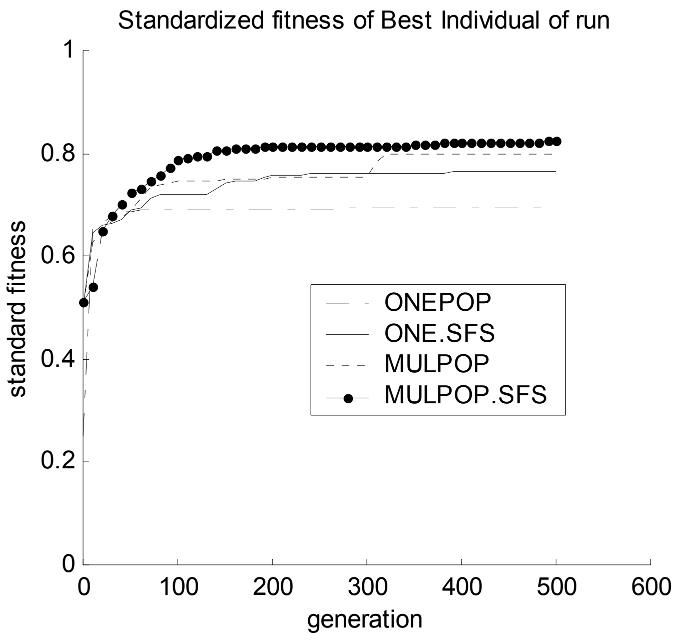


a) Comparison of best-of-run fitness

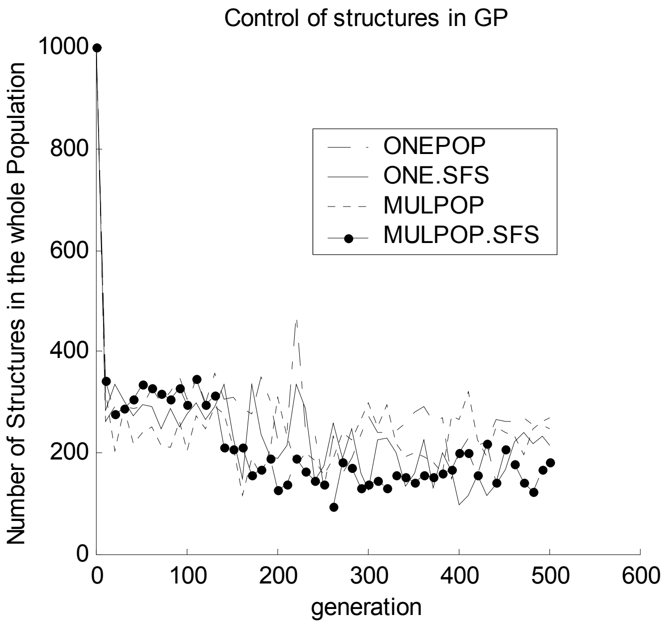


b) Number of topologies in the population

Figure 6.3 Experimental results on 8 eigenvalue problem



a) Comparison of best-of-run fitness



b) Number of topologies in the population

Figure 6.4 Experimental results on 10 eigenvalue problem

6.4 Summary

In this chapter, a speculative review of the scalability problem in evolutionary synthesis and potential research directions are presented, illustrating critical limitations of the current GP-based evolutionary synthesis scheme. While the fundamental remodeling of GP remains to be invented, this chapter identifies the issue of balanced topology and parameter search in evolutionary synthesis. In biological evolution, if we regard each species as a distinct topological structure, the balanced simultaneous topology and parameter search problem or the maintaining of the capability for topology innovation is handled by the variety of physical environmental niches. While environmental niches are not typically modeled in genetic programming, this chapter introduces the fitness sharing scheme to deal with the balanced topology and parameter search problem.

Structure Fitness Sharing (SFS) is proposed to achieve balanced topology and parameter search in evolutionary synthesis. SFS turns out to be able to effectively prevent the dominance of any specific topology and, when combined with probabilistic control of topology and parameter modification, SFS can maintain a stable number of topologies for simultaneous topology and parameter search. The labeling technique in SFS eliminates the necessity of computing the distance between two individuals, which saves computing effort that we believe is often largely wasted when attempting to measure GP structural similarity. The user parameters of the standard fitness sharing method are also eliminated (e.g., the sharing radius). All that must be done is to define the fitness adjustment scheme: how to penalize the fitness of a topology when the number of individuals with that topology label grows large enough to threaten the diversity of the population. The hash table technique allows SFS to quickly update the topology information about the current population during evolution.

Based on the concept of "healthy" topology diversity, more complicated balanced topology parameter search methods can be derived. For example, the *age* concept of topologies may be incorporated into SFS such that no topology can linger in the population for too long, which can promote the opportunity for topology innovations. Another potential improvement is to apply a fast local parameter search algorithm to evaluate the goodness of each topology, thus explicitly distinguishing topology and parameter search. The benefit is that we then have more control over allocation of search efforts between topology search and parameter search.

Chapter 7

REPRESENTATION IN GENETIC PROGRAMMING FOR EVOLUTIONARY SYNTHESIS

Scalability of genetic programming for evolutionary synthesis depends on many aspects of the evolutionary system, including the evolution model as discussed in Chapter 3 and the balance of topology and parameter search discussed in Chapter 6. Another important aspect of any evolutionary system is the representation or encoding method: how should one encode genomes and map genotypes into phenotypes to achieve scalability? In genetic programming, the question is how to choose the primitives/building blocks or the function set to achieve high performance? This chapter investigates the relationship between representation, evolvability and scalability.

7.1 The Representation Problem in Evolutionary Computation

Representation of the genotypes along with their mapping to phenotypes is one of the three essential components of any artificial evolutionary system. Actually, representation is the most distinguishing feature that defines different genres of evolutionary algorithms such as genetic algorithms (Holland, 1975), evolution strategies (Rechenberg, 1974), evolutionary programming (Fogel et al., 1966), genetic programming (Koza, 1992), linear genetic programming (Nordin, 1994), grammatical programming (O'Neill & Ryan, 2001), and gene expression programming (Ferreira, 2001). Representation usually determines how one should design an appropriate set of genetic operators to achieve best performance. Although some generic genetic operators are available, without understanding of the underlying genome representation, there is no way to guarantee good performance. Representation may also affect how to select reproductive plans (or algorithms) which specify how to apply genome manipulating operators to representations in order to form executable evolutionary algorithms. Due to the simplicity of theoretical analysis for binary problems, many researchers have worked on the representation problem in genetic algorithms (Mathias & Whitley, 1994; Whitley et al., 1998; Rothlauf & Goldberg, 2002a, 2002b; Yu & Miller, 2002; Rothlauf & Goldberg, 2003). Although results from their research come from genetic algorithms, some general guidelines and principles also emerge. For example, Rothlauf and Goldberg (2003) examined the effect of redundant representation on the performance of genetic algorithms. Their conclusions include:

1) Only synonymously redundant representations (all genotypes encoding the same phenotype are similar to each other) may improve performance.
2) Uniformly redundant representations do not change the performance of genetic algorithms
3) If the optimal building blocks are over-represented, GA performance increases. Otherwise, GA performance decreases.
4) If there is no *a priori* knowledge about the optimal solution, use a uniformly redundant representation

It is clear that a good representation is only available with understanding of the specific problem.

Besides considering the redundancy in representation, Rothlauf and Goldberg (2002a) also gave some general guidelines for designing a good representation for evolutionary algorithms:

1) Coverage: representation should encode all feasible solutions. All genotypes, particularly those generated by crossover and mutation, should also represent feasible solutions

2) Locality: neighboring phenotypes should correspond to neighboring genotypes so a mutated genotype should usually represent a solution similar to that of its parent.

3) Heritability: Offspring of crossover should represent solutions that combine substructures of their parental solutions – i.e., building blocks should tend to be preserved from parents to offspring

4) Bias: representation should be unbiased (all possible individuals are equally represented in the set of all possible genotypic individuals) or biased to optimal solutions if knowledge of optimal solutions is available.

These guidelines and principles are very important in designing an evolutionary synthesis system. In the next section, these principles will be used to analyze the representation in GP-based evolutionary synthesis of bond graphs, despite the fact that crossover and mutation in genetic programming are quite different from genetic algorithms in the open-ended evolutionary synthesis.

7.2 The Representation Problem in GP-based Evolutionary Synthesis

The representation issue in genetic programming is much less investigated than in genetic algorithms, due to the complexity of variable-length representations and primitives of multiple types. The most intensively studied aspect of representation in GP is how tree-structure leads to the bloating problem (Luke, 2002). Another important decision in formulating GP systems is the selection of function set. In a symbolic regression problem (Johnson et al., 2000), it is suggested that a large function set can improve GP performance. However, Koza (1992) warned that including too many extraneous functions might cause GP performance to degrade. A recent study by Wang and Soule (2004) of this problem gives some interesting results. They use different function sets selected from a repository and test which group works best. They found that function primitives can be classified into different functional groups, and for the even-parity problem and two symbolic regression problems, they showed that a function set which has exactly one member from each functional group gives the best performance. This is an interesting concept. However, their explanation that additional unnecessary functions increase the size of the search space is not satisfactory, since the cardinality of the open-ended search space of programs is not determined by the function set at all. The complexity of the solution depends on the problem itself, given a sufficient alphabet in the function set. My explanation for the observation that the function set with one member from each group works best is the following. For the two tested problems, we have the following classification of groups: for even-parity problem, {NOT, XOR}, {AND, OR}; for regression, {+,-}, {*,/},{sin, cos, tan, gtan}, {iflte}. It is interesting that the groups {NOT, XOR}, {AND, OR}, {+,-}, {*,/} all tend to have inverse functions. When these inverse functions are included in the function set and thus mixed in the program tree, the mutation or crossover operators have higher probability of "flipping" from one function to an inverse function, thus dramatically changing the behavior of the individual and violating the inheritability and locality principle discussed in the previous section.

One limitation of all previous research on function set selection is that they all use the even-parity and symbolic regression problem, in which all nodes share the same type and can exchange freely. In GP-based evolutionary synthesis, usually multiple types of functions and terminals are employed and this strongly-typed GP may have different characteristics in terms of relationships between function set and the GP performance.

One of the puzzles in GP is that there are only vague guidelines as to how many and what types of function set are appropriate for a given problem, and whether adding more GP functions is helpful or not. One guideline is that it is preferred to find a suitable high-level set of functions (encapsulating the common modules in a problem domain) plus a set of primitive functions. It is believed that using only low-level functions may lead to extreme difficulty to evolve even essential modules in a problem domain. The function sets are usually chosen such that solutions can be constructed by a relatively small GP tree (Seo et al., 2003). In the following sections, I investigate how the GP function set affects performance in the context of bond graph synthesis by genetic programming. In particular, I examine whether it is appropriate to incorporate high-level modules into the function set to speed up evolution, as in many cases we have some problem-specific knowledge and a set of modules is available.

7.3 Three Approaches to Evolve Bond Graph-Based Dynamic Systems

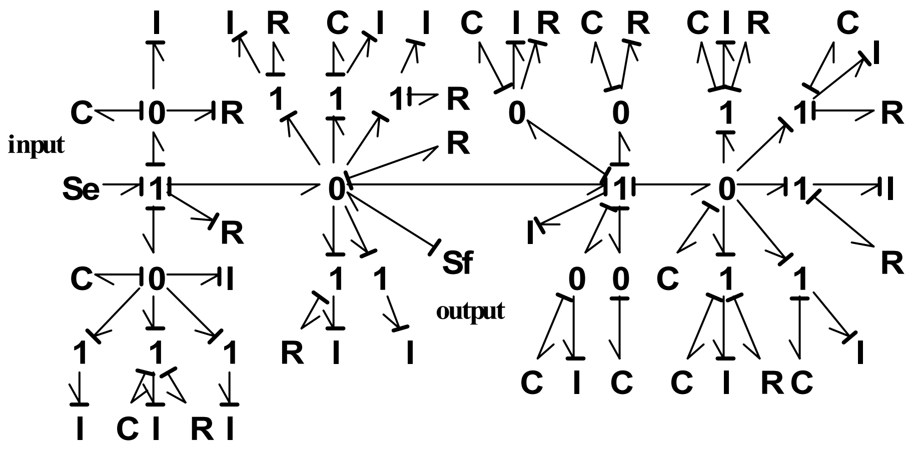


Figure 7.1 A causally well-posed bond graph of a low-pass analog filter (parameters omitted)

As a general modeling approach, bond graphs can be used to represent arbitrary dynamic systems. However, here we are interested in evolving the so-called causally-well posed systems. There are two reasons. Firstly, causally ill-posed systems may have some difficulty to be implemented physically in the mechanical or other energy transformation domain (Karnopp et al., 2000). Next, our bond graph simulator can only simulate causally-well posed systems. In this chapter, we explore the GP representation issue in automated synthesis of a special kind of causally-well posed systems – a canonical form of causally-well posed systems. In these bond graph models, all 1-junctions/0-junctions are only connected to junctions of opposite type and no duplicate C/I/R elements are allowed to be attached to a junction. For each junction, at least one and at most three element can be attached, each with its unique type. Figure 7.1 shows such a causally –well-posed bond graph model. In the context of automated synthesis, causally –well-posed bond graphs have another nice property: if an embryo bond graph is causally well-posed and each topology manipulating operator can maintain the causality after their operation, then the final system is guaranteed to be causally –well-posed. Since bond graphs are typically applied in mechanical, hydraulic, and other multi-domain physical systems, it is justified to investigate how GP-based evolutionary synthesis can contribute to designing causally –well-posed systems in addition to our previous exploration of synthesizing general bond graphs (Seo, 2002; 2003).

7.3.1 Basic Set Approach

In Section 3.4, a basic set approach is introduced to synthesize generic bond graphs. In this approach, each insertion/replacement/deletion of a basic element is implemented as a GP function. This straightforward application of GP to bond graph synthesis has successfully evolved some interesting designs, including analog filters (Fan et al., 2001), an improved typewriter drive mechanism (Seo et al., 2003a), and an air pump (Seo et al., 2003b). However, no design solutions to significantly complex problems have been evolved and scalability remains as a major issue.

After close examination of the evolved solutions with this basic approach, it turns out that these designs are usually highly redundant, with many physically meaningless local structures. We need to, and do, conduct a simplification procedure to remove those redundant structures (Figure 7.2). This redundancy is partially derived from the bond graph modeling method, in which the building blocks of 1-junction and 0-junction are not physical entities. For example, two adjacent 0-junctions or 1-junctions can be simplified to only one such junction. Another redundancy comes from the fact that multiple copies of the same element type, like C/I/R, can be attached to the same junction, but are equivalent to a single such element. Since redundancy may unnecessarily increase the search space, it is thus desirable to find a method to evolve bond graphs without such redundancy.

In the next sections, two modular set approaches are proposed to evolve causally – well-posed bond graphs without redundancy. A preliminary similar approach is studied in (Seo et al., 2003a), and shows that the modular set approach can effectively remove the redundancy phenomenon, but its performance gain is not clear. Although we know that factors such as the embryo, the function set, and the types of modifiable sites can all influence the search efficiency, we have no clear idea of precisely how they affect it and what we can do to make better decisions.

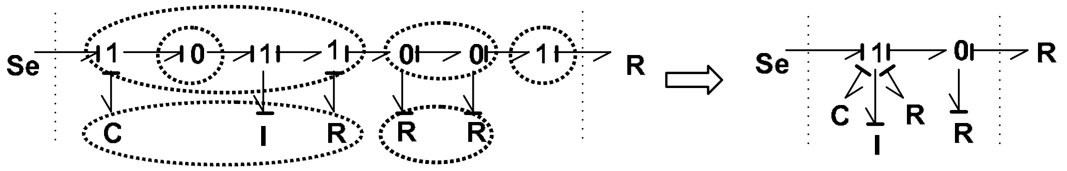


Figure 7.2 Example of a redundant bond graph model and its simplified equivalent (right). The dotted lines represent the boundary of the blocks that can be merged or simplified.

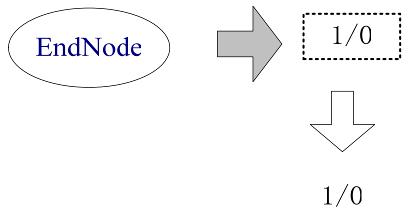
To investigate how representation affects evolvability, two modular set approaches for bond graph synthesis are proposed in the following sections. They are devised based on the concept of causally well-posed bond graphs: a bond graph in which two junctions of the same type (0- or 1) are never adjacent to each other, each element (I/R/C) is connected to a junction and all I, R, or C elements attached to the same junction are coalesced to a maximum of three elements with the types of I, R, and C. Such bond graphs are physically meaningful, at least in some domains. These modular set approaches greatly reduce the number of GP functions while still ensuring the capacity to find solutions to a set of test examples.

7.3.2 Node-Encoding Approach

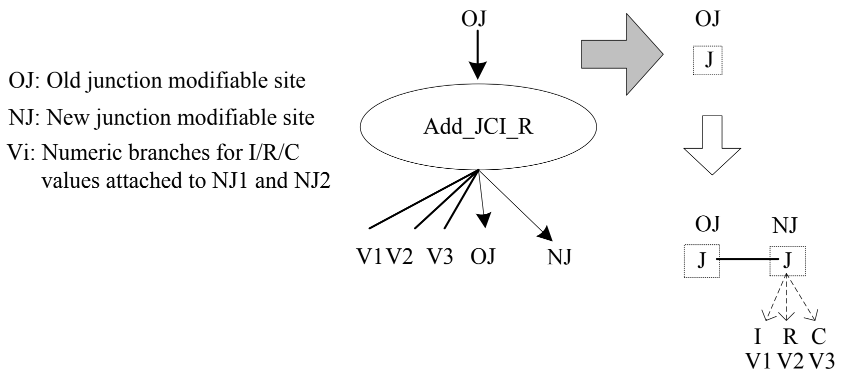
In the node-encoding approach for bond graph synthesis by genetic programming, only one topological manipulation function (Add_J_CI_R) is used in the function set, augmented only by the growth termination terminal, EndNode (Figure 7.3). So the function set for the node-encoding approach is:

F={ Add_J_CI_R, EndNode, ERC}

where ERC represents a real number that can be modified by Gaussian mutation. In this chapter, in order to focus attention purely on the evolution of topologies, all ERCs are fixed at a value of 1.0.

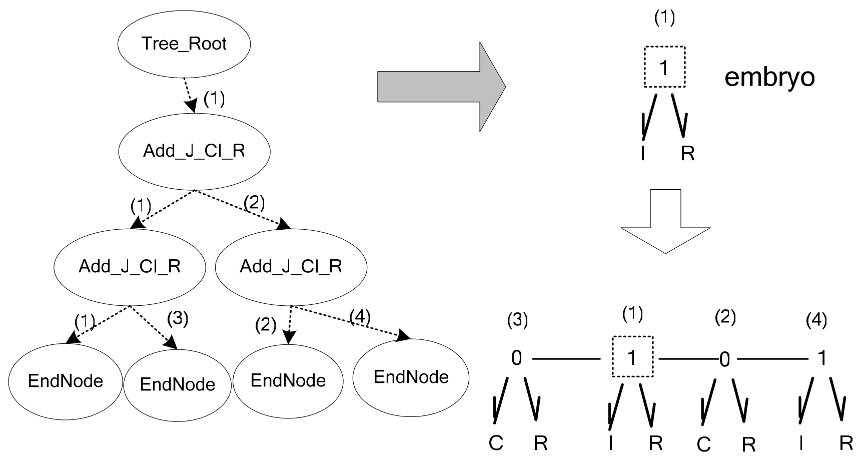


(a)



OJ: Old junction modifiable site
NJ: New junction modifiable site
Vi: Numeric branches for I/R/C values attached to NJ1 and NJ2
OJ
Add_JCI_R
V1 V2 V3 OJ NJ
OJ
J
OJ NJ
J J
I R C
V1 V2 V3

(b)



(c)

Figure 7.3 GP function set for node-encoding approach. (a): the EndNode terminal simply stops the growth from the input modifiable site; (b): the Add_JCI_R function; (c): an example of the GP tree, composed of topology operators applied to an embryo generates a bond graph after depth-first execution (numeric branches are omitted).

There is only one type of modifiable site, the junction (or node) site, in the node-encoding approach. Function Add_J_CI_R works by first checking the junction type of the incoming modifiable site; if it is a 1-junction, it will attach a 0-junction to it, and vice-versa (Figure 7.3). The number of elements attached to a junction is determined by a two-bit flag inside the Add_J_CI_R node in the GP tree (Figure 7.4). A *flag mutation operator* is introduced to the GP procedure to evolve these flag bits. To ensure the causality of the bond graphs developed by execution of the GP tree, we mandate that a 0-junction have only a C element and/or an R element attached to it and a 1-junction have only an I element and/or an R element attached to it. Each Add_J_CI_R function has two branches, with two junction modifiable sites and three numeric branches for establishing the values of the attached I/R/C elements. When the switch for an element (such as C) is off, the element is not present, and the corresponding numeric value is left unused. It is clear that the number of topology manipulation GP functions has decreased dramatically from 11 to 2. Preliminary experiments have shows that for some types of problems, this modular set can achieve better performance than the basic set (Seo et al., 2003a).

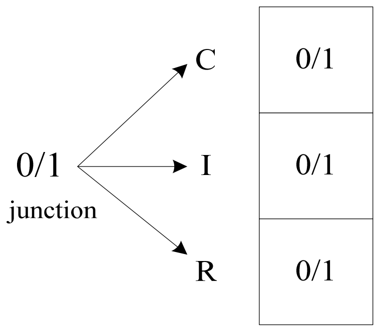
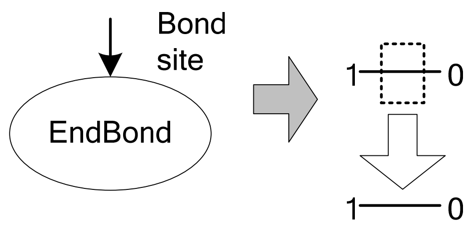
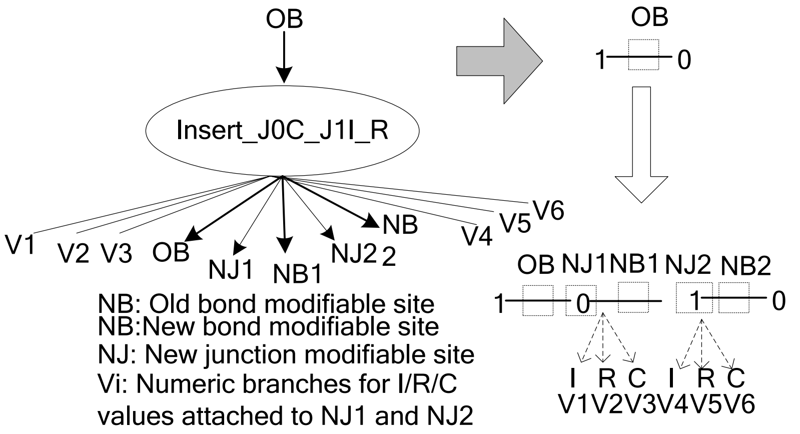


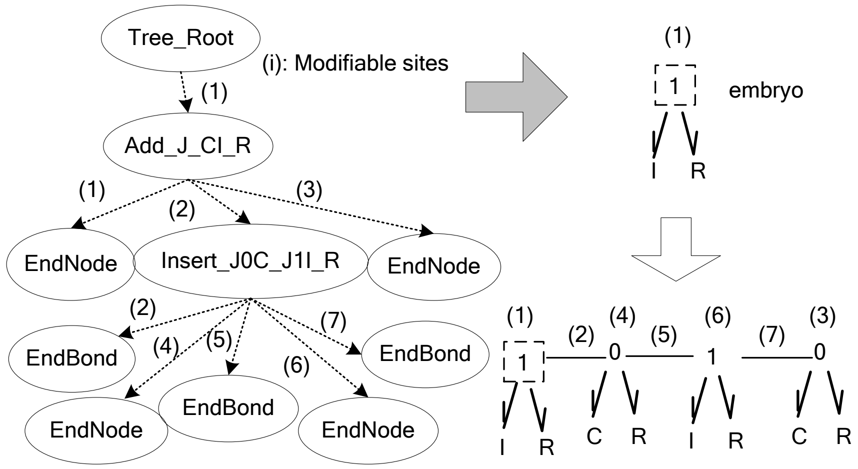
Figure 7.4 Switches for elements attached to junctions



(a)


OB
Insert_J0C_J1I_R
V1
V2
V3
OB
NJ1
NB1
NJ2
NB 2
V4
V5
V6
NB: Old bond modifiable site
NB:New bond modifiable site
NJ: New junction modifiable site
Vi: Numeric branches for I/R/C values attached to NJ1 and NJ2
OB
1
0
OB NJ1 NB1 NJ2 NB2
1
0
1
0
I R C I R C
V1V2V3V4V5V6

(b)



(c)

Figure 0.3 Function Insert_J0C_J1I_R and terminal EndBond in hybrid encoding approach for bond graph evolution. (a) EndBond terminates the growth at a modifiable site; (b) Insert_J0C_J1I_R function and its effect (c) a GP tree with hybrid encoding and its result (numeric branches are omitted).

7.3.3 Hybrid-Encoding Approach

One limitation of the node-encoding approach for bond graph synthesis is that there is only one type of modifiable site, the junction type, which limits the types of embryo bond graphs that can be used. In the basic set approach, both bonds and junctions can be used as modifiable sites, achieving high flexibility to grow bond graphs.

To solve this problem a new encoding approach is introduced by adding a new GP function that can work on a bond as modifiable site. A high-level GP function Insert_J0C_J1I_R (Figure 7.5) is added to the function set of the node-encoding scheme described in the previous subsection. Then the function set of the hybrid encoding is: F= {Add_J_CI_R, EndNode, Insert_J0C_J1I_R, EndBond, ERC}. Instead of inserting one junction at a time as the Add_J_CI_R operator does, the Insert_J0C_J1I_R operator inserts a pair consisting of a 0-junction and a 1-junction, each with appropriate elements attached to it. The type of modifiable site at which this function may be applied is a bond. Since in simplified bond graphs, all junctions are adjacent only to junctions of the opposite type, this function continues to span the space of junction connections, and assures that the resulting bond graphs are not redundant. The hybrid encoding approach allows both the junctions (nodes) and the bonds (edges) to be modifiable sites, at which new structures could be inserted or attached. Such approaches are widely used by Koza and others (Koza et al., 2000; Luke and Spector, 1997). Our intuition is that this additional GP function may greatly improve the efficiency of evolutionary search. Figure 7.5 shows this additional function and the related terminal (EndBond) of the hybrid encoding, compared to node-encoding. A typical GP tree with this encoding scheme and the result of this GP tree after applying to an embryo are also provided in the same figure.

In the hybrid encoding approach above, the high-level GP function Insert_J0C_J1I_R is designed according to the general properties of bond graphs rather than to a specific problem domain. However, in many cases, the modules in a given problem domain are very specific, and then a problem-oriented set of GP functions might need to be designed, each encapsulating a particular type of module. For example, GP functions have been devised to model typical physical modules in resonators of MEMS design, which are then used to evolve physically feasible resonators represented as bond graphs (Fan et al., 2003). Since this realizable function set approach is used to evolve MEMS systems in a highly constrained problem domain, it requires high-level modules as building blocks which are hard to evolve using basic-set approach. It is hard to compare the efficiency of these types of problem-specific approaches with that of the basic approach discussed in Section 3.3.

7.4 Benchmark Problem and Experiments

To evaluate how representation affects GP performance it is desirable to have a set of scalable benchmark problems. The analog filter synthesis problem introduced in Section 3.3.2 has the limitation that it is relatively difficult to define a series of problems of increasing difficulty because the fitness has no strong correlation with the complexity of the topology. As a result, the eigenvalue placement problem is selected as the benchmark problem in this chapter. This problem formulation explicitly specifies the order of the system. It is useful and often pre-specified in real-world applications, as cost often correlates well with order. Alternative formulations could allow candidate systems of higher order than the target to receive finite error measures, but require that unmatched eigenvalues be outside the range of the target set, for example. Another interesting formulation of the problem, with many of the same properties but requiring much less computational effort, would use the coefficients of the characteristic polynomial of the state (A) matrix as the targets to be matched by those coefficients of candidate systems, thereby avoiding the cost of numerically calculating the eigenvalues. However, systems with widely differing eigenvalues can possess only small differences in characteristic polynomials, so the eigenvalues themselves are used to calculate the fitness instead of using the characteristic polynomial coefficients, with the hope of better illuminating structure/function relationships in the systems evolved.

There is another reason for choosing the eigenvalue placement problem as the benchmark problem. As discussed in Chapter 6, evolutionary synthesis of dynamic systems involves the need to search simultaneously for the topology and parameters of the system. However, to explore the issue of how topology encoding affects search efficiency it is preferable to look at a reduced sub-problem that eliminates the requirement for parameter searching by making all parameters unity (i.e., all resistors, capacitors, and inductors have numerical parameters of 1).

To ensure that the target optimum solutions indeed exist, the target eigenvalue sets are determined from assorted topologies with unit parameters for all of their components. Since parameter search is very expensive for high-order eigenvalue problems, this topology-only search defines a much simpler problem, but can still illuminate some properties of the topology search for more general problems. The function set primitives used here restrict the resulting bond graphs to having C components attached to each and only 0-junctions and I components attached to each and only 1-junctions, which results in systems readily realizable in any energy domain and which need not be checked for being causally well-posed. Each junction also has an R component attached. Of course, this is only a sub-problem of the more general case in which I components might optionally be attached to 0-junctions and C components to 1-junctions, as well.

The following experiments are conducted:

1) To investigate whether the representation (here, the design of function set) affects its performance on different types of synthesis tasks. In other words, is it necessary to design a special type of encoding for each type of problem?
2) To study the effect of introducing GP functions encapsulating high-level modules into the function set and whether it is helpful in all cases.
3) To examine how to seed the initial GP population with typical modules to increase the search efficiency in evolutionary topological synthesis.

7.4.1 Experiment 1: Search Bias in Representation

The first issue of evolutionary synthesis in an unfamiliar domain is the representation problem. It is important to check whether the representation would bias the genetic search or not. If there is no domain knowledge available, it is better to use an unbiased search so as not to restrict the range of the search in an attempt to increase efficiency; otherwise, a special problem-specific representation might be employed. For each specific representation, it is important to evaluate its search bias for different types of problems to ensure its appropriateness in the problem domain. In this section, we evaluate the search bias of the node-encoding approach for synthesis of bond graphs as described in Section 7.3.2.

Three representative target topologies with different sizes are selected to evaluate the average search effort of the node-encoding approach. The first topology is a bush, in which 0-junctions are attached to a central 1-junction. The third topology type is a chain, consisting of alternating 1-junctions and 0-junctions. The second is a mixed topology type, which is neither bush nor chain. Table 7.1 gives examples of order 8. There are many possible intermediate topologies, only one of them is selected for comparative evaluation. As described in the beginning of this section, we assume that each 0-junction has one capacitor

(C) and one resistor (R) attached, each with fixed size parameters of 1.0, and each 1-junction has one inductor (I) and one resistor (R) attached, each sized at 1.0.

Table 7.1 Three types of target bond graph topologies (the attached C, I, and R elements are omitted for simplicity)

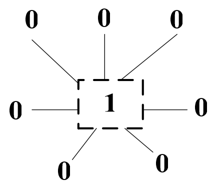
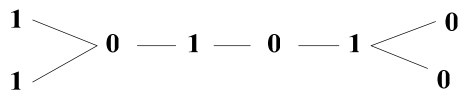
target topology	structure
bush topology	
intermediate topology	
chain topology	**1 —— 0 —— 1 —— 0 —— 1 —— 0 —— 1 —— 0**

Figure 7.6 shows the search effort to find the target topologies of each type for 8, 12, 16, 18, and 20-eigenvalue problems, using a generational GP with population size 500 for 8- and 12-eigenvalue problems, and 1000 for 16- and higher-order eigenvalue problems (with 20 runs for each target). Other primary parameters for the GP runs were as follows:

> crossover rate: 0.4; mutation rate: 0.05; tournament selection with tournament size: 7; subtree swap mutation: 0.55 (this operator chooses two separate subtrees of the same individual and swaps them, preserving system order); initialization: ramped-half-and-half initialization to generate random trees with maximum depth of 5. Maximum evaluations: 100,000 for 8- and 12-eigenvalue problems, 200,000 for 16-, 18-, and 20-eigenvalue problems.

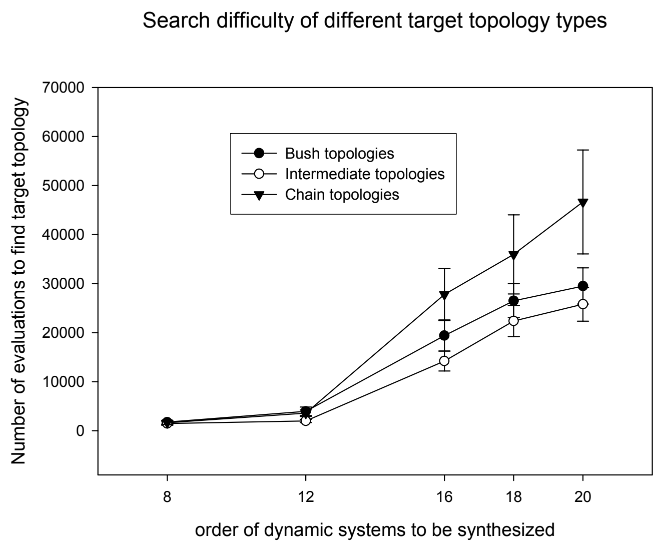


Figure 7.6 Search effort to find three types of target topologies for 8-, 12-, 16-, 18, 20-eigenvalue problems with the node-encoding approach, averaged over 20 runs

To see the effectiveness of genetic programming-based topology search, let us first consider the sizes of the topology-only search spaces of various orders, expressed in terms of distinct bond graph junction structures, which have been enumerated (Harary & Palmer, 1973). Since each 0-junction or 1-junction in the causally well-posed function set used has a determined set of attached components (the 1-junction has I and R, while the 0-junction has C and R), and since the embryo contains a fixed 1-junction, counting the junction topologies also counts the distinct bond graph topologies, and the number of junctions and order of system are identical. For systems of order four, there are only two possible bond graphs. For order 8, there are 23 different bond graphs; for 12, there are 551; for 16, there are 19,320, for 18 there are 128,340, and for 20, there are 823,065). It is obvious that the search space increases quickly with increasing numbers of components. However, since multiple GP trees map to the same bond graph topology, the *genotype* search space is actually much larger – that is, there are multiple encodings for each distinct bond graph.

From Figure 7.6, it is evident that the GP method tested can easily find the target topologies for 8- to 20-eigenvalue problems. For example, the 20-eigenvalue problem has 823,065 possible topologies, and the search procedure found the target topology with an average of 46,651 evaluations. We also found that the node-encoding approach was somewhat biased away from finding bush topologies. For all problems with order greater than 16, node-encoding used about 50% more evaluations to find the bush target topologies. Superficially, it may be surprising that the intermediate topologies take the fewest evaluations to find the targets; however, because genetic programming starts from an initial population with myriad topologies, it seems to have been easier to find an intermediate topology than those strongly biased target topologies like chains and bushes, which represent two extreme bond graph topologies. Generally speaking, node encoding solved all the 20-eigenvalue problems. This experiment demonstrated the bias of the function set that skewed its efficiency in solving certain types of problems. This is reminiscent of the underpinnings of the well-known No-Free-Lunch theorem (Wolpert & Macready, 1997).

7.4.2 Experiment 2: The Effect of Function Set on Performance

Another difficult decision in genetic programming-based synthesis is how to determine an appropriate function set. In practice, there is generally some domain knowledge available. For example, some types of components or modules appear often in specific types of problems. One straightforward way is to encapsulate many high-level modules into the function set and let genetic programming decide which kinds of modules to use in the final design. It is expected that more GP functions should help the search by providing more flexibility, or that, at least, they will not hurt the search.

To evaluate the effect of adding more high-level modules on search efficiency, we launched the same set of experiments of synthesizing three types of topologies of different orders with the hybrid encoding approach described in Section 7.3.3. The hybrid-encoding approach simply adds a new GP function Insert_J0C_J1I_R (Figure 7.5) to the function set F={EndNode, Add_J_CI_R, ERC} of the node-encoding approach used in the Section 7.3.2. This new function is expected to speed up the search for chain topologies, since it inserts a 1-0 junction pair into a bond. Since node-encoding is a complete set that is able to represent all the target bond graphs, this new set F'={EndNode, Add_J_CI_R, Insert_J0C_J1I_R, EndBond, ERC} is also complete. We used the same set of parameters as in the previous section for the GP experiments. The experimental results are shown in Figure 7.7.

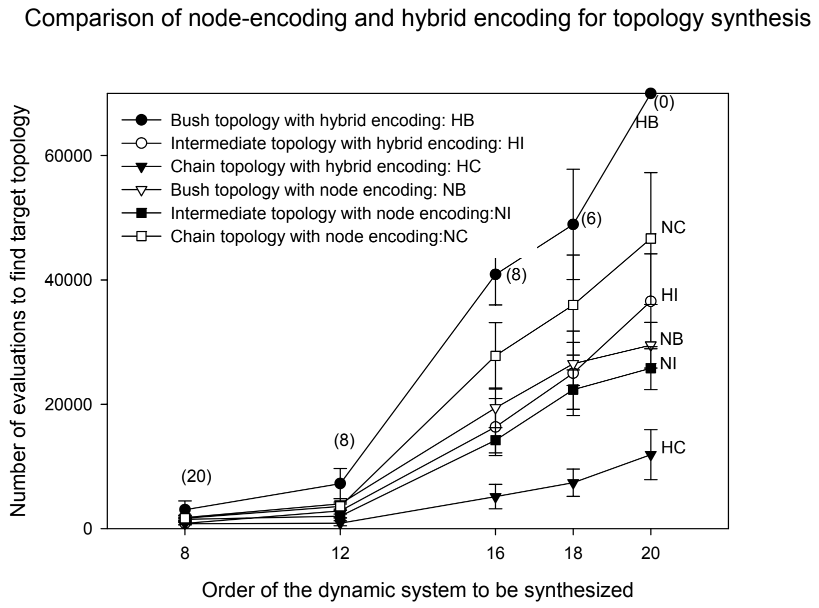


Figure 7.7 Comparison of hybrid encoding with node encoding approach for 8-, 12-, 16-, 18-, and 20-eigenvalue problems. All results are averaged over 20 runs. The numbers in brackets are the numbers of runs that successfully found the bush topology, using the hybrid encoding. The unlabelled experiments always found target topologies within 200,000 evaluations.

The first observation of Figure 7.7 is that hybrid encoding has reduced the number of evaluations needed to find the chain topology (see curve labeled HC) by a factor of three.

However, this improvement is not surprising, since inserting a junction pair into bonds can quickly construct a chain. What is more surprising is that the hybrid encoding—the simple adding of the Insert_J0C_J1I_R function—made this approach quite *unsuccessful* in evolving bush topologies, which is in sharp contrast with the improvement on chain topologies. Actually, even for 12-eigenvalue problems, the hybrid encoding only found the target bush topology in 8 runs out of 20. It is also interesting that the hybrid encoding had similar performance to node-encoding on the intermediate topologies, though with a few more evaluations. Node encoding was clearly more robust than hybrid encoding, when judged across all topologies tested.

Hybrid encoding with the GP function Insert_J0C_J1I_R allows all constructs including the bonds and junctions of the bond graph to become modifiable sites, and thus might be expected to allow more rapid or effective evolution. However, experimental results showed that this is not the case. While the hybrid encoding approach sometimes succeeded on the 8- to 18-eigenvalue problems, its bias made it unable to scale to the 20-eigenvalue problem for bush target topologies.

Seeking the source of the difficulty revealed that the hybrid-encoding approach could find a topology fairly similar to the target topology quite quickly. However, since there were two types of modifiable sites in the genome (bond and junction modifiable sites), it was highly constrained in the capability of local manipulations of the genome, because of gene incompatibility – a bond modifiable GP tree node cannot be exchanged with a junction modification GP tree node. In this case, to remove an undesirable GP node without reducing its fitness is extremely difficult, and then no genetic operations can incur large modification of the topology to help the search process jump out of local optima. It thus lacks sufficient local topology modification capability, at least in a probabilistic sense. The superficial flexibility in the phenotype space with all possible modifiable sites hides the rigidity in the genotype manipulation capability. In contrast, the node-encoding approach allows very flexible genotype manipulation, and thus is able to achieve better results.

It is apparent that the hybrid encoding method is comparable to the node-encoding approach for the easy 8-eigenvalue problems when searching for intermediate and chain topologies, but at the cost of slower search on the bush topology caused by its bias, while the unbiased node-encoding achieves similar performance for all types of topologies. Adding more high-level modules may thus be quite harmful for synthesizing some types of topologies. We also found that if one can devise a good function set based on the characteristics of the solution space, it is possible to achieve striking speed-up in search efficiency simply by modifying the representation alone.

7.4.3 Experiment 3: Knowledge-Based Initialization Improves Performance

While designing a good GP function set based on problem knowledge is a good strategy for topology synthesis by genetic programming, another approach for knowledge incorporation in evolutionary synthesis is to seed the initial population with individuals possessing certain domain-relevant properties. Nachbar (2000), for example, seeded his initial populations with different types of components whose proportions were extracted from existing molecule designs. To evaluate the impact of various population initialization methods, a set of topology transformation experiments was conducted here. We started from an initial population with identical individuals (bond graph constructor trees) whose execution generated an identical source topology—-e.g., an initial population consisting entirely of the bush topology. Then we evolved this initially uniform population toward a target topology. The experimental results are summarized in Table 7.2, with means and standard deviations. Each source-target pair experiment was repeated 20 times, all with the same running parameters as in Section 7.4.1.

Table 7.2 Topological transformation with different initial homogenous populations

target topology / initial uniform topology	mean (std. dev) evaluations		
	Bush	Intermediate	chain
bush	N/A	2825 (± 411)	3433 (± 396)
intermediate	2348 (± 471)	N/A	1864 (± 359)
chain	3183 (± 294)	2227 (± 273)	N/A

Table 7.2 shows some interesting, but not too surprising, observations. Since bush and chain topologies represent the two extreme types of the topology space, transformation between them is most difficult, while transformation from the selected initial intermediate to either chain or bush is much easier. As the intermediate topology is *between* bush and chain topologies in some sense, our results confirmed that seeding with a more similar topology will speed up the evolution. If one does not have any idea what the topology of the final solution looks like, it thus appears advisable to use one or more intermediate topologies as starting points. Another observation is that genetic programming is able to break the symmetry in the initial population and evolve toward the target topology even if it starts with the opposite topology type, such as evolving from bush to chain.

7.5 Summary

This chapter investigates the representation problem in topologically open-ended evolutionary synthesis of a canonical form of causally-well posed bond graphs. All the parameters of the candidate solutions are fixed as unity in our experiments to avoid the interference of parameter search on the topology search issues. This synthesis problem has quite different properties from other (parametric) search problems, including low neutrality, high multi-modality, discreteness, and lack of information to guide local topology search.

The experiments show that standard GP with node-encoding only could easily solve up-to-20-eigenvalue problems, but did not scale well to 24-eigenvalue problems (with a search space of 39,299,897) or higher. Two encoding schemes, node-encoding and hybrid (node and edge) encoding are compared, as applied to 8-, 12-, 16-, 18-, and 20-eigenvalue problems. The result is that node-encoding was much better than the hybrid encoding in terms of scalability and robustness. It turns out that incompatibility of the "genes" in the hybrid encoding introduces too strong a bias for efficient solution of the bush topology problem. However, this bias enables it to find chain topologies very efficiently. It shows that arbitrarily introducing many high-levels GP functions may introduce strong bias to make it hard to solve certain types of problems, while speeding the solution of other types of problems. This is a clean example of no-free-lunch theory in evolutionary computation.

With topology transforming experiments, it is confirmed that appropriate population seeding is beneficial to the search process. If faced with an unfamiliar problem, it is better to seed the population with diverse topologies. Here, parameter search which is treated in Chapter 6, is intentionally neglected. However, more systematic examination of the interaction between topology and parameter search is needed. For example, the neutrality introduced by parameter search may facilitate, rather than block, topology search.

Experiments also show that current developmental GP is powerful for topological search in design innovation. But it still lacks some essential mechanisms to scale up to even larger dynamic system synthesis problems, at least unless enormous population sizes and numbers of evaluations are used. One promising direction appears to be introduction of mechanisms for framework or module discovery and reuse, especially those mechanisms inspired by biological developmental principles (Stanley & Miikkulainen, 2003; Miller & Thomson, 2003). A second is a local genotype operator that exploits the locality of topology.

Another possible improvement may come from smarter crossover and mutation operators, rather than the random swapping and mutation of GP sub-trees.

Experiments also show that there is a strong relationship between representation/encoding, evolvability, and scalability of topologically open-ended synthesis by genetic programming. As is generally true, the evolvability of topology search depends strongly on the correlation of the topology with its fitness. The less rugged and deceptive the fitness landscape, with respect to changes caused by single applications of genetic operators, the easier the problem will be to solve. By watching the solutions evolve toward the target topology, we see a correlation between topologies and their functional behavior in the eigenvalue location problems. It is found that the 8- to 20-eigenvalue problems can be solved fairly quickly. Identifying the target bush topology from 823,065 possible topologies using only 29,506 evaluations illustrates the capability of GP for topology search. Next, the same experimental settings were used to solve 24-eigenvalue problems (with a search space cardinality of 39,299,897), except that the population size was increased to 2,000, maximum number of evaluations to 500,000, and the maximum tree depth to 30. Unfortunately, the current node encoding approach failed to find the target bush topology in any of 10 runs. This hints at one of the most severe problems of current genetic programming-based systems, the lack of scalability, especially considering that we have removed parameter search in our experiments here, which would add additional difficulty to the synthesis engine.

This failure can be attributed to several causes. One is the high epistasis and multimodality of the topology search space. The resulting low neutrality and sparseness of intermediate solutions make it extremely difficult to find the target topology. However, an interesting phenomenon is observed—-that GP usually can quickly find a reasonably good topology with high similarity to the target. But after that, there is not sufficient guiding information to fine-tune similar topologies into the target solution. One factor is that this GP system for topology synthesis lacks effective local topology mechanisms. Moreover, the standard GP, with its random crossover and mutation, is random in its choice of directions to explore, and perhaps that is too unstructured to succeed in such cases. For example, it would be desirable that, after identifying a basic framework of a possible solution, more exploration be allocated to local topology search while leaving the framework undisturbed. However, current GP randomly chooses crossover/mutation points, thus often leading to random scrambling, most of which destroys the framework discovered so far and wastes search effort. In the genotype space, we observe, for example, that although one near-optimal solution discovered was only one swap away from the target topology, GP did not find the target, because the probability to swap the two given branches is extremely low.

Chapter 8

CONCLUSIONS

This chapter presents a summary of the investigations in this dissertation toward improving sustainability and scalability of evolutionary algorithms as well as evolutionary synthesis based on genetic programming. It also discusses some future research directions based on the progress in this dissertation.

8.1 Summary

This dissertation is composed of two parts within the same context of improving scalability of artificial evolution systems. The first part addresses one of the fundamental problems in evolutionary computation, namely, the convergent nature of the traditional evolutionary computation framework. Most of existent evolutionary algorithms are derived from the common convergent evolution model: starting from a randomly generated or artificially seeded initial population, the selection operator, which biases survival toward high fitness individuals, and the genetic operators (typically, crossover and mutation) drive the whole population toward higher and higher fitness levels until stagnation occurs. This naive interpretation of Darwin's "survival of the fittest" principle of natural evolution leads to the following undesirable result: as the average fitness of the population goes increases, newly generated offspring deviating from their parents face increasingly stronger selection pressure from the population and thus cannot survive long enough to be exploited, which is essential to maintain the exploration capability for the population. Thus, only offspring with minor modifications from the parents can survive, so search capability is gradually lost. This convergent model neglects an important fact in natural evolution—that environmental niches provide support for sustainable evolution at all fitness levels, and higher-fitness living organisms typically do not threaten the existence of lower-level organisms in some of those niches. Instead, living organisms of all levels, from bacteria to *homo sapiens,* evolve simultaneously all the time. It is argued that the convergent evolution model underlies many of the fundamental defects of current artificial evolutionary systems, such as premature convergence, loss of diversity, and lack of search efficiency, sustainability, and scalability. In terms of evolutionary algorithms, it is thus necessary to maintain individuals of all fitness levels, rather than only those of increasingly high fitness, to support a sustainable evolution.

To transform convergent evolutionary algorithms into *non-convergent* search processes and thus remove a fundamental limitation, Chapter 4 proposes a generic sustainable evolutionary computation model, the Hierarchical Fair Competition (HFC) model, which can greatly improve the state of the art of current evolutionary algorithms in terms of reliability, sustainability, and scalability. The key idea of this model is to maintain individuals of all fitness levels and organize them into hierarchical fitness levels, add a random individual generator at the bottom fitness level, and keep evolution going on at all fitness levels. This structure of HFC does not allow convergence of the population to the vicinity of any set of optimal or locally optimal solutions, and ensures a continuous supply and incorporation of genetic material in a hierarchical manner. It cultures and maintains, but continually renews, populations of individuals of intermediate fitness levels. The continuing search capability of HFC is achieved by employing an "assembly-line" structure in which subpopulations are hierarchically organized by fitness levels, reducing the selection pressure within each subpopulation while maintaining the global selection pressure to help ensure exploitation of good genetic material found.

As a generic sustainable evolution model, HFC allows many different implementations and can be used to improve all kinds of existent evolutionary algorithms. In this dissertation, two general implementation approaches have been proposed to support sustainable evolution. One uses the multi-population model to segregate the competition between high-fitness individuals and low-fitness individuals. The other uses a density function of fitness to control the distribution of individuals in single population over the whole fitness range, which has no explicit levels. Five HFC-based sustainable evolutionary algorithms with different features have been derived, including the multi-population HFC algorithm, multi-population HFC algorithm with adaptive thresholds, continuous HFC algorithm, multi-population HFC algorithm for multi-objective optimization, and quick HFC algorithm with adaptive breeding mechanism. Through a number of benchmark problems in both genetic programming and genetic algorithms, it has been demonstrated that this HFC family of evolutionary algorithms can greatly improve the reliability, efficiency, and scalability for both genetic programming and genetic algorithm applications.

Chapter 5 presented a speculative survey of current research toward scalable evolutionary computation and, more specifically, evolutionary synthesis. Four major principles of handling the complexity in building complex artificial and biological systems are discussed, including the principle of modularity, the principle of reuse, the principle of context, and the principle of developmental/generative representation. These general principles set up a context in which to examine the limitations of existent GP-based evolutionary synthesis scheme. Within this understanding, several promising research ideas for improving scalability of evolutionary synthesis are discussed at the end of the chapter.

In Chapter 6 we investigated the issue of simultaneous topology and parameter search existing in all topologically open-ended evolutionary synthesis. An effective approach, called structure fitness sharing (SFS), has been proposed to address this problem. The key idea is to control the distribution of individuals to topologies and thus avoid the elimination of topologies that have not been allocated sufficient parameter search to expose their true potentials. In this way, SFS can maintain its topological innovation capability throughout the run without suffering structural premature convergence. Experiments showed that this technique can improve evolutionary synthesis performance for both single-population and multi-population genetic programming.

Finally, in Chapter 7, we explored how problem representation affects the performance of evolutionary synthesis. More specifically, we examined the role of the function set in genetic programming for topologically open-ended synthesis. Two modular approaches for synthesizing causally –well-posed dynamic systems were proposed and their performances were compared for a dynamic systems synthesis problem -- the eigenvalue placement problem. It was found that node –encoding, in which there is only one type of modifiable site and the search is thus more homogeneous and flexible in genetic operation, tends to achieve higher scalability across all problem types, including synthesizing chain, bush, and hybrid topologies. In contrast, hybrid encoding, in which both bonds and junctions are used as modifiable sites and higher-level modules are encapsulated as GP functions, introduces a strong bias into its performance. It can greatly improve the search capability for chain topology problems, but only at the cost of degraded performance for other bush-like and hybrid topology problems – a clear demonstration of the no-free-lunch theorem for representation.

8.2 Main Conclusions

Scalability and reliability of existent evolutionary algorithms are constrained by their underlying convergent evolution model. This limitation can be mitigated by the proposed Hierarchical Fair Competition (HFC) model. As a generic framework, this sustainable evolution model can be implemented in different ways. HFC model can enhance existent evolutionary algorithms with improved scalability, reliability, or efficiency in one to all three aspects. It allows evolutionary algorithm practitioners to use smaller population size, less parameter tuning, and fewer evaluations to obtain better solutions. This is true especially in highly multi-modal search problems.

Scalability of topologically open-ended evolutionary synthesis depends on many aspects. One of them is the balance of topology and parameter search. Structure Fitness Sharing proposed in this dissertation helps us to achieve such balanced search with its simplicity and effectiveness.

Problem representation plays a significant role in the evolvability and scalability of an evolutionary synthesis system. The best representation for a particular problem is problem specific. The "No-free-lunch theory" of representation applies to the design of function set of genetic programming. Good performance of a representation for one class of problems corresponds to its inefficacy for other problems.

8.3 Future Research

In Chapter 5 we suggested some general research directions based on the understanding of the limitations of current GP-based evolutionary systems. This section outlines some future research topics that are more closely related to this dissertation research on sustainable evolutionary algorithms and scalable evolutionary synthesis.

8.3.1 Characteristics of HFC, Based on Extensive Experimental Studies

The Hierarchical Fair Competition principle provides a solid foundation to achieve sustainability of evolutionary research. Its actual performance, however, relies on the implementation details. New techniques like the active breeding mechanism in QHFC are continuously being discovered. During the development of five sustainable evolutionary algorithms based on the HFC principle, some aspects of its behavior have been understood, but more details and guidelines need to be explored to realize its potential. Since HFC-based algorithms employ quite different structures and mechanisms to achieve sustainability, it is hard to extrapolate existing theories for canonical evolutionary algorithms to HFC. For example, experiments using HFC show that the population sizing theory widely described by genetic algorithm researchers does not apply to HFC-based algorithms. The parameter setting guidelines for evolutionary algorithms based on canonical evolution models also do not hold for HFC. For example, the population size used in traditional genetic programming is usually much larger than is required in HFC. In addition, the relatively complex structure and dynamics also make theoretical analysis difficult. In this situation, systematic experimental research is needed to understand the behavior of HFC algorithms and to obtain guidelines for how to use them. The following issues are the next steps of HFC research:

- Parametric study of HFC with genetic algorithms: investigate how the parameters of HFC algorithms affect their behavior; how the problem difficulty should affect our decisions about setting parameters; and what guidelines exist that parallel those in traditional evolutionary algorithms
- Characterize some classes of problems in which HFC is especially helpful compared to existing techniques.

- Investigate how to combine the HFC framework with traditional techniques to sustain evolution.

Guidelines uncovered from these research efforts will greatly enhance our confidence in applying the HFC principle widely to solve large problems.

8.3.2 Adaptive Breeding for Continuous HFC

The latest development in the HFC paradigm is the invention of a new adaptive breeding mechanism used in QHFC (a preliminary patent application had already been filed as of the time of this writing). In this algorithm, the breeding of lower-level individuals is performed only when the population experiences difficulty in making progress in the aggressive exploration of the top level. This provides a very appealing solution to the balance of exploration and exploitation for sustainable and efficient search: an evolutionary algorithm can run (and exploit) as aggressively as possible, so long as this aggressive search is backed up by the diversity of the supporting lower levels. As the building blocks of higher levels are fully exploited, breeding at lower levels to introduce new building blocks to higher levels is intensified. Experiments showed that this technique enabled QHFC to beat existing genetic algorithms in both sustainability and efficiency on several benchmark problems.

The Continuous HFC algorithm proposed in Section 4.4 has the benefit of ease-of-use and amenability to theoretical analysis. However, incorporating the adaptive potency detection and adaptive breeding of QHFC into CHFC is not straightforward, as there is no explicit concept of levels in CHFC. More investigation is needed to see if that can be achieved.

All HFC-based evolutionary algorithms developed so far only use information of individuals in the current population while neglecting those die-out individuals. However, in problems with expensive evaluations, it is highly desirable to keep all evaluated individuals in the history and organize them into the HFC fitness levels in an appropriate way, e.g., to maximize the diversity in all fitness levels.

8.3.3 Application of HFC to Evolution Strategies

The HFC principle for sustainable evolution has been applied so far to genetic programming and genetic algorithms. Since current evolution strategies have the same fundamental limitation arising from their canonical convergent framework and HFC is a general evolution model, HFC is expected to be applicable also to evolution strategies to enhance their sustainability, robustness and scalability. Experiments are needed to evaluate the performance of the HFC model for evolution strategies. However, because evolution strategies mainly employ adaptive mutation and HFC is hypothesized to work best by assembling building blocks, the question of how HFC should be enhanced to be more effective in evolution strategies is also an interesting topic.

8.3.4 Application of HFC to Other Optimization and Evolutionary Systems

The "survival of the fittest" idea is not only employed in most existent evolutionary algorithms, it also underlies many other traditional mathematical optimization and search algorithms. The illumination of the risk of keeping only the good solutions and searching based on these high-fitness solutions discovered earlier suggests that the sustainability enabled by the HFC principle can also be exported to these traditional mathematical optimization and search algorithms. Potential applications of HFC include simulated annealing, particle swarm optimization, artificial life systems, brute-force search, etc.

8.3.5 Better Strategy for Simultaneous Topology and Parameter Search

This dissertation has proposed the structure fitness sharing (SFS) technique to achieve balanced topology and parameter search simultaneously. However, fitness sharing works by modifying the fitness of individuals and thus changes the fitness landscape. This will cause a problem when we want to combine the SFS technique with the HFC algorithm, since the organization of the hierarchical levels in HFC depends on the absolute fitness of individuals. How to modify SFS to achieve a synergistic effect with HFC is an interesting topic and has bright promise for scalable evolutionary synthesis.

8.3.6 Evolvability in Evolutionary Synthesis of Bond Graphs

In this dissertation, it has been demonstrated that the particular representation of the GP function set can have significant effect on the performance of the synthesis engine. We also showed that compared to hybrid encoding, node encoding achieves higher scalability in bond graph evolution. There are several additional issues here. One is that the experiments in that chapter of the dissertation only searched in the topology space without considering parameters (fixed as unity), which makes the search space much more rugged than the real-world evolutionary synthesis space. Since introducing parameter search into these experiments can greatly smooth the search landscape, it would be interesting to see how this will affect the final performance of these two encoding approaches. Another research question is the influence of GP operators on the distribution of the generated candidates in the initial generation and how this distribution would affect the later evolution process. In addition, in many situations, we do have some high-level modules for synthesizing certain types of dynamic systems. To this point, there has been no study on appropriate ways to incorporate multi-port modules into GP-based synthesis.

8.3.7 Development and Modularity in Evolutionary Synthesis

It is widely believed that the scalability of topologically open-ended synthesis depends on the capability of discovering and exploiting these hierarchical building blocks. HFC provides a good evolution model for this kind of building-block-type evolution. But HFC is only a weak approach that may, but is not guaranteed to, exploit the underlying building blocks inherent in the representation of the building blocks. It would be much more effective if HFC could be combined with some explicit building block identification mechanism. Current approaches in the area of artificial embryology are very promising, though no breathtaking results have yet been reported. Combining HFC with these developmental mechanisms is another promising research topic. One issue here is that to realize the potential of HFC, it is critical to implement the context mechanism, which is invariably neglected by existent evolutionary developmental systems.

BIBLIOGRAPHY

Aickelin, U. & Dowsland, K. (2000). Exploiting problem structure in a genetic algorithm approach to a nurse rostering problem. *Journal of Scheduling*, 3(3), 139-153.

Aiyarak, P., Saket, A. S. & Sinclair, M. C. (1997). Genetic programming approaches for minimum cost topology optimization of optical telecommunication networks. *Proc. 2nd IEE/IEEE Intl. Conf. on Genetic Algorithms in Engineering Systems: Innovations and Applications* (GALESIA'97), Glasgow, September 1997, 415-420

Altenberg, L. (1994). The evolution of evolvability in genetic programming. In *Advances in Genetic Programming*, K.E.Kinnear Jr., Eds. MIT press.

Andre D. & Teller, A. (1999). Evolving Team Darwin United. RoboCup-98: Robot Soccer World Cup II. *Lecture Notes in Computer Science*, M. Asada and H. Kitano, eds., Vol. 164, Springer-Verlag, Berlin, pp.346-352, 1999.

Angeline, P. J. (1994). Genetic programming and emergent intelligence. In *Advances in Genetic Programming*, E., Kinnear, Jr, (editor) MIT Press, Cambridge, MA.

Angeline, P. J. (1998). A Historical Perspective on the Evolution of Executable Structures. *Fundamenta Informaticae*, 35(1-4), 179-195.

Angeline, P. J. & Pollack, J. B. (1994). Coevolving high level representations. In *Proc. of Artificial Life IIII.*

Antonsson, E. K. & Cagan, J. (2001). *Formal Engineering Design Synthesis*, Cambridge University press, Cambridge.

Bäck, T. (1994). Selective pressure in evolutionary algorithms: A characterization of selection mechanisms. In *Proc. of the First IEEE conf. on Evolutionary Computation*, 1, 55-62. Piscataway, NJ: IEEE Service Center

Bäck, T., Fogel, D. B., Michalewicz, Z., & others, (eds.) (1997). *Handbook on Evolutionary Computation*. IOP Publishing Ltd and Oxford University Press.

Bäck, T. & Schwefel, H.-P. (1993). An overview of Evolutionary algorithms for parameter Optimization. *Evolutionary Computation,* 1(1), 1-24.

Baldwin, C. Y., and K. B. Clark. (2000). *Design Rules: The Power of Modularity*. Cambridge, MA: MIT Press.

Banzhaf, W., Banscherus, D. & Dittrich, P. (1998). Hierarchical genetic programming using local modules. In *Proc. Intl. Conference on Complex Systems* 1998, Nashua, NH.

Beyer, H-G. (1995). Toward a theory of evolution strategies: on the benefits of sex- the ($\mu/(\mu,\lambda)$) theory. *Evolutionary Computation*, 3(1), 81-111.

Beasley, D., Bull, D. R. & Martin, R. R. (1993). A Sequential Niche Technique for Multimodal Function Optimisation. *Evolutionary Computation*, 1, 101-125.

Bentley, P. J., Gordon, T., Kim, J. & Kumar, S. (2001). New Trends in Evolutionary Computation. In *Proc. of the Congress on Evolutionary Computation* (CEC-2001), Seoul, Korea, 162-169.

Bentley, P. J. (2002). Evolving Fractal Proteins. A late-breaking paper in *Proc. of the Genetic and Evolutionary Computation Conference* (GECCO 2002), New York.

Bentley, P. J. (2004) Fractal Proteins. *Genetic Programming and Evolvable Machines*, 5, 71-101. Kluwer Academic Publishers, London.

Blickle, T. & Thiele, L. (1996). A Comparison of Selection Schemes used in Evolutionary Algorithms. *Evolutionary Computation*, 4 (4), 361-394.

Boers, E. J. W. & Kuiper, H. (1992). *Biological metaphors and the design of modular artificial neural networks*. Master Thesis, Leiden University.

Bongard, J. C. (2002). Evolving modular genetic regulatory networks. In *Proc. of the IEEE Congress on Evolutionary Computation* (CEC2002), 1872-1877. IEEE Press.

Bossert, W. (1967). Mathematical optimization: Are there abstract limits on natural selection?". In *Mathematical Challenges to the Neo-Darwinian Interpretation of Evolutions*, P.S. Moorhead and M.M. Kaplan (eds), The Wistar Institute Press, Philadephia, PA, pp.35-46.

Brameier, M. & Banzhaf, W. (2001). A Comparison of Linear Genetic Programming and Neural Networks in Medical Data Mining. *IEEE Transactions on Evolutionary Computation*, 5, 17 – 26.

Bremermann,H. J. (1962). Optimization through evolution and recombination. In *Self-organizing Systems*. M.C. Yovits, G.T.Jacobi, and G.D. Goldstine, Eds. Washington, DC: Spartan Books, 93-106.

Bremermann, H. J., Rogson, M. & Salaff, S. (1966). *Global properties of evolution processes. Natural Automata and Useful Simulations*. H.H. Pattee, E.A.Edelsack, L.Fein, and A.B.Callahan (eds), Spartan Books, Washington D.C., 3-41.

Burke, E. S., Gustafson, G. K. (2002). A Survey and Analysis of Diversity Measures in Genetic Programming. In *Proceeding of Genetic and Evolutionary Computation Conference* (GECCO2002), New York.

Burke, E., Gustafson, S. & G. Kendall. (2004). Diversity in Genetic Programming: An Analysis of Measures and Correlation with Fitness. *IEEE Transactions on Evolutionary Computation*, 8(1), 47-62.

Carter, B. & Park, K. (1994). Scalability problems of genetic search. In *Proc. of the 1994 IEEE International Conference on Systems, Man, and Cybernetics*, 1591-1596

Cavicchio, Jr. (1970). *Adaptive search using simulated evolution*. Ph.D thesis, University of Michigan, Ann Arbor, MI. (University Microfilms No. 25-0199).

Cantú-Paz, E. & Goldberg, D. E. (1999). On the scalability of Parallel Genetic Algorithms. *Evolutionary computation*. 7(4), 429-449

Cantú-Paz, E. & Goldberg, D. E. (2003). Are Multiple Runs of Genetic Algorithms Better than One? *Proceeding of Genetic and Evolutionary Computation Conference* (GECCO 2003), 801-812, Chicago, IL.

Cellier, F.E. (1991). *Continuous System Modeling*, Springer-Verlag, New York

Chakraborty, U. K. & Janikow, C. Z. (2003). An analysis of gray versus binary encoding in genetic search. *Information Sciences: an International Journal*, 156 (3-4), 253-269.

Chen, D., Aoki, T., Homma, N., Terasaki, T. & Higuchi, T. (2002). Graph-Based Evolutionary Design of Arithmetic Circuits. *IEEE Transactions on Evolutionary Computation*, 6 (1), 86-100.

Chowdhury, D. & Stauffer, D. (2003). *Computer Simulations of History of Life: Speciation, Emergence of Complex Species from Simpler Organisms, and Extinctions*, 2003-11-4, DOI: q-bio.PE/0311002, arXiv.

Cramer, N. L. (1985). A representation for the Adaptive Generation of Simple Sequential Programs. In *Proc. of an International Conference on Genetic Algorithms and the Applications*, Grefenstette, John J. (ed.), CMU.

Collins, R. J. & Jefferson, D. R. (1991). Selection in massively parallel genetic algorithms. In *Proc. Of the Int'l Conf. On Genetic Algorithms* (ICGA-91), 249-256.

Daida, J., Ward, D., Hilss, A., Long, S. & Hodges, M. (2004). Visualizing the Loss of Diversity in Genetic Programming. In *Proc. of Congress on Evolutionary Computation*, CEC2004, IEEE Press.

Darwin, C. (1859). *The Origin of Species*. John Murray, London.

Darwen, P. J. (1996). *Co-evolutionary Learning by Automatic Modularisation with Speciation.* Ph.D. Thesis. University of New South Wales.

Darwen, P. & Yao, X. (1996). Automatic modularization by spe-ciation. In *Proc. of IEEE International Conference on Evolutionary Computation*, CEC1996, 88–93.

Davidor, Y. (1991). A Naturally Occurring Niche & Species Phenomenon: The Model and First Results. In *Proceedings of the Fourth International Conference on Genetic Algorithms*, 257-263. San Mateo, California, USA: Morgan Kaufmann Publishers.

Dawkins, R. (1987). The Evolution of Evolvability. In *Proc. of Artificial Life VI*. Langton (Ed.) USA.

Dawkins, R. (1986). *The Blind Watchmaker: Why the Evidence of Evolution Reveals a Universe Without Design*. W.W. Norton & Company, New York.

Day, S. J. & Lawrence P. A. (2000). Measuring dimensions: the regulation of size and shape. *Development*, 127, 2977–2987.

Deb, K & Gulati, S. (2001). Design of truss-structures for minimum weight using genetic algorithms. *Journal of Finite Elements in Analysis and Design*, 37, 447-465.

De Garis, H. (1990). Building artificial nervous systems using genetically programmed neural network modules. In *Proc. of the Seventh International Conference on Machine Learning*, 132–139, Porter, B. and Mooney, R., eds. Morgan Kaufmann, Palo Alto, California.

De Jong, K. A. (1975). *An Analysis of the Behavior of a Class of Genetic Adaptive Systems*, PhD thesis. University of Michigan. Dissertation Abstracts International 36(10), 5410B. (University Microfilms No. 76--9381).

De Jong, K. A. (2002). *Evolutionary Computation*. MIT press.

De Jong, E. D. & Oates, T. (2002). A Coevolutionary Approach to Representation Development. In *Proc. of the ICML-2002 Workshop on Development of Representations*.

De Jong, E. D. & Pollack, J. B. (2001). Utilizing Bias to Evolve Recurrent Neural Networks. In *Proc. of Int. Joint Conf. on Neural Networks*, IJCNN 2001.

De Jong, E. D., Watson, R. A. & Pollack, J. B. (2001). Reducing Bloat and Promoting Diversity using Multi-Objective Methods. In *Proc. of the Genetic and Evolutionary Computation Conf.* (GECCO 2001), San Francisco, California.

Dellaert, F. & Beer, R. D. (1994). Toward an Evolvable Model of Development for Autonomous Agent Synthesis. In *Artificial Life IV, Proc. of the Fourth International Workshop on the Synthesis and Simulation of Living Systems*, Brooks, R. and Maes, P. (Eds) MIT Press: Cambridge, MA.

Dellaert, F. (1995). *Toward a Biologically Defensible Model of Development*. Master Thesis. Department of Computer Engineering and Science. Case Western Reserve University.

Dengiz, B., Altiparmak, F. & Smith, A. E. (1997). Local search genetic algorithm for optimal design of reliable networks. *IEEE Transactions on Evolutionary Computation*, 1(3), 179-188.

Dobzhansky, T. (1973).*The American Biology Teacher*, 35, 125-129.

Dorigo M., G. Di Caro & L. M. Gambardella (1999). Ant Algorithms for Discrete Optimization. *Artificial Life*, 5(2), 137-172.

Eggenberger, P. (1996). Cell interactions as a control tool of developmental processes for evolutionary robotics. *FROM ANIMALS TO ANIMATS 4*, Fourth International Conference on Simulation of Adaptive Behavior, Cambridge, MA: The MIT Press/Bradford Books.

Eggenberger, P. (1997). Evolving Morphologies of Simulated 3d Organisms Based on Differential Gene Expression." In: Husbands, P. & Harvey, I. (eds.) *Proc. of the 4th European Conference on Artificial Life* (ECAL97). Cambridge: MIT Press.

Eggenberger, P. (2003). Genome-Physics Interaction as a New Concept to Reduce the Number of Genetic Parameters in Artificial Evolution. In *Proc. of the Congress of Evolutionary Computation*, Canberra 2003.

Eiben, A.E., Hinterding, R., & Michalewicz, Z. (1999) Parameter Control in Evolutionary Algorithms. *IEEE Transaction on Evolutionary Computation*, 3(2), 124 - 141.

Ekárt, A. & Németh, S. Z. (2000). A Metric for Genetic Programs and Fitness Sharing. In R. Poli, W. Banzhaf, W. B. Langdon, J. Miller, P. Nordin, T. Fogarty (eds.), *Genetic Programming, Proc. of EUROGP'2000*, Edinburgh, 15-16 April 2000, LNCS volume 1802, 259-270.

Emmerich. M., Groetzner. M. & Schuetz. M. (2001). Design of Graph-based Evolutionary Algorithms: A case study for Chemical Process Networks. *Evolutionary Computation*, 9(3), MIT Press.

Eshelman, L. J. (1990). *The CHC Adaptive Search Algorithm: How to Have Safe Search When Engaging in Nontraditional Genetic Recombination.* Morgan Kauffman, San Mateo, CA.

Fan, Z., Hu, J., Seo, K., Goodman, E. D., Rosenberg, R. C. & Zhang, B. (2001). Bond Graph Representation and GP for Automated Analog Filter Design. In *Proc. of Genetic and Evolutionary Computation Conference Late-Breaking Papers*, E. Goodman, ed., ISGEC Press, San Francisco, 81-86.

Fan, Z, Seo, K., Hu, J., Rosenberg, R. & Goodman, E. (2003). System-Level Synthesis of MEMS via Genetic Programming and Bond Graphs. In *Proc. of Genetic and Evolutionary Computation Conference*, Chicago, Springer, Lecture Notes in Computer Science, 2058-2071.

Fang, E. W., (1994). Simultaneous Type and Dimensional Synthesis of Mechanisms by Genetic Algorithms. *Mechanism Synthesis and Analysis* – DE, (70).

Ferreira, C. (2001). Gene Expression Programming: a New Adaptive Algorithm for Solving Problems. *Complex Systems*, 13 (2), 87-129.

Fogel, D. (1998). *Evolutionary Computation: The Fossil Record.* (eds). IEEE Press.

Fogel, D. B., Fogel, L.J., Atmar, W., & Fogel, G.B. (1992) Hierarchic methods of evolutionary programming. In *Proc. Of the First Ann. Conf. On Evolutionary Programming.*, D.B.Fogel and W.Atmar, Eds. La Jolla, CA: Evolutionary Programming society, 11-22.

Fogel, L. J. (1962). *Autonomous automata.* Ind. Res., 4,14-19.

Fogel, L.J., Owens,A. J. & Walsh,M. J. (1966). *Artificial Intelligence through Simulated Evolution*, Wiley, New York.

Fonlupt, C., Preux, P., & Robilliard, D. (1994). Preventing Premature Convergence via Cooperating Genetic Algorithms. In *Proc. of the Workshop on Foundations of Genetic Algorithms*, Darrell Whitley, eds. Vail, CO, USA, 1993. Morgan Kaufmann, San Mateo, CA, USA.

Fonseca, C. M. & Fleming, P. J. (1993). Genetic Algoritms for Multiobjective Optimisation: Formulation, Discussion, and Generalization. In *Proc. Fifth International Conference on Genetic Algorithms*, Morgan Kaufmann, San Francisco, 416-423.

Fraser, A.S. (1957). Simulatio of genetic systems by automatic digital computers. I. Introduction. *Australian Journal of Biological. Sci.*, 10, 484-491.

Friedberg, R. M. (1958). A learning machine, Part I. *IBM. Journal of Research and Development*, 2(1), 282-287

Friedberg R. M., Dunham, B., & North, J. H. (1959). A learning machine: Part II. *IBM Journal of Research and Development*, 3, 282-287.

Fukunaga, A. (1997). Restart scheduling for genetic algorithms. In *Proc. of the Seventh International Conference on Genetic Algorithms*, Thomas Bǎck, eds.

Funes, P. & Pollack, J. (1998). Evolutionary Body Building: Adaptive physical designs for robots. *Artificial Life* 4, 337-357.

Wang, G. & Soule, T. (2004). How to Choose Appropriate Function Sets for Genetic Programming. *Proc. of Genetic Programming, 7th European Conference*, EuroGP2004, Coimbra,Portugal, April 5-7, 2004, ,198-207.

Galar, R. (1985). Handicapped Individuals in Evolutionary Processes. *Biological Cybernetics*, 51: 1-9.

Gang W. & Soule, T. (2004). How to Choose Appropriate Function Set for Genetic Programming. *Proc. of Genetic Programming 7th European Conference*, EuroGP 2004.

Gao, Y. (2003). Population Size and Sampling Complexity in Genetic Algorithms. *Proc. of the Bird of a Feather Workshops* (GECCO2003)---Learning, Adaptation, and Approximation in Evolutionary Computation, 178-181, AAAI.

Gilbert, S. F. (2003). *Developmental Biology.* Seventh Edition 2003 Sinauer Associates,Inc., Sunderland, MA.

Giordana, A. & Neri, F. (1996). Search-intensive concept induction. *Evolutionary Computation*, 3(4), 375–416.

Glickman, M. R. & Sycara, K. (2000). Evolvability and Static vs. Dynamic Fitness. In C.C. Maley, ed., *Workshop Proc. of Artificial Life VII*, August, 2000.

Globus, Al, Lawton, J. & Wipke, T. (1999). Automatic molecular design using evolutionary techniques. *Proc. of Sixth Foresight Conference on Molecular Nanotechnology*, Sunnyvale, California, November, 1998 and Nanotechnology, 10(3), 290-299.

Glover, F. (1989). Tabu search—part. I. *ORSA. Journal of Computing*, 1(3), 190-206

Goldberg, D. E. (1989a). *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison Wesley.

Goldberg, D. E. (1989b). Sizing Populations for Serial and Parallel Genetic Algorithms. In P*roc.3rd Int'l Conf. on Genetic Algorithms*, Morgan Kaufman Publishing

Goldberg, D. E., Korb, B., & Deb, K. (1989). Messy genetic algorithms: Motivation, analysis and first results. *Complex Systems*, 3(5), 493-530.

Goldberg, D. E., Deb, K. & Clark, J. H. (1992). Genetic algorithms, noise, and the sizing of populations. *Complex Systems*, 6:333-362.Kaufman.

Goldberg, D. E. (2002). *The Design of Innovation: Lessons from and for Competent Genetic Algorithms*. Boston, MA: Kluwer Academic Publishers.

Goodman, E. D., Seo, K., Fan, Z. Hu, J. & Rosenberg R. C. (2003). Automated Design of Mechatronic Systems: Novel Search Methods and Modular Primitives to Enable Real-World Applications. In *Proc. of 2003 NSF Design, Service and Manufacturing Grantees and Research Conference*, Birmingham, Alabama..

Gordon, T. G. M & Bentley, P.J. (2002). Towards Development in Evolvable Hardware. In *Proc. of the 2002 NASA/DoD Conference on Evolvable Hardware*, Washington D.C., U.S.A., 241-250.

Gordon, T. G. M. (2003). Exploring Models of Development for Evolutionary Circuit Design. In *Proc. of the Congress on Evolutionary Computation*, CEC2003, Canberra, Australia.

Gorges-Schleuter, M. (1989). ASPARAGOS: An asynchronous parallel genetic optimization strategy. In *Proc. Of the Int'l Conf. On Genetic Algorithms* (ICGA-89), 422-428.

Grady, D. (1993). The vision thing: mainly in the brain. *Discovery*, 14, 57-66.

Gruau, F. (1992). Genetic Synthesis of Boolean Neural Networks with a Cell Rewriting Developmental Process. In *Proc. of the International Workshop on Combinations of Genetic Algorithms and Neural Networks* (COGANN-92), 55–74. L. Darrell Whitley and J. David Schaf-fer, Eds. Los Alamitos. IEEE Press.

Gruau, F. & Whitley, D. (1993). Adding learning to the cellular Development of neural networks: Evolution and the Baldwin effect. *Evolutionary Computation*, 1 (3), 213-234.

Gruau, F. 1994. Automatic definition of modular neural networks. *Adaptive Behavior*, 3(2), 151-184.

Haddow, P. C. & Tufte, G. (2001). Briding the Genotype-Phenotype Mapping for Digital FPGAs. *Proc. Of the 3rd NASA/DoD Workshop on Evolvable Hardware*, Pasadena, California.

Hansen, J. V. (2003). Genetic Programming Experiments with Standard and Homologous Crossover Methods. *Genetic Programming and Evolvable Machines*, 4(1), 53-66.

Harary, F. & Palmer, E. (1973). *Graphical Enumeration*. Academic Press, New York.

Harding, S. L. & Miller, J. F. (2004). Evolution in materio: Initial experiments with liquid crystal. In *Proc. of 2004 NASA/DoD Conference on Evolvable Hardware*

Harik G. R. (1994). Finding multimodal solutions using restricted tournament selection. In *Proc. of Sixth International Conference on Genetic Algorithms*.

Harik, G. R. (1997). *Learning Linkage to Efficiently Solve Problems of Bounded Difficulty Using Genetic Algorithms*. PhD thesis. University of Michigan, Ann Arbor.

Harik, G. R., Cantú-Paz, E., Goldberg, D. E., & Miller, B. L. (1997). The Gambler's Ruin Problem, Genetic Algorithms, and the Sizing of Populations. In *Proc. of the 1997 IEEE Conference on Evolutionary Computation*, 7-12, IEEE Press, New York, NY.

Harjani, R., Rutenbar, R. A. & Carley, L. R. (1989). OASYS: A framework for analog circuit synthesis. *IEEE Transactions on Computer Aided Design*, 8, 1247-1266.

Harvey, I., Husbands, P., Cliff, D., Thompson, A. & Jakobi, N., (1996). Evolutionary Robotics: the Sussex Approach. Robotics and Autonomous Systems, 20, 205-224.

Herdy, M. (1992). Reproductive Isolation as Strategy Parameter in Hierarchically Organized Evolution Strategies. In *Proc. of Parallel Problem Solving from Nature*, PPSN 1992, R. Männer and B.Manderick, (eds), 2, Amsterdam, Elsevier.

Hillis, D. W. (1991). Co-evolving parasites improve simulated evolution as an optimization procedure. In Langton, C. G., Taylor, C., Farmer, J. D. and Rasmussen, S., editors, Artificial Life II, *SFI Studies in the Sciences of Complexity*, 10, 313–324. Addison-Wesley, Redwood City, California.

Holland, J.H. (1975). *Adaptation in natural and artificial systems*. First edition.University of Michigan Press. Reprint MIT Press, 1992.

Holland, J. H. & Reitman, J. S. (1978). Cognitive systems based on adaptive algorithms. In Water-man, D. A. and Hayes-Roth, F., editors, *Pattern-Directed Inference Systems*. Academic Press, New York.

Holland. J. H. (2000). Building Blocks, Cohort Genetic Algorithms, and Hyperplane-Defined Functions. *Evolutionary Computation*, 8(4), 373-391.

Hornby, G. S. & Pollack, J. B. (2001). The Advantages of Generative Grammatical Encodings for Physical Design. *In Proc. of IEEE. Congress on Evolutionary Computation*, Seoul, South Korea

Hornby, G. S. & Pollack, J. B. (2002). Creating High-Level Components with a Generative Representation for Body-Brain Evolution. *Artificial Life*, 8(3), 223-246.

Hornby, G. S., Hod, L. & Pollack, J. B. (2003). Generative Representations for the Automated Design of Modular Physical Robots. *IEEE Transactions on Robotics and Automation*, 19:4, 703-719.

Hsu. W. H. & Gustafson, S. M. (2002). Genetic Programming and Multi-Agent Layered Learning by Reinforcements. *Proceeding of the Genetic and Evolutionary Computation Conference*: 764-771. Langdon.W. B. et al. (eds). Morgan Kaufmann.

Hu, J., Seo, K., Li, S., Fan, Z., Rosenberg, R. C., Goodman, E. D. (2002). Structure Fitness Sharing (SFS) for Evolutionary Design by Genetic Programming. In *Proc. of the Genetic and Evolutionary Computation Conference*, GECCO-2002, 780-787, New York.

Hu, J., Goodman, E. D. (2002). Hierarchical Fair Competition Model for Parallel Evolutionary Algorithms. *Proc., Congress on Evolutionary Computation*, CEC 2002, IEEE World Congress on Computational Intelligence, Honolulu, Hawaii.

Hu, J. Seo, K., Li, S., Fan, Z., Rosenberg, R.C., Goodman, E.D. (2002a). Structure Fitness Sharing (SFS) for Evolutionary Design by Genetic Programming. *Proc. of the Genetic and Evolutionary Computation Conference*, GECCO-2002, 780-787, New York.

Hu, J., Goodman, E. D., Seo, K. & Pei, M. (2002b). Adaptive Hierarchical Fair Competition (AHFC) Model for Parallel Evolutionary Algorithms. *Proc. of the Genetic and Evolutionary Computation Conference*, GECCO-2002, New York, 772-779.

Hu, J., Goodman, E. D & Rosenberg, R. (2004). Robust and Efficient Genetic Algorithms with Hierarchical Niching and a Sustainable Evolutionary Computation Model. *Proc. of the Genetic and Evolutionary Computation Conference*, *Lecture Notes in Computer Science,* Springer, GECCO-2004, Seattle, Part I, 1220-1232.

Husbands, P. & Mill, F. (1991). Simulated co-evolution as the mechanism for emergent planning and scheduling. *Proc. of the Fourth International Conference on Genetic Algorithms*, 264–270, Belew, R. K. and Booker, L. B., eds. Morgan Kaufmann, San Mateo, California.

Hutter, M. (2002). Fitness Uniform Selection to Preserve Genetic Diversity. In *Proceedings of the 2002 Congress on Evolutionary Computation*, 783-788 (CEC-2002), Hawaii.

Huynen, M. (1996). Exploring phenotype space through neutral evolution. *Journal of Molecular Evolution*, 43, 165-169.

Huynen, M, Stadler, P & Fontana,W. (1996). Smoothness within ruggedness: the role of neutrality in adaptation. *Proc. Of the National Academy of Science*, 93, 397-401.

Igel, C. & Stagge, P. (2002). Effects of Phenotypic Redundancy in Structure Optimization. *IEEE Trans. On Evolutionary Computation*, 6 (1), 74-85.

Jakobi, N. (1996). Harnessing Morphogenesis. University of Sussex, *Cognitive Science Research Report* #429, Brighton, UK.

Jin, Y. (2002). Fitness approximation in evolutionary computation - A survey. *Proc. of Genetic and Evolutionary Computation Conference*, 1105-1112, New York.

Jin, Y, Olhofer, M, & SendHoff, B. (2002). A Framework for Evolutionary Optimization with Approximate Fitness Functions. *IEEE Transactions on Evolutionary Computation*, 6(5), 481-494.

Johnson H. E., Gilbert R. J., Winson K., Goodacre R., Smith A. R., Rowland J. J., Hall M. A., & Kell D. B. (2000). Explanatory analysis of the metabolome using genetic programming of simple, interpretable rules. *Genetic Programming and Evolvable Machines*, 1(3), 243-258.

Kargupta, H. (1996). The gene expression messy genetic algorithm. *Proc. Of the IEEE Int. Conf. on Evolutionary Computation*, 814-819. IEEE press.

Kargupta, H. & Park, B. H. (2001). Gene expression and fast construction of distributed evolutionary representation. *Evolutionary Computation*, 9(1), 43-69.

Karnopp D. C., Margolis, D. L. & Rosenberg, R. C. (2000). *System Dynamics: Modeling and Simulation of Mechatronic Systems*. Third Edition. New York: John Wiley & Sons, Inc.

Kauffman, S. A. (1993). *The Origins of Order: Self-organization and Selection in Evolution*. Oxford University Press. New York.

Keane, Martin A., Koza, John R., & Streeter, Matthew J. (2002). Automatic synthesis using genetic programming of an improved general-purpose controller for industrially representative plants. In Stoica, Adrian, Lohn, Jason, Katz, Rich, Keymeulen, Didier and Zebulum, Ricardo (editors). *Proc. of 2002 NASA/DoD Conference on Evolvable Hardware*, 113-122, Los Alamitos, CA: IEEE Computer Society.

Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, 13, 220, 4598, 671-680.

Kitano, H. (1990). Designing Neural Networks Using a Genetic Algorithm with a Graph Generation System. *Complex Systems* 4, 461–476.

Kitano, H. (1997). Challenges of evolvable systems: Analysis and future directions. *Evolvable Systems: From biology to Hardware*, 1259, Lecture Notes in Computer Science, 125-135.

Koza, J. R. (1994). *Genetic Programming II: Automatic Discovery of Reusable Programs*. MIT Press, Cambridge Massachusetts, May 1994.

Koza, J. R., Forrest H. Bennett III, Andre, D., Keane, M. A. & Dunlap, F. (1997). Automated synthesis of analog electrical circuits by means of genetic programming. *IEEE Transactions on Evolutionary Computation*, 1 (2), 109-128

Koza, J., Bennett,F. H, Andre, D. & Keane, M.A. (1999). *Genetic Programming III: Darwinian Invention and Problem Solving*. Morgan Kauffmann, San Francisco, CA.

Koza, J. R., Keane, M. A., Yu, J., Bennett, F. H. III, & Mydlowec, W. (2000). Automatic creation of human-competitive programs and controllers by means of genetic programming. *Genetic Programming and Evolvable Machines*, 1 (1-2), 121-164.

Koza, J. R., Keane, M. A., Streeter, M. J. (2003a). What's AI Done for Me Lately? Genetic Programming's Human-Competitive Results. *IEEE Intelligent Systems*, 18(03), 25-31.

Koza, J. R., Keane, M. A., Streeter, M. J., Mydlowec, W., Yu, J. & Lanza, G. (2003b). *Genetic Programming IV: Routine Human-Competitive Machine Intelligence*. Kluwer Academic Publishers.

Koza, J. R., Streeter, M. J., & Keane, M. A. (2003c). Automated synthesis by means of genetic programming of complex structures incorporating reuse, parameterized reuse, hierarchies, and development. In Riolo, Rich and Worzel, William. 2003. *Genetic Programming: Theory and Practice*. Boston, MA :Kluwer Academic Publishers.

Koza, J. R., Jones, L.W., & Keane, M.A. (2004). Toward Industrial-Strength Automated Design of Analog Circuits by Means of Genetic Programming. In *Genetic Programming Theory and Practice*. Rick Riolo and Bill Worzel (eds.). Kluwer Publishers, Boston, MA.

Kumar, S. (2004). *Investigating Computational Models of Development for the Construction of Shape and Form*. Ph.D. Thesis, Department of Computer Science, University College London.

Kumar, S. & Bentley, P. J. (Editors) (2003). *On Growth, Form and Computers*. Academic Press, London UK.

Kunjur, A. & Krishnamurty, S. (1995). Genetic Algorithms in Mechanism Synthesis. *Proc. of Fourth Applied Mechanisms and Robotics Conference*, Cincinnati, Ohio, AMR 95-068, 01-07.

Lawrence, P. (2001). Morphogens: how big is the big picture, *Nature Cell Biology*, 3.

Li, J. P., Balazs, M., Parks. G. T & Clarkson. P. J. (2002). A Species Conserving Genetic Algorithm for Multimodal Function Optimization. *Evolutionary Computation*, 10(3), 207-234.

Lin, S-C., Goodman, E. D., & Punch, W. (1994). Coarse-grain parallel genetic algorithms: Categorization and new approach. *Proc. of IEEE Conf. on Parallel and Distribution. Processing*, 11.

Lipson, H., Antonsson, E. K. & Koza, J. R. (Eds). (2003). *Proceeding of AAI Spring Symposium Series: Computational Synthesis: From basic building blocks to high level functionality*. Palo Alto, CA, USA, AAAI Press.

Lis, J. & Lis, M. (1996). Self-adapting parallel genetic algorithm with the dynamic mutation probability, crossover rate and population size. In J.Arabas, eds. *Proc. Of the 1st Polish National Conf. on Evolutionary Computation*, 324-329.

Lodish, F. H. et. al. (2003). *Molecular Cell Biology*. W. H. Freeman & Company. 5th Edition.

Lohn, J. D. & Colombano, S. P. (1998). Automated analog circuit synthesis using a linear representation. *Proc. of Second International Conference on Evolvable Systems: From Biology to Hardware*, Springer-Verlag.

Lohn, J. D, Colombana, S.P. (1999). A circuit Representation Technique for Automated Circuit Design. *IEEE Trans. on Evolutionary Computation*, 3(3), 205-219.

Lones, M. A. & Tyrrell, A. M. (2002). Biomimetic Representation with Genetic Programming Enzyme. *Genetic Programming and Evolvable Machines*, 3(2), 193-217.

Lones, M. A. (2004). *Enzyme Genetic Programming: Modeling Biological Evolvability in Genetic Programming*. PhD Thesis, Department of Electronics, University of York.

Lones, M. A. & Tyrrell, A. M. (2004). Modelling Biological Evolvability: Implicit Context and Variation Filtering in Enzyme Genetic Programming. *BioSystems*.

Luke, S., & Spector, L. (1996). Evolving Graphs and Networks with Edge Encoding: Preliminary Report. In Koza, John R. (editor), *Late-Breaking Papers at the Genetic Programming 1996 Conference*. Palo Alto, CA.

Luke, S. (2000). *Issues in Scaling Genetic Programming: Breeding Strategies, Tree Generation, and Code Bloat*. Ph.D. Dissertation, Department of Computer Science, University of Maryland, College Park, Maryland.

Luke, S. (2001). When short runs beat long runs. In *Proc. of the Genetic and Evolutionary Computation Conference*, GECCO-2001, Lee Spector et al, eds. Morgan Kaufmann, 74-80.

Mahfoud, S. W. (1992). Crowding and preselection revisited. *Proc. of Parallel Problem Solving from Nature*, PPSN 1992, 27-36.

Mahfoud. S. W. (1995). *Niching Methods for Genetic Algorithms*. PhD thesis, University of Illinois at Urbana-Champaign. IlliGAL Report No. 95001.

Mahfoud, S. W. (1997). Boltzmann selection. In *Handbook of Evolutionary Computation*. T. Bäck, D.Fogel, and Z. Michalewicz, eds. Institute of Physics Publishing Ltd, Bristol and Oxford University Press, New York. C2.5, 1-4

Mason, S. J. (1953). Feedback theory: Some properties of signal flow graphs. In *Proc. Institute of Radio Engineers*, 41, 1144-1156.

Mason, S. J. & Zimmermann, H. J. (1960). *Electronic Circuit, Signals, and Systems*. John Wiley & Sons, New York.

Mathias, K. E. & Whitley, L. D. (1994). Changing Representations During Search: A Comparative Study of Delta Coding. *Evolutionary Computation*, 2(3), 249-278.

McCarthy, J. (1987). Generality in Artificial Intelligence. *Communications of the ACM,* 30(12), 1030-1035.

McKay, R. I. (2000). Fitness Sharing in Genetic Programming. *Proc. of Genetic and Evolutionary Computation Conference*, GECCO2000, Las Vegas, NV, Morgan Kaufmann, San Francisco.

McPhee, N. F. & Hopper, N. J. (1999). Analysis of genetic diversity through population history. In W. Banzhaf et al., editors, *Proc. of the Genetic and Evolutionary Computation Conference*, 1112–1120, Florida, USA. Morgan Kaufmann.

Mengshoel, O. J. & Goldberg, D. E. (1999). Probablistic crowding:Deterministic crowding with probabilistic replacement. In *Proc. Of the Genetic and Evolutionary Computation Conference* (GECCO-99), 409-416.

Miller, J. F. & P. Thomson. (2003). A Developmental Method for Growing Graphs and Circuits . In Proc. Of Fifth International Conference on Evolvable Systems: From Biology to Hardware. *Proc. published as Lecture Notes in Computer Science*, 2606, 93-104, Tyrrell, Andy M., Haddow P., Torrensen J.(Eds.).

Miller. G.F. & Todd. P.M. (1989). Designing neural networks using genetic algorithms. *Proc. Of the Third Int. Conf. On Genetic Algorithms and their Applications*. J. D. Schaffer, Ed. San Mateo, CA: Morgan Kauffman.

Miller, J. F., Job, D. & Vassilev, V. K. (2000). Principles in the Evolutionary Design of Digital Circuits -- Part I. *Genetic Programming and Evolvable Machines*, 1(1), 8-35.

Miller, J. F. & Thomson, P. (2003). A Developmental Method for Growing Graphs and Circuits. *Proc. of Fifth International Conference on Evolvable Systems: From Biology to Hardware*, Trondheim, published as Lecture Notes in Computer Science, 2606, 93-104, Tyrrell, Andy M., Haddow P., Torrensen J.(Eds.).

Moriarty, D. E. & Miikkulainen, R. (1997). Forming neural networks through efficient and adaptive coevolution. *Evolutionary Computation*, 5(4), 373-399.

Mühlenbein, H. (1991). Evolution in time and space- the parallel genetic algorithm. *Foundations of Genetic Algorithms*, 316-337.

Murakawa, M. (1996). Hardware evolution at function level. *Proc. of Parallel Problem Solving from Nature IV (PPSNIV)*. Springer Verlag, LNCS 1141.

Nachbar, R. B. (1998). Molecular evolution: a hierarchical representation for chemical topology and its automated manipulation. *Proc. of the Third Annual Genetic Programming Conference*, 246-253.

Nachbar, N. B. (2000). Molecular Evolution: Automated Manipulation of Hierarchical Chemical Topology and Its Application to Average Molecular Structures. *Genetic Programming and Evolvable Machines*, 1(1), 57-94.

Natsui, M., Aoki, T. & Higuchi, T. (2001). Evolutionary Graph Generation with Terminal-Color Constraint for Heterogeneous Circuit Synthesis. *Electronics Letters*, 37(13), 808-810.

Needham, J. (1975). *Science and Civilization in China*, 4, part 2, 54. Cambridge: Cambridge University Press.

O'Neill M., Ryan C. (2001). Grammatical Evolution. *IEEE Transactions on Evolutionary Computation*, 5(4).

Nordin P. (1994). A Compiling Genetic Programming System that Directly Manipulates the Machine Code. In: Kinnear K.E. (ed.): *Advances in Genetic Programming*, Chapter 14. MIT Press, Cambridge, MA, 311–334

Nordin, P. (1997). *Evolutionary Program Induction of Binary Machine Code and Its Application*, PhD thesis, University of Dortmund, Germany.

Nordin, P., Banzhaf, W., & Francone, F. (1999). Efficient Evolution of Machine Code for CISC Architectures using Blocks and Homologous Crossover. *Advances in Genetic Programming III*: 275-299, MIT Press, Cambridge, MA.

Ogata, K. (1997). *Modern Control Engineering*, 3rd ed. Prentice-Hall.

Oliker, S., Furst, M. & Maimon, O. (1992). A distributed genetic algorithm for neural network design and training. *Complex Systems*, 6, 459-477.

Oltean M., Grosan C. (2004a). Evolving Digital Circuits using Multi Expression Programming. *Proc. NASA/DoD Conference on Evolvable Hardware*, Seattle, R. Zebulum et. al. (eds), 87-90, IEEE Press, NJ.

Oltean, M, Grosan, C. (2004b). A Comparison of Several Linear Genetic Programming Techniques. *Complex-Systems*, 14 (4), 285-313.

Oppacher, F. & Wineberg, M. (1999). The Shifting Balance Genetic Algorithm: Improving the GA in a Dynamic Environment. In: *Proc. of the Genetic and Evolutionary Computation Conference*, Banzhaf, W., Daida, J., Eiben, A.E., Garzon, M.H., Honavar, V., Jakiela, M., and Smith, R.E. (eds.), 1, 504–510.

Patel, N. H. & Lall, S. (2002). Developmental biology: Precision patterning. *Nature* 415, 748 – 749.

Paynter, H. M. (1961). *Analysis and Design of Engineering Systems*, MIT Press, Cambridge, Mass.

Paynter, H. M. (1970) System Graphing Concepts. *Instruments Control Systems*, 43(7),77-78.

Paynter, H. M. (1992). An epistemic prehistory of bond graphs. In P. C. Breedveld and G. Dauphin-Tanguy (ed.), *Bond Graphs for Engineers*, 3-17. Amsterdam, The Netherlands: Elsevier Science Publishers.

Pelikan, M. (2002). *Bayesian Optimization Algorithm: From Single Level to Hierarchy*. Ph.D. Dissertation, Dept. of Computer Science at the University of Illinois at Urbana-Champaign

Pincus, M. (1970). An evolutionary Strategy. *Journal of Theoretical Biology*, 28,483-488.

Poli, R. (1997). Evolution of Graph-Like Programs with Parallel Distributed Genetic Programming. *Proc. of Seventh International Conference on Genetic Algorithms*, E Goodman, (Eds). Michigan State University,346-353, East Lansing, USA. Morgan Kaufmann

Polya, G. (1962). *Mathematical Discovery*, 1, Wiley, New York.

Potter, M.A. & De Jong, K.A. (2000). Cooperative Coevolution: An Architecture for Evolving Coadapted Subcomponents. Evolutionary Computation, 8(1): 1-29. MIT press.

Pujol, J. & Poli, R. (1998). Evolving the topology and the weights of neural networks using a dual representation. Special Issue on Evolutionary Learning. *Applied Intelligence*, 8(1), 73-84.

Rechenberg, I. (1973). *Evolutionsstrategie: Optimierung Technischer Systeme nach Prinzipien der Biologischen Evolution*, Frommann-Holzboog, Stuttgart.

Reed, J., Toombs, R. & Barricelli, N.A. (1967). Simulation of biological eovolution and machine learning. *Journal of Theoretical Biology*, 17, 319-342.

Reisinger, J. (2004). An Overview of Modularity in Artificial Evolutionary Systems. *Proc. of Parallel Problem Solving from Nature*, PPSN 2004.

Reisinger, J., Stanley, K. O. & Miikkulainen, R. (2004). Evolving Reusable Neural Modules. *Proc. of the Genetic and Evolutionary Computation Conference*, (GECCO-2004), New York. Springer-Verlag.

Rodriguez-Vazquez, K., Fonseca, C. M. & Fleming, P. J. (1997). Multiobjective Genetic Programming: A Nonlinear System Identification Application. *Proc. Genetic Programming '97 Conference*, 207-212.

Rosca, J. P. & Ballard, D. H. (1994). Hierarchical Self-Organization in Genetic Programming. *Proc. of the Eleventh International Conference on Machine Learning* (ICML-94): 251-258. Morgan Kaufmann, Los Altos, CA.

Rosca J. P.& Ballard, D.H. (1995) Causality in Genetic Programming. In *Proc. of the Sixth International Conference on Genetic Algorithms*, Morgan Kaufmann Publ., San Francisco CA.

Rosca, J. P. & D. Ballard, D. (1996). Discovery of subroutines in genetic programming. In P. Angeline and K. Kinnear, editors, *Advances in genetic programming*:, 2, 177-202, MIT Press.

Rosenberg, R.C., Goodman, E. D. & Seo, K. (2001). Some Key Issues in Using Bond Graphs and Genetic Programming for Mechatronic System Design. *Proc. ASME International Mechanical Engineering Congress and Exposition*, New York.

Rosin, C. D. & Belew, R. K. (1995). Methods for competitive co-evolution: Finding opponents worth beating. *Proc. of the Sixth International Conference on Genetic Algorithms*, 373–380, Eshelman, L., eds. Morgan Kaufmann, San Francisco, California.

Rothlauf F. & Goldberg, D.E. (2002). *Representations for Genetic & Evolutionary Algorithms*. Physica-verlag press.

Rothlauf, F., Goldberg, D. & Heinzl, A. (2002). Network Random Keys - A Tree Network Representation Scheme for Genetic and Evolutionary Algorithms. *Evolutionary Computation*, 10(7), 75-97.

Rothlauf, F. (2003). Representations for Genetic and Evolutionary Algorithms. *Studies in Fuzziness and Soft Computing*, 104. Heidelberg: Springer, 1st edition 2002. 2nd printing.

Rothlauf, F. & Goldberg, D. E. (2003). Redundant Representations in Evolutionary Computation. *Evolutionary Computation*, 11(4), 381-415.

Ryan, C. (1994). Pygmies and Civil Servants, in K. E. Kinnear, Jr., ed., *Advances in Genetic Programming*, 243-263. MIT Press.

Ryan, C. (1995). Racial Harmony and Function Optimization in Genetic Algorithms - the Races Genetic Algorithm. *Proc. Of Evolutionary Programming* 1995, 296-307. MIT press.

Ryan. C. (1996). *Reducing Premature Convergence in Evolutionary Algorithms*. Ph.D. Thesis. National University of Ireland.

Ryan C., Collins J. J., O'Neill M. (1998). Grammatical Evolution: Evolving Programs for an Arbitrary Language. *Lecture Notes in Computer Science, 1391*. First European Workshop on Genetic Programming.Schoenauer, M., Deb, K., Rudolph, G., Yao, X., Lutton, E., Merelo, J.J., and Schwefel, H.P. (eds.), 1, 817–826.

Schwefel,H.-P.(1975). *Evolutionsstrategie und numerische Optimimierung*. Dissertation, Technische Universitat Berlin, Germany.

Schwefel, H.-P. (1981). *Numerical Optimization of Computer Models*. Chichester, UK: John Wiley.

Seo, K., Hu, J., Fan, Z., Goodman, E. D. & Rosenberg, R. C. (2002). Automated Design Approaches for Multi-Domain Dynamic Systems Using Bond Graphs and Genetic Programming. *The International Journal of Computers, Systems and Signals*, 3(1), 55-70.

Seo, K., Fan, Z. Hu, J. Goodman, E. & Rosenberg, R. (2003a). Dense and Switched Modular Primitives for Bond Graph Model Design. *Proc. Genetic and Evolutionary Computation Conference*, Chicago, Springer, Lecture Notes in Computer Science, 1764-1775.

Seo, K., Fan, Z., Hu, J. Goodman, E. D. & Rosenberg, R. C. (2003b). Toward an Automated Design Method for Multi-Domain Dynamic Systems Using Bond Graphs and Genetic Programming. *Mechatronics*, 13 (8-9), 851-885.

Shackleton, M., Shipman, R. & Ebner, M. (2000). An investigation of redundant genotype-phenotype mappings and their role in evolutionary search. In *Proc. of the 2000 Congress on Evolutionary Computation*, La Jolla, CA, 493-500. IEEE Press.

Shimodaira, H. (1999). A Diversity Control Oriented Genetic Algorithm (DCGA): Development and Experimental Results. In: *Proc. of the Genetic and Evolutionary Computation Conference*, Banzhaf, W., Daida, J., Eiben, A.E., Garzon, M.H.,Honavar, V., Jakiela, M., and Smith, R.E. (eds.), 1, 603–611

Simon, H. (1973). The organization of complex systems. In Pattee, H. H. (eds), *The Challenge of Complex Systems*, George Braziller Publisher, New York.

Smith, R. E., Forrest, S. & Perelson, A. S. (1992). Searching for diverse cooperative populations with genetic algorithms. *Evolutionary computation*, 1(2), 127-149.

Spears, W. M. (1994). Simple Subpopulation Schemes," Proc. Evolutionary Programming Conf., 296-307.

Spector, L., Barnum, H. & Bernstein, H. J. (1999). Quantum Computing Applications of Genetic Programming. *Advances in Genetic Programming 3*, L. Spector et al., eds., MIT Press, Cambridge, Mass., 135-160.

Stanley, K.O & Miikkulainen, R (2002). Evolving Neural Networks through augmenting topologies. *Evolutionary Computation*, 10(2), 99-127.

Stanley, K.O & Risto Miikkulainen (2003). A Taxonomy for Artificial Embryogeny, *Artificial Life* 9(2), 93-130.

Tan, K.C., T.H. Lee, & E.F. Khor. (2001). Evolutionary Algorithms with Dynamic Population Size and Local Exploration for Multiobjective Optimization. *IEEE Transactions on Evolutionary Computation*, 5(6), 565--588.

Tay E., Flowers, W. & Barrus, J. (1998). Automated Generation and Analysis of Dynamic System Designs. *Research in Engineering Design*, 10 (1), 15-29.

Teller, A. & Veloso, M. (1995). PADO: Learning tree-structured algorithms for orchestration into an object recognition system. *Technical Report CMU-CS-95-101*, Department of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA.

Thierens, D. (2000). Scalability Problems of Simple Genetic Algorithms. *Evolutionary Computation*, 7(4), 331-352.

Thierens, D. (2002). Adaptive mutation rate control schemes in genetic algorithms. In *Proc. of the 2002 IEEE World Congress on Computational Intelligence*: Congress on Evolutionary Computation, 980-985. IEEE Press.

Thomsen, R., Rickers, P., & Krink, T. (2000). A Religion-Based Spatial Model For Evolutionary Algorithms. *Proc. Parallel Problem Solving from Nature* – PPSN VI,

Thompson, A., Layzell, P. J., Zebulum, R. S. (1999). Explorations in design space: unconventional electronics design. *IEEE Transactions on Evolutionary Computation.* 3(3), 167-196.

Tiwari,A., Roy, R., Jared, G. & Munaux, O. (2002). Evolutionary-based techniques for real-life optimisation: development and testing. *Applied Soft Computing*, 1(4), 301-329.

Torresen, J. (1998). A divide-and-conquer approach to evolvable hardware. In M. Sipper et al., eds, Evolvable Systems: From Biology to Hardware. *Proc. Second Int. Conf., Int. Conf. on Evolution Strategies*. ICES98, 57–65. Springer-Verlag. Lecture Notes in Computer Science, vol. 1478.

Torresen, J. (2000). Scalable evolvable hardware applied to road image recognition. *Proc. of the 2nd NASA/DoD Workshop on Evolvable Hardware*, Silicon Valley, USA.

Torresen, J. (2002). A Scalable Approach to Evolvable Hardware. *Genetic Programming and Evolvable Machines*, 3 (3), 259-282.

Trent, H. M. (1955). Isomorphisms between oriented linear graphs and lumped physical systems. *The Journal of the Acoustical Society of America*, 27, 500-527.

Tsutsui, S., Fujimoto, Y., & Ghosh, A. (1997). Forking Genetic Algorithms: GAs with Search Space Division Schemes. *Evolutionary Computation*, 5, 61–80

Turing, A.M. (1948). Intelligent machinery. Reprinted in in B. Meltzer & D. Michie, eds, (1970). *Machine Intelligence 5*. American Elsevier Publishing, New York, 3-23.

Tuson, A. & Ross, P. (1996). Cost based operator rate adaptation: An investigation. In *Proc. Of the 4th Conf. on Parallel Problem Solving from Nature,* H.-M. Voigt, W. Ebeling, I. Rechenberg, and H.-P.Schwefel, eds., 461-469

Ursem, R.K. (1999). Multinational Evolutionary Algorithms. *Proc. of the Congress of Evolutionary Computation* (CEC-99), Angeline, P.J., Michalewicz, Z., Schoenauer, M., Yao, X., and Zalzala, A. (eds.), 3, 1633–1640.

Ursem, R. K. (2002). Diversity-Guided Evolutionary Algorithms. *Proceedings of Parallel Problem Solving from Nature VII* (PPSN-2002), 462-471.

Vassilev, V. K., Miller, J. F. (2000). Scalability Problems of Digital Circuit Evolution. *Proc. of the 2nd NASA/DOD Workshop on Evolvable Hardware*, Lohn, J., Stoica, A., Keymeulen, D. and Colombano, S. (eds.), 55-64. Los Alamitos, CA: IEEE Computer Society.

Venkatasubramanian, V., Chan, K. & Caruthers, J. M. (1994). Computer Aided Molecular Design Using Genetic Algorithms. *Computers and Chemical Engineering*, 18 (9), 833-844.

Venkatasubramanian, V., Chan, K., J. M. Caruthers. (1995). Evolutionary Large Scale Molecular Design Using Genetic Algorithms. *Journal of Chem. Info. and Comp. Sci.*, 35, 188-195.

Vitetta, A. (1997). Genetic algorithms for transportation network design. *Advances in Intelligent Systems* xv, 546. IOS Press, Amsterdam, Netherlands.

Wang, G., Dexter, T, Punch, W., & Goodman, E. (1996). Optimization of a GA and within a GA for a 2-Dimensional Layout Problem. In *Proc., First Int'l Conf. on Evolutionary Computation and its Applications*, Presidium, Russian Academy of Sciences, June, 18-29.

Watson, R. A. & Pollack, J. B. (2000). Symbiotic composition as an alternative to sexual recombination in genetic algorithms. In S. et al. (eds). *Proc. of Parallel Problem Solving from Nature*, PPSN VI, 425-434. Springer.

Weaver, R. (2002). *Molecular Biology*, Chapter 22. McGraw Hill Press.

Weininger, D. (1995). Method and apparatus for designing molecules with desired properties by evolving successive populations. *U.S. patent US5434796*, Daylight Chemical Information Systems, Inc.

Wheeler, D.A. http://www.dwheeler.com/sloc.

Whitley,D.(1989). The GENITOR algorithm and selective pressure:Why rank-based allocation of reproductive trials is best. *Proc. 3rd Int. Conf. Genetic Algorithms and Their Applications*, J. D. Schaffer, Ed. San Mateo, CA: Morgan Kaufmann, 116-121.

Whitley, D., Rana, S. & Heckendorn, R. B. (1998). Representation Issues in Neighborhood Search and Evolutionary Algorithms. *Genetic Algorithms and Evolution Strategies in Engineering and Computer Scienc*e. D. Quagliarella, J. Periaux, C. Poloni and G. Winter, eds., 39-57, Wiley.

Wilson, S. W. (1995). Classifier Fitness Based on Accuracy. *Evolutionary Computation*, 3 (2), MIT Press.

Wolpert, D.H. & Macready, W.G. (1997). No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 4, 67-82.

Wolpert L. (1994). Positional information and pattern formation in development. *Developmental Genetics*, 15, 485–90.

Wolpert, D. & Tumer, K. (2001). Optimal Payoff Functions for Members of Collectives. *Advances in Complex Systems*.

Xie, T. & Chen, H. (2001). Problem Decomposition-Based Scalable Macro-Evolutionary Algorithms. *Proc. of the 2001 Congress on Evolutionary Computation* CEC2001, 223-231

Yao X. (1993). A Review of Evolutionary Artificial Neural Networks. *International Journal of Intelligent Systems*, 8, 539-567.

Yao, X. & Higuchi, T. (1999). Promises and Challenges of Evolvable Hardware. *IEEE Transactions on Systems, Man, and Cybernetics*, Part C, 28(4).

Youcef-Toumi K. (1996). Modeling, Design, and Control Integration: A necessary Step in Mechatronics, *IEEE/ASME Trans. Mechatronics*, 1(1), 29-38.

Yu, T. & Miller, J. (2002). Finding needles in haystacks is not hard with neutrality. In *Proc. of the 5th European Conference on Genetic Programming* (EuroGP), 13-25.