

THE PROTEOMICS APPROACH TO EVOLUTIONARY COMPUTATION:
AN ANALYSIS OF PROTEOME-BASED LOCATION INDEPENDENT
REPRESENTATIONS BASED ON THE PROPORTIONAL GENETIC
ALGORITHM

by

IVAN IBARGÜEN GARIBAY

B.S. Ricardo Palma University, 1994

Professional Degree of Electronics Engineer, 1995

M.S. University of Central Florida, 2000

A dissertation submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy
in the School of Computer Science
in the College of Engineering and Computer Science
at the University of Central Florida
Orlando, Florida

Fall Term
2004

Major Professor:
Annie S. Wu

© 2004 by Ivan Ibargüen Garibay

This version of this text has been reformatted
to reduce the total number of pages required to print

ABSTRACT

As the complexity of our society and computational resources increases, so does the complexity of the problems that we approach using evolutionary search techniques. There are recent approaches to deal with the problem of scaling evolutionary methods to cope with highly complex difficult problems. Many of these approaches are biologically inspired and share an underlying principle: a problem representation based on basic representational building blocks that interact and self-organize into complex functions or designs. The observation from the central dogma of molecular biology that proteins are the basic building blocks of life and the recent advances in proteomics on analysis of structure, function and interaction of entire protein complements, lead us to propose a unifying framework of thought for these approaches: the proteomics approach. This thesis propose to investigate whether the self-organization of protein analogous structures at the representation level can increase the degree of complexity and “novelty” of solutions obtainable using evolutionary search techniques. In order to do so, we identify two fundamental aspects of this transition: (1) proteins interact in a three dimensional medium analogous to a multiset; and (2) proteins are functional structures. The first aspect is foundational for understanding of the second.

This thesis analyzes the first aspect. It investigates the effects of using a genome to proteome mapping on evolutionary computation. This analysis is based on a genetic algorithm (GA) with a string to multiset mapping that we call the proportional genetic algorithm (PGA), and it focuses on the feasibility and effectiveness of this mapping. This mapping leads to a fundamental departure from typical EC methods: using a multiset of proteins as an intermediate mapping results in a *completely location independent* problem representation where the location of the genes in a genome has no effect on the fitness of the solutions. Completely location independent representations, by definition, do not suffer from traditional EC hurdles associated with the location of the genes or positional effect in a genome. Such representations have the ability to self-organize into a genomic structure that appears to favor positive correlations between form and quality of represented solutions. Completely location independent representations also introduce new problems of their own such as the need for large alphabets of symbols and the theoretical need for larger representation spaces than traditional approaches. Overall, these representations perform as well or better than traditional representations and they appear to be particularly good for the class of problems involving proportions or multisets.

This thesis concludes that the use of protein analogous structures as an intermediate representation in evolutionary computation is not only feasible but in some cases advantageous. In addition, it lays the groundwork for further research on proteins as functional

self-organizing structures capable of building increasingly complex functionality, and as basic units of problem representation for evolutionary computation.

To my parents, Mama Chabuca and Papa Jaime

ACKNOWLEDGMENTS

I would like to thank my committee members: Dr. Gonzalez, Dr. Guha, Dr. Lang, Dr. De Jong, and Dr. Wu. In particular, I would like to thank my advisor, Annie S. Wu who gave me the freedom to explore, constant encouragement and sound advice when most needed. I would also like single out Ken De Jong for invaluable feedback and critical comments. Additionally, I would like to thank two faculty members from Ricardo Palma University who encouraged me to start graduate school in a far away country called America: my father, who planted the seed and Dr. Victor Latorre who made it a reality. This work would have not been possible without the support of Tom O'Neal and Beverly Laakso at the UCF Office or Research who allowed me to continue my educational quest in one of the best environments for working and conducting research. Last but not least, I would like to offer a big thanks to my wife, Ozlem, for her constant support, encouragement and contributions to this work on a personal and professional level. Finally, I am forever grateful to my mom and dad who inspired me through every step.

TABLE OF CONTENTS

| | |
|--|------------|
| LIST OF TABLES | xi |
| LIST OF FIGURES | xiv |
| CHAPTER 1 INTRODUCTION | 1 |
| 1.1 Evolutionary Computation | 2 |
| 1.2 Complexity Problem | 3 |
| 1.3 Proteomic Approach | 4 |
| 1.4 Studying Proteome-based Location Independent Representations | 5 |
| 1.4.1 Strategy and Methodology | 5 |
| 1.4.2 Contributions | 5 |
| 1.5 Overview | 6 |
| CHAPTER 2 BACKGROUND | 8 |
| 2.1 Evolutionary Computation | 8 |
| 2.1.1 Genetic Algorithms | 9 |
| 2.2 Related Work | 10 |
| 2.2.1 Location Independent Representations | 10 |
| 2.2.2 Genotype to Phenotype Transformations | 14 |
| CHAPTER 3 FROM GENETICS TO GENOMICS | 17 |
| 3.1 Genetics | 17 |
| 3.1.1 Information Structures | 17 |
| 3.1.2 Functional Structures | 19 |
| 3.1.3 The Universal Genetic Code | 19 |
| 3.1.4 The Gene Expression Process | 21 |
| 3.1.5 Modular Structure of Genomes and Evolution of Complexity | 21 |
| 3.1.6 Location Independence in Genomes | 22 |

| | | |
|---|---|-----------|
| 3.2 | Genomics | 23 |
| 3.3 | Proteomics | 24 |
| 3.3.1 | Proteome and Complexity | 25 |
| 3.3.2 | Separation of Roles | 25 |
| 3.4 | From Genotypes to Phenotypes | 26 |
| CHAPTER 4 PROTEOMICS APPROACH | | 28 |
| 4.1 | The Proteomics Approach to Evolutionary Computation | 28 |
| 4.1.1 | Representation | 28 |
| 4.1.2 | Functional Units | 29 |
| 4.2 | Fundamental Aspects of the Proteomics Approach | 31 |
| 4.3 | Proteome-Based Location Independent Representations | 31 |
| CHAPTER 5 THE PROPORTIONAL GENETIC ALGORITHM | | 33 |
| 5.1 | The Proportional GA Representation | 33 |
| 5.1.1 | PGA1 | 34 |
| 5.1.2 | PGA2 | 35 |
| 5.1.3 | PGA3 | 37 |
| 5.1.4 | PGA Further Enhancements | 38 |
| CHAPTER 6 MATHEMATICAL ANALYSIS | | 39 |
| 6.1 | Resolution Analysis | 40 |
| 6.1.1 | Notation | 41 |
| 6.1.2 | The PGA Mapping | 42 |
| 6.1.3 | Neutral Classes | 43 |
| 6.1.4 | Resolution | 44 |
| 6.1.5 | Examples | 45 |
| 6.2 | Random Sampling Analysis | 46 |
| 6.2.1 | Probabilities for Finding a Solution | 47 |
| 6.2.2 | The PGA Hypothesis | 48 |
| 6.2.3 | Average Probabilities Over All Solutions | 49 |
| 6.3 | Discussion | 51 |

| | |
|---|---------------|
| CHAPTER 7 EMPIRICAL ANALYSIS | 52 |
| 7.1 Experimental Setting | 52 |
| 7.1.1 Test Problems | 52 |
| 7.1.2 GA Configuration | 53 |
| 7.2 Formation of Buidling Blocks and Regulation of Length and Resolution Analysis | 54 |
| 7.2.1 Resource Allocation Test Problem | 54 |
| 7.2.2 Randomly Generated Target Allocations | 55 |
| 7.2.3 Formation of Building Blocks | 56 |
| 7.2.4 Regulation of Length and Resolution | 58 |
| 7.2.5 Generational Behavior of Best Fitness | 60 |
| 7.2.6 Sensitivity Analysis | 61 |
| 7.3 Performance Analysis | 64 |
| 7.3.1 Number Match Test Problem | 64 |
| 7.3.2 Symbolic Regression Test Problem | 64 |
| 7.3.3 Results | 65 |
| 7.4 Discussion | 68 |
| CHAPTER 8 EMERGENCE OF GENOMIC SELF-ORGANIZATION | 72 |
| 8.1 Emergent White Noise Behavior | 72 |
| 8.1.1 Symbolic Sequence Analysis | 73 |
| 8.2 Genomic Self-Similarity Analysis | 75 |
| 8.2.1 Self-Similarity Metric for Genomes | 76 |
| 8.2.2 Fitness Evaluation | 76 |
| 8.2.3 Settings | 79 |
| 8.2.4 Results | 80 |
| 8.3 Discussion | 80 |
| 8.4 Summary | 83 |
| CHAPTER 9 CASE STUDY: ADVANCE LIFE SUPPORT SYSTEM | 84 |
| 9.1 Background | 85 |
| 9.1.1 Advanced Life Support System Simulator | 85 |
| 9.1.2 Related Work | 87 |
| 9.2 Algorithm Descriptions | 88 |
| 9.2.1 Genetic Algorithm | 88 |

| | | |
|---|--|------------|
| 9.2.2 | Stochastic Hill Climbing | 88 |
| 9.3 | Fitness Definitions | 89 |
| 9.3.1 | The Optimization Problem | 89 |
| 9.3.2 | The Control Strategies | 90 |
| 9.3.3 | The Fitness Measures | 90 |
| 9.4 | Problem Representation | 90 |
| 9.4.1 | Binary Representation | 91 |
| 9.4.2 | Proportional Representation | 92 |
| 9.4.3 | How to Fairly Compare Binary and Proportional Representations | 94 |
| 9.5 | Comparative Experimental Analysis | 95 |
| 9.6 | Discussion | 98 |
| 9.6.1 | Algorithms: GA vs. SH | 98 |
| 9.6.2 | Control Strategies: $n = 1, 3, 5, 7, 9$ | 99 |
| 9.6.3 | Representations: Binary vs. Proportional | 99 |
| 9.6.4 | Resolution: P-Same vs. P-Half vs. P-Max | 100 |
| 9.6.5 | Performance: Comparing Binary and Proportional Representations | 101 |
| 9.7 | Summary | 102 |
| CHAPTER 10 FUTURE WORK | | 104 |
| 10.1 | Self-organization in Representations | 104 |
| 10.2 | Proteins as Representational Building Blocks | 104 |
| 10.3 | Building Blocks Interactions | 106 |
| CHAPTER 11 CONCLUSION | | 108 |
| 11.0.1 | Limitations | 109 |
| 11.0.2 | Contributions | 110 |
| CHAPTER A ADVANCED LIFE SUPPORT SYSTEM SIMULATOR | | 112 |
| A.1 | ALSS Simulation Mathematical Description | 113 |
| A.2 | Defining the Optimization Problem | 118 |
| A.3 | Defining the Fitness Functions | 119 |
| CHAPTER B ADDITIONAL TABLES AND FIGURES | | 120 |
| LIST OF REFERENCES | | 153 |

LIST OF TABLES

| | | |
|-----|--|-----|
| 3.1 | The universal genetic code. | 21 |
| 5.1 | PGA1 character assignment in a five user resource allocation problem. | 35 |
| 5.2 | Allocation of resources specified by example individual of length 50. | 35 |
| 5.3 | PGA2 character assignment in a five value problem. | 36 |
| 5.4 | Allocation of resources specified by example individual of length 50. | 36 |
| 5.5 | Allocation of resources specified by example individual of length 50. | 37 |
| 6.1 | Possible solutions using a PGA with binary alphabet and length 8. | 40 |
| 6.2 | Possible solutions using a PGA with alphabet size 3 and length 8. | 41 |
| 7.1 | GA parameter settings. | 53 |
| 7.2 | Results of PGA1 versus GA on finding randomly generated allocation values. Average (and standard deviation) over 100 runs. | 56 |
| 7.3 | Results of PGA1 versus GA on finding fixed resolution allocation values. Average (and standard deviation) over 100 runs. A hit is a run that achieves the fitness of 0.99 or higher. All fitnesses are 0.95 or higher. | 58 |
| 7.4 | Crossover and mutation rates for the 15 experiments of the sensitivity analysis | 62 |
| 9.1 | Vector of control parameters, \vec{c}^t , of ALSS simulator. | 85 |
| 9.2 | Constant parameter settings for all GA runs. | 88 |
| 9.3 | Proportional representation character assignments for ALSS problem. | 92 |
| 9.4 | Allocation of resources as specified by example individuals from Figures 9.3 and 9.4. | 93 |
| 9.5 | Summary of experimental results of comparing GA with binary and proportional representations to optimize the ALSS simulator in 45 different configurations.. | 101 |
| 9.6 | Summary of experimental results of comparing Proportional-SH and Binary-SH to optimize the ALSS simulator in 45 different configurations. | 102 |

| | | |
|------|---|-----|
| 10.1 | Summary of recent results related to biological building block organization, over 43 different organisms representing all three domains of life. | 107 |
| B.1 | Results of PGA1 and GA on finding randomly generated allocation values. Average and standard deviation (std) over 100 runs. | 121 |
| B.2 | Results of PGA1 and GA on finding a fixed test allocation of values: 1:2:4:2:1. Average and standard deviation (std) over 100 runs. | 122 |
| B.3 | Results of PGA2 and GA on the number matching problem. Average and standard deviation (std) over 100 runs. | 123 |
| B.4 | Results of PGA3 and GA on the number matching problem. Average and standard deviation (std) over 100 runs. | 124 |
| B.5 | Results of PGA2 and GA on the symbolic regression problem. Average and standard deviation (std) over 100 runs. | 125 |
| B.6 | Results of PGA3 and GA on the symbolic regression problem. Average and standard deviation (std) over 100 runs. | 126 |
| B.7 | PGA and GA on NASA ALSS simulator, optimizing mission productivity with strategy Γ_1 . Average and standard deviation (std) over 40 runs. | 126 |
| B.8 | PGA and GA on NASA ALSS simulator, optimizing mission productivity with strategy Γ_3 . Average and standard deviation (std) over 40 runs. | 127 |
| B.9 | PGA and GA on NASA ALSS simulator, optimizing mission productivity with strategy Γ_5 . Average and standard deviation (std) over 40 runs. | 127 |
| B.10 | PGA and GA on NASA ALSS simulator, optimizing mission productivity with strategy Γ_7 . Average and standard deviation (std) over 40 runs. | 128 |
| B.11 | PGA and GA on NASA ALSS simulator, optimizing mission productivity with strategy Γ_9 . Average and standard deviation (std) over 40 runs. | 128 |
| B.12 | PGA and GA on NASA ALSS simulator, optimizing mission duration with strategy Γ_1 . Average and standard deviation (std) over 40 runs. | 129 |
| B.13 | PGA and GA on NASA ALSS simulator, optimizing mission duration with strategy Γ_3 . Average and standard deviation (std) over 40 runs. | 129 |
| B.14 | PGA and GA on NASA ALSS simulator, optimizing mission duration with strategy Γ_5 . Average and standard deviation (std) over 40 runs. | 130 |
| B.15 | PGA and GA on NASA ALSS simulator, optimizing mission duration with strategy Γ_7 . Average and standard deviation (std) over 40 runs. | 130 |
| B.16 | PGA and GA on NASA ALSS simulator, optimizing mission duration with strategy Γ_9 . Average and standard deviation (std) over 40 runs. | 131 |
| B.17 | PGA and GA on NASA ALSS simulator, optimizing mission productivity and duration with strategy Γ_1 . Average and standard deviation (std) over 40 runs. | 132 |

- B.18 PGA and GA on NASA ALSS simulator, optimizing mission productivity and duration with strategy Γ_3 . Average and standard deviation (std) over 40 runs. 132
- B.19 PGA and GA on NASA ALSS simulator, optimizing mission productivity and duration with strategy Γ_5 . Average and standard deviation (std) over 40 runs. 133
- B.20 PGA and GA on NASA ALSS simulator, optimizing mission productivity and duration with strategy Γ_7 . Average and standard deviation (std) over 40 runs. 133
- B.21 PGA and GA on NASA ALSS simulator, optimizing mission productivity and duration with strategy Γ_9 . Average and standard deviation (std) over 40 runs. 134

LIST OF FIGURES

| | | |
|-----|---|----|
| 2.1 | Basic steps of a typical genetic algorithm. | 10 |
| 2.2 | Example of location dependent problem representation. Each parameter value is encoded at a fixed location on each individual. | 10 |
| 2.3 | Example of the floating representation. | 13 |
| 3.1 | DNA, the biological unit of information storage, schematic and double helix structure | 18 |
| 3.2 | Protein, the biological functional building block, schematic showing the three-dimensional folding on primary, secondary, tertiary and quaternary protein structures. | 20 |
| 4.1 | Typical representation mapping. Problem representation is based on the individual's genome. | 29 |
| 4.2 | Evolutionary algorithms encode instances of solutions in artificial genomes. Designing this encoding is known as the problem representation. In the proteomics approach to evolutionary computation, we propose intermediate structures analogous to a proteome and related interaction networks. Basic building blocks are represented in the genome but the ultimate functionality of the representation will emerge from the building block network of interactions. | 30 |
| 4.3 | Proteome-based location independent representation: problem representation is based on the individual's proteome (multiset of proteins). | 32 |
| 6.1 | Representation and Solution Spaces | 42 |
| 6.2 | Neutral Classes in the PGA mapping | 43 |
| 6.3 | Solution point p_s and its associated neutral class $\mathcal{N}(p_s)$ | 47 |
| 7.1 | GA -vs- PGA1 on resource allocation: Fitness of best solution found averaged over 100 run and 95% confidence intervals. | 55 |
| 7.2 | Center of gravity of characters on a population of PGA individuals. | 57 |
| 7.3 | Range of characters on a population of PGA individuals. | 57 |
| 7.4 | Example of individual with repeated sequences. | 59 |

| | | |
|------|--|----|
| 7.5 | Best fitness per generation obtained by the GA, PGA-255, PGA-var, PGA-255-nc, and PGA-var-nc for the resource allocation problem using randomly generated allocation values. | 60 |
| 7.6 | Best fitness per generation obtained by the GA, PGA-255, PGA-var, PGA-255-nc, and PGA-var-nc for the resource allocation problem using a fixed test allocation(1:2:4:2:1). | 61 |
| 7.7 | GA -vs- PGA1 on resource allocation: absolute best fitness found over 20 runs for experiments e1 to e15. Crossover and mutation rates used on each experiment are shown in table 7.4 | 63 |
| 7.8 | GA -vs- PGA1 on resource allocation: fitness of best solution found averaged over 20 runs and 95% confidence intervals for experiments e1 to e15. Crossover and mutation rates used on each experiment are shown in table 7.4 | 63 |
| 7.9 | GA -vs- PGA2 on number match: Fitness of best solution found averaged over 100 runs and 95% confidence intervals. | 65 |
| 7.10 | GA -vs- PGA2 on number match: Number of runs out of 100 that find solutions with fitness 0.99 or higher. | 66 |
| 7.11 | GA -vs- PGA3 on number match: Fitness of best solution found averaged over 100 runs and 95% confidence intervals. | 67 |
| 7.12 | GA -vs- PGA3 on number match: Number of runs out of 100 that find solutions with fitness 0.99 or higher. | 67 |
| 7.13 | GA -vs- PGA2 on symbolic regression: Fitness of best solution found averaged over 100 runs and 95% confidence intervals. | 68 |
| 7.14 | GA -vs- PGA2 on symbolic regression: Number of runs out of 100 that find solutions with fitness 0.99 or higher. | 69 |
| 7.15 | GA -vs- PGA3 on symbolic regression: Fitness of best solution found averaged over 100 runs and 95% confidence intervals. | 69 |
| 7.16 | GA -vs- PGA3 on symbolic regression: Number of runs out of 100 that find solutions with fitness 0.99 or higher. | 70 |
| 8.1 | White(A), Pink(B) and Brownian Motion(C) noises and their spectral densities, S(f). | 73 |
| 8.2 | Log-log spectral density plots of the best of the last generation individuals. From the bottom up, symbols A and B for PGA on number matching, on symbolic regression and on dynamical system control problems. The graphics have been offset for clarity. | 74 |

| | | |
|-----|---|----|
| 8.3 | Predicted genomic self-similarity in populations that have converged to the optimal fitness of 1. (A) Ideal self-similar case: all genomic segments have fitness of 1 regardless of their size. (B) No self-similarity case: segments have random fitness with average equal to the fitness median. (C) PGA expected self-similarity: segments gradually reduce in fitness as they get smaller. Experimental results resemble the ideal case (see Figure 8.4). | 77 |
| 8.4 | PGA best, PGA average, and GA baseline comparison of average fitness of genome segments($\hat{f}_r(g_{<1,L>})$) of sizes(r) L(entire genome), L/2(half genome), L/4, ... , and L/512 on a simple number matching problem averaged over 20 runs with 95% confidence intervals. (Top: self-similarity) PGA segment fitness remains approximately equal to the entire genome fitness for segment sizes of L/2, L/4, ... until L/128. (Middle: no self-similarity) No GA segment has average fitness close to the entire genome fitness for any segment size. Baseline note: for clarity, only GA average is shown; GA best is not shown but has similar behavior. | 78 |
| 8.5 | Average encoded values 1 and 2 of genomic segments of size L/2 (half-genome) over 500 generations. The plots show averages and standard deviations over 20 runs. (A) PGA encoded values 1 and 2 have tight standard deviations that indicate that encoded values of PGA genomic segments of size L/2 converge. From Figure 8.4 we observe that their fitness converges to a near optimal solution, similar to the fitness of the entire genome L. (B) GA encoded values 1 and 2 have wide standard deviations that indicate that GA genomic segments of size L/2 do not converge. Furthermore, the encoded values of these segments appear to be randomly distributed. From Figure 8.4 we observe that their average normalized fitness is not similar to the entire genome, but near to the median 0.5 instead. We observe similar behavior for segments of size L/4 to L/128. | 81 |
| 9.1 | Basic transitions in the ALSS simulator. | 86 |
| 9.2 | A binary encoded individual and its corresponding encoded values. | 91 |
| 9.3 | An example proportional representation individual of length 40 with a single vector ($n = 1$). | 93 |
| 9.4 | Another example proportional representation individual of length 40 with a single vector ($n = 1$). Encodes equivalent solution as individual from Figure 9.3. | 93 |
| 9.5 | GA and SH with binary, proportional-same, proportional-half, and proportional-max representations using control strategies $\Gamma_1, \Gamma_3, \Gamma_5, \Gamma_7$, and Γ_9 to optimize ALSS (A) mission productivity, (B) mission duration, and (C) both. Fitness of best solution found averaged over 40 runs and 95% confidence intervals. | 96 |

| | | |
|------|--|-----|
| 10.1 | Genes, mRNA, and proteins as the basic building blocks of biological function. These basic components interact at many levels. Gene regulation networks, protein-protein interaction networks, and metabolic networks among others describe their interactions. From these interactions, complex high-level biological functionality emerge. Recently, it has been shown that these basic biological components self-organize into scale-free hierarchical networks with embedded modularity. It has also been shown that these type of networks are robust and error tolerant. These results seem to suggest that scale-free hierarchical networks with embedded modularity are a potentially good architecture for combining building blocks into high-level functionality in general. | 105 |
| B.1 | Fitness data (best, average, worst) over 100 runs for the proportional GA on the resource allocation domain using randomly generated allocations | 135 |
| B.2 | Genome length data (longest, average, shortest) over 100 runs for the proportional GA on the resource allocation domain using randomly generated allocations | 136 |
| B.3 | Best fitness data for the resource allocation domain using randomly generated allocations: (top) close look at the best fitness per generation over 100 runs, comparing: GA, PGA-255, PGA-var, PGA-255-nc, and PGA-var-nc; (bottom) absolute and average best fitness over 100 runs for algorithms 1 to 9: GA, PGA-255-nc, PGA-40-nc, PGA-var-nc, PGA-var-nc-parsimony, PGA-255, PGA-40, PGA-var, PGA-var-parsimony | 137 |
| B.4 | Fitness data (best, average, worst) over 100 runs for the proportional GA on the resource allocation domain using fixed test allocation values: 1:2:4:2:1 . . . | 138 |
| B.5 | Genome length data (longest, average, shortest) over 100 runs for the proportional GA on the resource allocation domain using fixed test allocation values: 1:2:4:2:1 | 139 |
| B.6 | Best fitness data for the resource allocation domain using fixed test allocation values: 1:2:4:2:1: (top) close look at the best fitness per generation over 100 runs, comparing: GA, PGA-255, PGA-var, PGA-255-nc, and PGA-var-nc; (bottom) absolute and average best fitness over 100 runs for algorithms 1 to 9: GA, PGA-255-nc, PGA-40-nc, PGA-var-nc, PGA-var-nc-parsimony, PGA-255, PGA-40, PGA-var, PGA-var-parsimony | 140 |
| B.7 | Fitness data (best, average, worst) over 100 runs for the proportional GA on the number matching domain using PGA2 | 141 |
| B.8 | Genome length data (longest, average, shortest) over 100 runs for the proportional GA on the number matching domain using PGA2 | 142 |

| | | |
|------|---|-----|
| B.9 | Best fitness data for the number matching domain using PGA2: (top) close look at the best fitness per generation over 100 runs, comparing: GA, PGA-255, PGA-var, PGA-255-nc, and PGA-var-nc; (bottom) absolute and average best fitness over 100 runs for algorithms 1 to 9: GA, PGA-40-nc, PGA-255-nc, PGA-var-nc-parsimony, PGA-var-nc, PGA-40, PGA-255, PGA-var-parsimony, PGA-var | 143 |
| B.10 | Fitness data (best, average, worst) over 100 runs for the proportional GA on the number matching domain using PGA3 | 144 |
| B.11 | Genome length data (longest, average, shortest) over 100 runs for the proportional GA on the number matching domain using PGA3 | 145 |
| B.12 | Best fitness data for the number matching domain using PGA3: (top) close look at the best fitness per generation over 100 runs, comparing: GA, PGA-255, PGA-var, PGA-255-nc, and PGA-var-nc; (bottom) absolute and average best fitness over 100 runs for algorithms 1 to 9: GA, PGA-40-nc, PGA-255-nc, PGA-var-nc-parsimony, PGA-var-nc, PGA-40, PGA-255, PGA-var-parsimony, PGA-var | 146 |
| B.13 | Fitness data (best, average, worst) over 100 runs for the proportional GA on the symbolic regression domain using PGA2 | 147 |
| B.14 | Genome length data (longest, average, shortest) over 100 runs for the proportional GA on the symbolic regression domain using PGA2 | 148 |
| B.15 | Best fitness data for the symbolic regression domain using PGA2: (top) close look at the best fitness per generation over 100 runs, comparing: GA, PGA-255, PGA-var, PGA-255-nc, and PGA-var-nc; (bottom) absolute and average best fitness over 100 runs for algorithms 1 to 9: GA, PGA-40-nc, PGA-255-nc, PGA-var-nc-parsimony, PGA-var-nc, PGA-40, PGA-255, PGA-var-parsimony, PGA-var | 149 |
| B.16 | Fitness data (best, average, worst) over 100 runs for the proportional GA on the symbolic regression domain using PGA3 | 150 |
| B.17 | Genome length data (longest, average, shortest) over 100 runs for the proportional GA on the symbolic regression domain using PGA3 | 151 |
| B.18 | Best fitness data for the symbolic regression domain using PGA3: (top) close look at the best fitness per generation over 100 runs, comparing: GA, PGA-255, PGA-var, PGA-255-nc, and PGA-var-nc; (bottom) absolute and average best fitness over 100 runs for algorithms 1 to 9: GA, PGA-40-nc, PGA-255-nc, PGA-var-nc-parsimony, PGA-var-nc, PGA-40, PGA-255, PGA-var-parsimony, PGA-var | 152 |

CHAPTER 1

INTRODUCTION

“Further research into intelligence of machinery will probably be very greatly concerned with searches... There is the genetical or evolutionary search by which a combination of genes is looked for, the criterion being the survival value. The remarkable success of this search confirms to some extent the idea that intellectual activity consist mainly of various kinds of search”

–ALAN TURING, “Intelligent Machinery”, 1948

As early as the very beginnings of the science of computation, in the times of Charles Babbage and his “analytical engine”, *search* has been recognized as an important conceptual tool for problem solving. Later, in the initial stages of modern computational theory and artificial intelligence, Alan Turing proposed various kinds of search as a means to achieve machine intelligence. Since then, search, in particular *heuristic search*, has played an important and historical role in artificial intelligence. In heuristic search, a computer seeks the answer to a problem by searching through the space of all possible candidate solutions using heuristics to guide the search toward promising areas. These heuristics are simply guiding principles obtained by using knowledge about the nature of the problem being solved.

Since the very beginning, there has been an intuitive understanding that, because heuristics are based on specific knowledge of the target problem, a universally best search algorithm, which is good for all problems, does not exist. It is only recently, however, that Wolpert and Macready (1997) have provided a formal proof. In their *No Free Lunch* theorems, they show, in essence, that an algorithm that performs well in one problem class is guaranteed to perform poorly on another. More importantly, they show that all search algorithms perform the same when averaged over all possible problems. As a result, there is a fundamental tradeoff between algorithm robustness across problems and algorithm performance.

Given that there is no universal search algorithm we are left with the task of designing algorithms that are specialized for a particular problem or a problem class. When there is an abundance of domain knowledge about the problem that we are trying to solve, a good deterministic algorithm can be designed and often its convergence to the optimal can be guaranteed. If the domain knowledge is incomplete, we still have the possibility of using some appropriate heuristic to guide the search. Unfortunately, for many interesting theoretical and practical problems, the space to be searched is too complex or too vast or both for us to devise an obvious heuristic or to collect a meaningful amount of domain knowledge. These

problems are difficult. For the worst of these cases, the best we can do is random or brute-force exhaustive search. Fortunately, many of these difficult problems share a key property that can be leveraged to guide the search: there is a *positive correlation between the form and quality of candidate solutions*. In other words, small changes in the form of a solution cause small changes in the resulting quality of the solution. *Stochastic search algorithms* are the class of search algorithms based on this key property. Stochastic search algorithms are more robust than other types of search algorithms in the sense that the key property they are based upon is quite general across problems and that they require less domain knowledge. They are weaker than other search algorithms in the sense that convergence to an optimum is usually not guaranteed. However, stochastic search algorithms often outperform random search and they have been shown to have wide applicability on this class of difficult problems where no strong heuristics are available, especially on computationally hard combinatorial problems such as planning, scheduling, constraint satisfaction, and satisfiability.

This thesis focuses on a subclass of stochastic search approaches called *evolutionary computation* (EC). More specifically, it examines a new kind of nature-inspired problem representation that attempts to capture the way in which nature deals with complexity. In addition to providing a better understanding of the impact of this new representation on EC, many of the results presented can be applied to other stochastic search algorithms including simulated annealing, stochastic hill-climbing, tabu search, and ant colony optimization.

1.1 Evolutionary Computation

Evolutionary computation methods or evolutionary algorithms (EA) are iterative stochastic search procedures inspired by the Darwinian concept of evolution by natural selection. These methods have proven to be particularly effective for solving computationally difficult problems. Evolutionary computation methods start with a population of randomly generated candidate solutions known as individuals. The quality of these individuals is evaluated using some fitness measurement. The best fit individuals are then selected to breed a new generation of individuals using operators such as mutation and crossover. New individuals are in turn evaluated, selected and bred. This procedure iterates until some criteria for optimality has been met or until the computational resources are exhausted.

Despite the fact that the characterization of the class of problems for which evolutionary computation is particularly suited for remains an open question, EC algorithms are increasingly used successfully in a variety of applications such as electronic circuit design, architectural design, antennas design, engineering optimization, factory job scheduling, network configuration, robot controllers, etc. Recently, there is a trend in the EC community to tackle human-competitive complex problems from which solutions can potentially be patented as innovative inventions on their own right.

1.2 Complexity Problem

As our society becomes more complex and computing power increases exponentially, we are reaching limits of complexity for computational problem solving that were not possible before. Examples of these limits are the amount of data generated by the genome sequencing project, the amount of data that can be gathered by modern radio telescopes, the complexity of software and hardware design projects, the amount of nodes and data in current communication networks, etc.

As complexity increases so does the need to create new tools to cope with and manage it. EC's generality, robustness, flexibility, and minimal dependence of domain knowledge make it particularly appropriate for these problems. Nevertheless, there are limits to what EC can currently accomplish in terms of complexity. As these methods are applied to increasingly difficult problems that require increasingly complex solutions, they face a number of problems: premature convergence to suboptimal solutions, stagnation of search in large search spaces, negative epistatic effects, and disruption of large building blocks, among others. We call the problem of overcoming current EC limitations in order to scale to high complexities the *EC complexity problem*.

While evolutionary algorithms constantly struggle with complexity, natural evolution deals with extremely complex organisms with ease. In fact, natural evolution seems to not have any problem evolving strikingly complex solutions. Our quest, therefore, is to localize and extract the principles that nature uses to deal with complexity and apply them to build better algorithms.

One of the most critical decisions of applying EC methods to a particular problem is the definition of the space to be searched. This space is defined by a mapping between the space of all possible problem solutions and the internal representation space actually explored by the algorithm. The choice of representation is crucial because it will determine not only the size and complexity of the search space but also the size and complexity of the structures representing candidate solutions. Recently, various *representational approaches* to deal with the complexity problem have been proposed including methods based on self-organization of representations, generative representations, developmental biology, morphogenesis, and cellular automata among others. Many representational approaches to the EC complexity problem have the underlying theme of using simple representational building blocks that interact in complex ways (growth, development, embryogenesis, self-organization of building blocks) to achieve complexity in function or design. We make this underlying principle explicit on the proteomic approach to evolutionary computation.

1.3 Proteomic Approach

Proteins are the basic building blocks of life. Evolutionary computation (EC) traditionally relies on gene analogous structures to represent candidate solutions for a given problem. We propose to investigate whether self-organization of protein analogous structures at the representation level can increase the degree of complexity and “novelty” of solutions obtainable using evolutionary search techniques. In this proteome-based representational approach, functional structures analogous to proteins act as complexity builders for a genome. As in many representational approaches, the proteome-based approach is motivated by the observation that nature seems to compress or at least bias the information encoded in the genome. The genome determines the size of the search space and it is the target of evolutionary changes. It is, however, the uncompressed version of the genes, the organism, that is tested for fitness in a given environment. Furthermore, there is recent biological evidence that suggest that the complexity of an organism is not necessarily correlated with the number of genes in their genome but more with the process of gene expression. For instance, there are more genes in the rice genome than in the human genome; and differences in the brain composition among humans and other primates are not so much due to differences in the genetic make up but to differences in amounts of expressed genes—proteins.

The biological processes motivating many representational approaches such embryogenesis and development, are far from understood, but there are some points that are widely accepted. The “central dogma” of molecular biology is one of them. The central dogma, in essence, says that DNA translates into proteins and that proteins, not genes, are the basic building blocks of biological functionality. In the proteomic approach, we use the central dogma as a guiding principle. We represent solutions not using a genome, as classic EC does, but base our problem representation on the protein complement of the genome—the proteome.

There are two fundamental aspects introduced by a proteome-based representation: (1) proteins interact in a three dimensional medium analogous to a soup or multiset; and (2) proteins are functional structures. The first aspect is foundational for the study of the second. This thesis focuses on analyzing the effects on EC introduced by the first aspect: using a “content” oriented structure such a multiset of proteins instead of a traditional “order” oriented structure such a string of genes as the basis for problem representation. On this representational analysis, the genetic operators act upon traditional strings of genes associated with the multisets of proteins. As a result, genetic operators are kept fixed. We call these types of representations *completely location independent* or *proteome-based location independent* representations. This analysis lays the groundwork for further study on representational approaches to the complexity problem, and to studies on the effects of using protein-like structures as a basis for EC problem representation.

1.4 Studying Proteome-based Location Independent Representations

The goal of this thesis is to study the basic effects of using proteome-based location independent representations in evolutionary computation methods.

1.4.1 Strategy and Methodology

A fundamental question about the proteomic approach, as well as other approaches to the EC complexity problem such the ones based on morphogenesis, development and self-organization is: *are these representations useful for scaling EC methods to high complexities? or what are these representations useful for?* Several studies in this areas show promising results. Nevertheless, there are still basic, fundamental questions about these approaches that remain unanswered such as: *is it feasible or effective or both to represent solutions using these approaches? and what are the effects on EC methods due to this representations?* Our strategy is to recast these abstract questions into a more controllable and simplified environment suitable for empirical and theoretical analysis. The first aspect of the proteome approach is realized by what we have called the proteome-based location independent representations. Therefore, we recast the fundamental questions as follows: *Are location independent proteome-based representation feasible? effective? for simple problems? complex problems? are they effective for a particular problem class? What are the effects of using location independent proteome-based representations compared with traditional representation in EC?* These questions and their answers are the central topics of this thesis.

Our method combines empirical and mathematical analysis. We define an empirical framework for study proteome-based location independent representations based on a canonical *genetic algorithm* (GA) that we called *proportional genetic algorithm* (PGA). We use this framework to analyze the comparative performance of these representations as well as to analyze the emergent genomic order. We use basic combinatorial and probability theories to analyze the basic effects of the genome to proteome mapping such as the genome and proteome space sizes, the quantity and sizes of neutral classes, the space requirements for representing the same problem on traditional and proteome-based approaches, and probabilities of finding solutions by random sampling in both approaches.

1.4.2 Contributions

This thesis contributes to the field of Evolutionary Computation in the following ways:

- It proposes a new approach to EC based on the central dogma of molecular biology. There are many approaches to scale EC to high complexities based on biologically inspired principles such as embryogenesis and developmental biology. The proteomic approach, however, captures and makes explicit the underlying principle of many of these representational approaches: *basing problem representation on basic building blocks that interact and self-organize into complex functions or designs*. This thesis provides the first comprehensive study of one of the basic aspects of this approach: proteome-based location independent representations.
- It defines an empirical framework with which to study proteome-based location independent representations using a genetic algorithm, the proportional genetic algorithm. The PGA proteome-based location independent representation, by definition, does not suffer from traditional EC hurdles associated with the arrangement of the genes or positional effect in a genome. This improvement is possible simply because proteome-based representations are based on “genomic content” whereas traditional representations are based on “genomic order”. Eliminating these hurdles delivers an algorithm that is less sensitive to implementation decisions and therefore more robust. Such a representation has the ability to self-organize into a genomic structure that appears to favor positive correlations between the form and quality of represented solutions. Additionally, it provides experimental evidence of some positive effects such as automatic adjustment of genomic resources and formation of self-similar building blocks.
- It provides theoretical and empirical evidence that a protein based representation is feasible and effective for practical problems. It shows theoretically the relationship between comparable genotypic and phenotypic space sizes needed for a fair comparison and provides substantial empirical evidence (380 million fitness evaluations over various problems and parameters settings) of the competence of this approach.
- It presents a case study for the proteomic approach to EC on a nontrivial and difficult real world problem: optimizing control of Advance Life Support Systems (ALSS), part of the NASA BioPlex project. This case study illustrates the effects of this approach on evolutionary algorithms and compares it to the effects of this approach on hill-climbing algorithms. This thesis gives a formalization of this problem, its optimization objectives, and its strategies in a framework given in Appendix A.

1.5 Overview

The remainder of this thesis proceeds as follows. The first three chapters introduce evolutionary computation and proteomics providing a review of the existing literature. Chapter 2 gives an introduction to evolutionary methods and genetic algorithms, and provides a literature

search on location independent representations and genotype to phenotype transformations. Chapter 3 gives a biological review from early genetics to current genomics and proteomics, with emphasis on proteins as functional structures.

The next two chapters introduce the proteomics approach to evolutionary computation. Chapter 4 describes the proteome approach as a new problem representation approach for evolutionary computation based on structures analogous to proteins. It also defines the aspect of the proteomic approach subject of analysis on this thesis: proteome-based location independent representations. Chapter 5 describes an empirical framework for studying proteome-based location independent representations based on a simple genetic algorithm that we call the proportional genetic algorithm.

The next four chapters provide a detailed analysis of proteome-based location independent representations. Chapter 6 gives a formal analysis of the proteome-based location independent genotype to phenotype mapping. Using combinatorics, we provide a closed form expression for the relationship between an individual's length and alphabet size that will result in comparable solution spaces. We also provide closed form expressions for: genotype and phenotype space sizes, the number of neutral classes in the genotype space, and the sizes of these neutral classes as function of the multiplicities of its elements. We conclude with a random sampling analysis that determines when proteome-based representation will have the advantage over traditional representations and shows that on average both approaches have similar chances of finding the solutions (statistical corroboration of the No Free Lunch theorem for this case). Chapter 7 provides an experimental analysis of the PGA. This analysis shows that PGA is as effective as or better than the GA on the problems tested and that PGA appears to be particularly suited for resource allocation problems or problems involving proportions or multisets. This chapter also shows that PGA genomes exhibit intriguing building blocks resembling the characteristics of the entire individual's genome. Chapter 8 presents an empirical and formal analysis of the emergent PGA genomic structure. It use standard spectral density techniques to determine that the PGA genomes of successfully evolved individuals resemble white noise, confirming results in the previous chapter that suggested evenly distributed symbols throughout the entire PGA genome. It determines empirically that self-similarity with respect to fitness emerges in the PGA genomes of successfully evolved individuals. These analyses have two important implications: that proteome-based representations provide a good positive correlation between form and quality of candidate solutions, and that building block disruption is minimal. Chapter 9 discusses the application of the proteome-based location independent representation as embodied in the PGA to control optimization of Advance Life Support Systems. Finally, Chapter 10 discusses future work and Chapter 11 provides some concluding remarks.

Appendix A gives a formalization of the advance life support system and Appendix B provides a full collection of tables and figures.

CHAPTER 2

BACKGROUND

This preservation of favorable individual differences and variations and the destruction of those which are injurious, I have called Natural Selection, or the Survival of the Fittest.

–CHARLES DARWIN, “The Origin of Species”, 1859.

2.1 Evolutionary Computation

Evolutionary Computation (EC) is a relatively young field. According to the *Handbook of Evolutionary Computation* [BFM97], the term itself was coined in 1991 as an attempt to bring together three related research communities: *Evolutionary Programming* [Fog62, Fog64], *Evolution Strategies* [Rec65] and *Genetic Algorithms* [Hol62, Hol75a, Gol89]. In the early 1960’s, these communities independently developed evolutionarily inspired computational paradigms that share the basis of any evolutionary process: reproduction, random variation, competition and selection. The earliest work on evolutionarily inspired computation can be traced back, at least, to the apparition of the first electronic computers, i.e., see Fraser [Fra57] and Friedberg [Fri58, FDN59].

In essence, evolutionary computation searches for good solutions to problems by testing large number of candidate solutions, selecting the “better” ones and randomly changing them to form a new set of candidates to be tested. This process repeats until a “good enough” solution is found or until the computational resources are exhausted.

Evolutionary computation is inspired by the process of *evolution by natural selection* originally proposed by Darwin [Dar59]. The EC jargon is grossly adapted from terms used in Genetics, Population Genetics, Evolutionary Biology, Molecular Biology, Evolutionary Theory and related fields. In EC, a candidate solution to a problem is known as an *individual*. A collection of such individuals form a *population*. When individuals undergo variations to generate another individuals, they are said to be *breeding*. While testing an individual, we assign a quantitative measurement of its performance known as its *fitness*. The set of individuals created by replication and variation of the current population are known as the next *generation*. The overall process of finding a good solution using this technique is known as *evolving* a solution.

Evolutionary Computation searches for “good solutions” over the space of all possible candidate solutions, the *search space*. In many cases, the variations to generate new individuals are not performed on the search space itself but on an auxiliary space, the *representation space*.

In this case, the individuals of the population are members of the representation space and are known also as *genomes, genotypes, or chromosomes*. Elements of the search space are known as *phenotypes*, and the process of mapping a genotype (structure subject to variation) into a phenotype (candidate solution) is known as *morphogenesis*.

2.1.1 Genetic Algorithms

Genetic algorithms were proposed by Holland in the context of *adaptive systems*. Their initial motivation was twofold: to understand the principles of natural adaptive systems and to construct robust artificial adaptive systems that could successfully respond to unexpected environmental changes. In contrast, Evolutionary Strategies were initially developed to evolve optimization techniques and Evolution Programming, to evolve intelligent agents. Another fundamental difference is that the main source of variation on Genetic Algorithms is recombination or *crossover*. The central role of recombination in the GAs is initially explained and further analysed by Holland in the *Building Block Hypothesis* and in its famous *Schema theorem* [Hol75a]. For the other paradigms, the main source of variation is mutation.

The idea of a genotype to phenotype separation, as found in nature, has been a central part of the GA since its inception. Individuals are represented by structures called genotypes. GA operators manipulate the genotypes of each individual. After a decoding process, these genotypes are translated into phenotypic structures upon which fitness is evaluated. The GA genotypes are typically, but not always, restricted to bit strings. A segment or various segments of the genotype that represent a component of a solution are usually called *genes*.

Figure 2.1 shows the basic steps of a GA. The initial population may be initialized randomly or with user-defined individuals. The GA then iterates thru an evaluate-select-reproduce cycle until either a stopping condition is satisfied or the computational resources are exhausted.

In the following, we survey two central issues for our study: GA representations, focusing on *location independent representations* and *genotype-phenotype mappings*—the mapping between the representation space to the search space.

```

procedure GA
{
  initialize population;
  while termination condition not satisfied do
  {
    evaluate current population;
    select parents;
    apply genetic operators to
      parents to create offspring;
    set current population equal to
      be the new offspring population;
  }
}

```

Figure 2.1: Basic steps of a typical genetic algorithm.

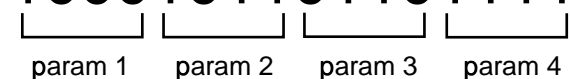
1000101101101111

 param 1 param 2 param 3 param 4

Figure 2.2: Example of location dependent problem representation. Each parameter value is encoded at a fixed location on each individual.

2.2 Related Work

2.2.1 Location Independent Representations

The importance of the arrangement of encoded information has been recognized since the introduction of the GA. Holland [Hol75a] recommended the use of an inversion operator as a way to rearrange information to find tightly linked building blocks. A *building block* is a collection of bits or regions on an individual that work together to affect the fitness of that individual. Early GA studies successfully used a representation in which both the value and the ordering of the genes are dynamically evolved by a GA [Bag67] and investigated the effects of gene order on a GA [Fra72]. Nevertheless since then, much of the community has, in a sense, “converged prematurely” to the neat, orderly, fixed encodings that are so common today.

Figure 2.2 shows an example of a typical GA problem representation. Information is encoded on an individual in a location dependent manner, e.g. bits 0 to 3 always represent parameter 1; bits 4 to 7, parameter 2; and so on. The ordering of the information on an individual is typically an arbitrary programmer decision. A difficulty with this type of representation arises due to crossover’s positional bias [ECS89]. Positional bias refers to the

fact that bits that are relatively far apart on an individual will be more likely to be disrupted (separated) by crossover than bits that are close together. Conversely, bits that are close together are more likely to be treated as atomic units (not separated) by crossover than bits that are far apart. If there is epistasis between various components of a problem solution, encoding those components close together will reduce the chance that crossover will disrupt those components. As encoded in Figure 2.2, the probability that crossover will disrupt `param1` and `param2` is much lower than the probability that crossover will disrupt `param1` and `param4`. The problem is that we do not always know which components are epistatic, and an arbitrary arrangement of the components of a solution on an individual may or may not elicit good performance from a GA.

A number of options have been investigated to deal with this problem, including operators such as inversion [Hol75a, Gol89] and uniform crossover [Sys89] which have little or no positional bias. A significant amount of research has also been devoted to examining ways in which problem representation may be designed to improve the GA search process. Numerous studies have focused on representations that allow a GA to dynamically evolve the arrangement of information on an individual. This class of representations provides flexibility at the expense of an increased search space. An alternative approach is the use of genotype-to-phenotype mappings to distinguish and transform between search space and solution space.

2.2.1.1 Bit Level Location Independence

The basic unit of information on a typical GA individual is a bit. Thus, the bit level is the lowest at which one would need to consider the arrangement of information. In some cases, problem encoding naturally places the bits that make up a building block in close proximity on an individual, e.g. each of the groups of four bits that encode for a single parameter value in figure 2.2. In other cases, a building block may consist of bits that are spread out across an entire individual. In the latter example, the ability to dynamically evolve the order of the bits on an individual would improve linkage among related bits and help identify and preserve building blocks.

Goldberg et al. [GKD89] developed the messy GA to investigate whether a GA is able to identify and recombine building blocks to form optimal solutions. The messy GA encodes individuals as a vector of pairs, where each pair specifies the location and value of a bit. Special rules deal with overspecified (duplicate) or underspecified (missing) bits. This encoding allows a GA to evolve both the value and the ordering of information on an individual. The fact that the physical ordering of information on an individual can be different from the interpreted ordering means that a messy GA can arrange related bits into close proximity to decrease the chance of disruption by crossover. Thus, a messy GA can rearrange information to form tightly linked building blocks which may then be recombined to create a complete

solution. Empirical tests indicate that a messy GA can solve difficult, deceptive problems, even when encodings are “loose”. In comparison, a simple GA, on average, is only able to achieve about 25% of the correct solution on the problems tested.

The fast messy GA [GDK93] improves upon the messy GA by reducing the computational efforts in the early phases of the messy GA. The gene expression messy GA [Kar96] enhances the messy GA to actively search for linkage relationships within a search space. The linkage learning GA [Har97] is an elegant extension which uses a modified representation and new operators that alter the relative positioning of genes to create building blocks.

While the messy GA and linkage learning GA allow for bitwise location independence in the problem representation, the mapping from a GA individual to a problem solution still retains some aspects of order: (1) the order in which bits appear in an individual affects their expression, and (2) some sort of reordering of the bits is typically required to decode an individual into a solution. A side effect of the latter is that each bit within a list of ordered bits has a specific meaning or use, e.g. the 2nd binary digit. As a result, all encoding bits must exist in order for a solution to be formed. In the messy GA, missing or duplicate bits require special repair or selection mechanisms to determine what is expressed. These special mechanisms can increase the complexity of the system and potentially introduce biases into the search process. The linkage learning GA eliminates some of these biases by maintaining copies of all bits in each individual.

2.2.1.2 Group Level Location Independence

Many problems such as the example in figure 2.2 use multiple bits to encode the component values of their solutions. These groups of bits are obvious building blocks which would likely benefit from a tight encoding. Combinations of these basic building blocks or genes may also create higher level building blocks. As a result, the ordering and arrangement of groups of bits (*basic building blocks*) has also been an active area of research.

Wu and Lindsay [WL96] developed the floating representation to study the maintenance of diversity in a GA. This representation assigns a tag to each parameter or basic building block in a solution. The fitness function then searches each individual for all tags and retrieves the value of each parameter from the bits immediately following each tag. Figure 2.3 shows an example of how a problem may be encoded with the floating representation. In this example, the tag for `param2` is 1101 and the tag for `param3` is 0111. The floating representation allows basic building blocks to be encoded anywhere on an individual while keeping the component bits of a basic building block together. This representation allows for overlapping building blocks which could reduce the amount of resources required to specify a solution. Like the messy GA, special rules are necessary to deal with overspecification and underspecification of values. Studies indicate that a GA using the floating representation is

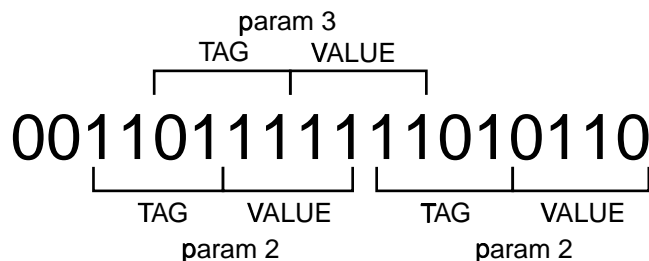


Figure 2.3: Example of the floating representation.

able to retain greater diversity in the population and that genetic operators in this system have more exploratory power [WD99].

Soule and Ball [SB01] investigated an encoding similar to the floating representation focusing on the idea of reading frames. Tags were used to delimit genes which could appear anywhere on an individual. These studies were conducted to investigate the likelihood and consequences of evolving overlapping genes. While performance varied from problem to problem, the GA was able to evolve genes much longer than the individuals encoding them.

Burke et al. [BDG98] investigated the evolution of the Virtual Virus (VIV) which encoded genes as location independent character segments. This work focused on investigating the factors that affect the evolved size and fitness of individuals and on understanding the use of non-coding regions in a GA. Empirical studies produced clear evidence of regions switching between coding and non-coding status, adding support to the hypothesis that non-coding regions may serve as backups for coding regions by storing extra copies of useful information. Both coding and non-coding regions contained obvious building blocks (potential genes). In addition, the GA appeared to monitor the length of evolved individuals in response to parameter settings such as mutation rate. Variation in average length of individuals throughout a run suggested that the GA exploits longer individuals early in a run when extra resources are helpful in searching for potential solutions then fine tunes to shorter individuals later in the run in response to parsimony pressure.

Group-wise location independence suffers from some of the same weaknesses as bit-wise location independence. There is again the notion that each building block has a specific meaning or contribution and, as a result, a value for every building block is needed in order for a solution to be formed. Missing and duplicate building blocks require special treatment. In addition, the final arrangement of building blocks can be sensitive to early decisions in a GA run. Bad decisions early in a GA run could result in arrangements in which it is impossible to encode all building blocks.

2.2.1.3 Alternative Location Independent Atomic Units

Both the bit-level and group-level use fairly general atomic units that can be used to encode more complex information. An alternate method is to make the atomic units themselves more complex. A number of applications have successfully used such representations. Each unit is a larger, more comprehensive idea or value than a bit, each variation of the unit may be considered a “character”, and the alphabet size is typically much larger than two. The collection of units on an individual are “expressed” and work together to create a solution. Information not encoded on an individual cannot be and is not expressed. The ordering of the units on a GA individual should not effect the fitness of the individual.

A common example of such a representation is the evolution of rule sets [GRS90, WSA99]. Each individual represents an entire rule set for controlling robots and the basic atomic units are condition-action rules. Crossover exchanges groups of rules, but will not fall within a rule. Mutation and other related operators change individual rules. Each individual (rule set) is evaluated on the behavior of a robot using that rule set. Rules are selected to fire based on the similarity of the condition clause of a rule with the current conditions of the robot. Thus, the collection of rules in a rule set is expected to affect its performance. The ordering of the rules in a rule set has absolutely no effect on the interaction of the evolved rules when applied to a problem.

Another related representation is Franceschini et al.’s [FWM00] work on disaggregation. A GA is used to evolve the configuration of a group of entities which satisfies pre-specified rules. As there is no distinction among the entities, each GA individual simply encodes a vector of coordinates. The order in which these coordinates are arranged on an individual had no effect on the solution. The fitness evaluation simply considers the content or collection of coordinates encoded. The existence of a coordinate affects the solution, the location of that coordinate does not.

Representations which use complex atomic units can be very effective at exploiting the advantages of location independence. Their implementation, however, will often require development of complex representations and problem specific operators.

2.2.2 Genotype to Phenotype Transformations

Genotype to phenotype transformations or mappings have also been studied as a way to improve GA problem representations. Such transformations distinguish between the search space and the solution space of a problem and allow for the development of a search space that is amenable to the GA search process when a solution space is not. One might take advantage of such a distinction to develop genotypes which, for example, simplify or otherwise modify the search space, include redundancies which could provide the GA with “backup copies”

or a form of memory, include neutral classes (classes of multiple genotypes that map to a single phenotype) to increase connectivity in the solution space, or eliminate the generation of invalid solutions (and the need for repair or specialized operators). Some examples of approaches to studying genotype to phenotype transformations include the following.

Kargupta and Park [Kar01, KP01] examine transformations of a fitness function into Fourier space as a way distributing fitness evaluation and show that for a given class of functions, such evaluations may be performed in polynomial time. Redundant representations, which investigate mappings where only a subset of the encoded information on an individual is expressed, include diploidy and dominance mapping [Lew97, NW95, SG92, YA94], the structured GA [DM92], as well as many of the approaches discussed in the previous sections. A common side effect of redundant representations is the emergence of neutral variants. Studies of mappings that allow neutral variants indicate benefit in terms of diversity and efficiency [Ban94, SSE00a, VM00]. Genetic programming studies have investigated dynamic evolution of the genetic code (which essentially dictates the genotype to phenotype mapping) [KB99, KB01, OR00]. A GA has also been used to develop simulations of gene expression and regulation in artificial cellular environments [KO97].

2.2.2.1 Redundant Representations and Neutrality

Redundant representations are characterized by their many-to-one genotype to phenotype mappings, i.e., several genotypes map to the same phenotype. Despite the fact that redundancy in a representation has been traditionally regarded as a poor design practice [Rad91] and that compact representations are often preferred, the advantages and disadvantages of redundant encodings have been the subject of recent reconsideration [Jul99, Shi99, SSE00b, WW01]. This reconsideration is due, in part, to relatively recent developments in evolutionary theory and molecular evolutionary biology on the role of neutrality in evolution.

In redundant representations, genetic operators can produce genotypic variations that are neutral with respect to selection, i.e., variations which produce an offspring that share the same phenotype with its parent. The possibility of having these neutral operations induces so called *neutral networks*. A neutral network is a set of genotypes that map to the same phenotype and that are connected by a sequence of neutral operations.

The existence of neutral networks was originally proposed by Kimura in his neutral theory of molecular evolution [Kim68]. Due to the recent availability of abundant DNA sequence data, this theory has received a substantial body of evidence in support [Kim83, Nei87] and is now being accepted as the most accurate theoretical framework for evolution at the molecular level [Hug99]. The main tenet of the neutral theory is that most genotype variations are selectively neutral and that most changes, over evolutionary time, occur as a result of a random fixation of these variations (genetic drift) [Kim68, Kim83]. This *neutralist*

approach to molecular evolution as well as its implications has been subject of extensive study [Kim77, KC78, Kim80, Kim81, Kim87, Huy96, HSF96, Hug99]

The importance of selective neutrality as a significant factor in artificial evolution has receive increasing attention, i.e. see [TI02, ELA01, RB99, Bar98, NE98, Ban94]. There is experimental evidence that neutral networks appear also in complex artificial evolutionary landscapes, for instance: in on-chip hardware evolution [HT96] and in neural networks based robot control systems [SHO01].

2.2.2.2 Self-organization in Representations

There is recent interest in approaches that increase the complexity of solutions without increasing the complexity of genomes in order to better scale evolutionary search. These approaches all share some degree of implicit self-organization. For instance the concepts of developmental mappings [KFA99, KKS03, KSK03], implicit embryogenies [KB03b, Ben03], developmental biology [Ben04, Kum04], generative representations [HLP03, Hor03a, Hor03b], molecular computing [Con00], self-replicating sequences [GBB03, BDE99], cellular automata variations [MT04, Roc04], and constrained generating procedures [Hol98], all share, to some degree, this perspective in which some sort of self-organizing principle drives the representation of the problems.

CHAPTER 3

FROM GENETICS TO GENOMICS

“It now became possible to understand the hereditary variation that is found throughout the biological world and that forms the basis of the theory of evolution. Genes are normally copied exactly during chromosome duplication. Rarely, however, changes (mutations) occur in genes to give rise to altered forms most, but not all, of which function less well...”

–JAMES WATSON, *co-discoverer of the DNA*, “Molecular Biology of the Gene”, 1976

3.1 Genetics

Genetics, classically, is the science of heredity and focuses on the transmission of characters or “traits” from one generation to the next. Modern genetics, however, can be defined as the study of biological information in general. All living organisms store, replicate, and transmit biological information to the next generation. This biological information represents the most valuable asset a living organism inherits from its ancestors and is used for development, reproduction and above all for surviving in its environment.

We consider that the major tenets of modern Genetics, as of the turn of the twenty-first century, are well characterized by Hartwell [HHG00] as:

- Biological information is encoded in the DNA molecule.
- Biological function emerges primarily from protein molecules.
- All living forms are closely related.
- The modular construction of genomes allows for the relatively rapid evolution of complexity.

3.1.1 Information Structures

A DNA molecule is a long polymer that stores biological information digitally in subunits known as *nucleotides*. Each nucleotide consists of a deoxyribose sugar, a phosphate, and one

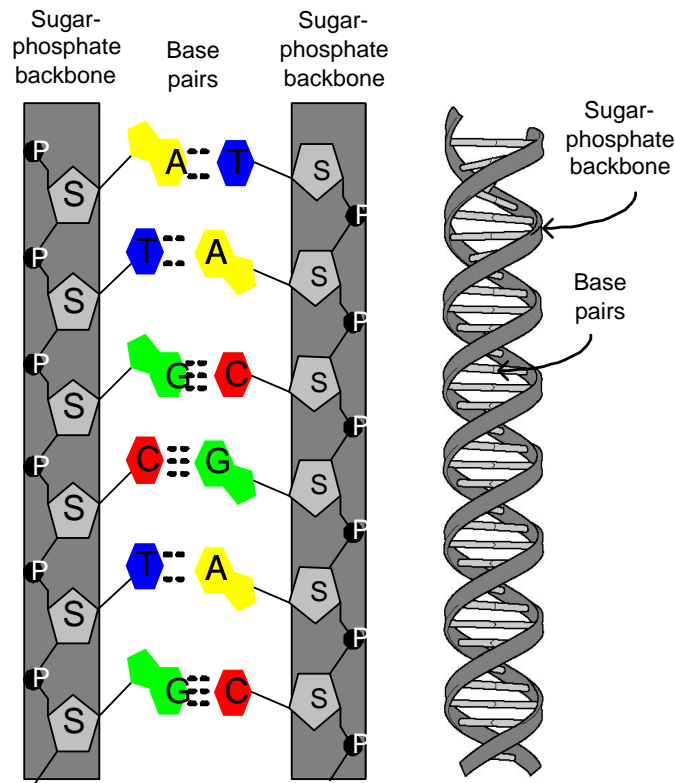


Figure 3.1: DNA, the biological unit of information storage, schematic and double helix structure

of four nitrogenous bases—adenine (A), thymine (T), cytosine (C), or guanine (G). These nucleotides are the building blocks of DNA. The DNA message is encoded as a long chain of these nucleotides, a *DNA strand*. The alphabet of this message consists of four letters, one for each nucleotide type—A, T, C, and G. A DNA molecule usually consists of two complementary DNA strands twisted about each other forming a double helix [Wat76]. The two strands carry complementary nucleotide chains with adenine always paired with thymine and guanine with cytosine. Each A-T and G-C pair is known as a *base pair*. Figure 3.1 shows a DNA schematic and the DNA double helix structure.

The complementary nature of these base pairs and the double stranded structure of the DNA allow for redundancy and self repair mechanisms that contribute to the correct storage and replication of the encoded biological information. The DNA structure appears also to be highly resistant to a range of conditions such as humidity, temperature and pressure. In fact, ancient DNA has been found to carry readable sequences: 8000-year-old DNA from human skulls found in Florida and 18-million-year-old DNA from fossilized leaves in an Idaho [HHG00]. Inside the cells of an organism, DNA is organized into *chromosomes*. The complete set of chromosomes in each cell of an organism constitutes its *genome*. A genome

is, in essence, the entire collection of an organism's biological information. Each cell of a multicellular organism contains a copy of the organism's genome.

The process of natural evolution has taken approximately 4 billion years to develop this amazingly efficient, highly specialized structure for information storage, replication, expression and evolution.

3.1.2 Functional Structures

The most elementary biological building blocks of functionality are proteins. Proteins are involved in almost every biological process that takes place in living organisms and are ultimately responsible for all of an organism's properties. Proteins are long molecules composed of one or more strings of amino acids chained together. Each of these individual chains is called a *polypeptide chain*. The encoded sequence of amino acids itself produces the necessary forces or molecular attractions that fold an amino acid string into the protein characteristic three-dimensional shape (see Figure 3.2).

3.1.3 The Universal Genetic Code

These amino acid chains are assembled directly from information encoded in segments of the DNA chain of base-pairs called *genes*. Three consecutive base-pairs are known as a *codon*. Each codon maps to a particular amino acid. The code that maps base-pairs into amino acids is virtually universal for all living organisms and is known as the *genetic code* (Table 3.1). The genetic code maps the 4 letter DNA alphabet into a 20 letter amino acid alphabet. Via this code, the order of bases in an organism's DNA determines the amino acid sequence of its corresponding proteins. The process of mapping itself is known as *gene expression*. Variations in the DNA encoded information can produce variations in the resulting amino acid sequence, which produce variations in the final three dimensional shapes of the proteins that are formed. Since proteins are typically composed of thousands of amino acids, there are virtually countless number of possible variations. The amazing variety of three-dimensional protein structure is the basis of the amazing variety of protein function, and the ability to produce new protein functionality is itself the basis of each organism's complex and adaptive behavior.

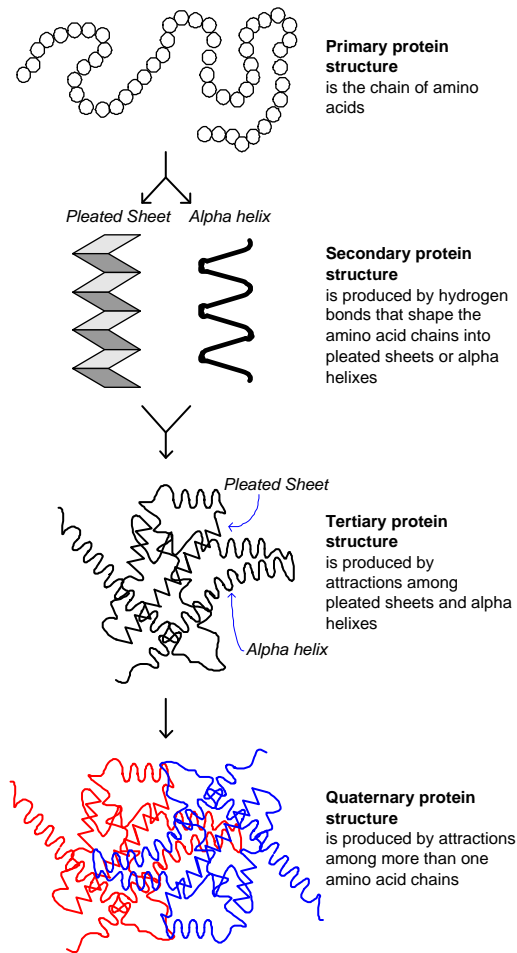


Figure 3.2: Protein, the biological functional building block, schematic showing the three-dimensional folding on primary, secondary, tertiary and quaternary protein structures.

Table 3.1: The universal genetic code.

| Amino acid | Codons |
|---------------------|------------------------------|
| Alanine (Ala) | GCU, GCC, GCA, GCG |
| Arginine (Arg) | CGU, CGC, CGA, CGG, AGA, AGG |
| Asparagine (Asn) | AAU, AAC |
| Aspartic acid (Asp) | GAU, GAC |
| Cysteine (Cys) | UGU, UGC |
| Glycine (Gly) | GGU, GGC, GGA, GGG |
| Glutamic acid (Glu) | GAA, GAG |
| Glutamine (Gln) | CAA, CAG |
| Histidine (His) | CAU, CAC |
| Isoleucine (Ile) | AUU, AUC, AUA |
| Leucine (Leu) | CUU, CUC, CUA, CUG, UUA, UUG |
| Lysine (Lys) | AAA, AAG |
| Methionine (Met) | AUG |
| Phenylalanine | UUU, UUC |
| Proline (Pro) | CCU, CCC, CCA, CCG |
| Serine (Ser) | UCU, UCC, UCA, UCG, AGU, AGC |
| Threonine (Thr) | ACU, ACC, ACA, ACG |
| Tryptophan (Trp) | UGG |
| Tyrosine (Tyr) | UAU, UAC |
| Valine (Val) | GUU, GUC, GUA, GUG |
| STOP | UAA, UAG, UGA |

3.1.4 The Gene Expression Process

The gene expression process involves two steps. First, the DNA of interest—the gene to be expressed—is transformed into *mRNA* (*messenger Ribonucleic Acid*) by the enzyme *RNA polymerase*. Messenger RNA is a single-stranded copy of the gene. This step is known as *transcription*. Next, the mRNA is translated into protein. Each codon in the mRNA is mapped to an amino acid in the protein by a mapping molecules called *transfer RNAs*. This mapping molecules act according with the genetic code. This last step is known as *translation*. With a few exceptions [Hao02, Rot00, AO02], the genetic code used in the translation process is the same for all living organisms on earth.

3.1.5 Modular Structure of Genomes and Evolution of Complexity

Recent technical advances in genomics, namely the numerous genome sequencing projects, has allowed for the analysis of complete genomes of several organisms. So far, the results of these analyses indicate that complex families of genes can arise by the duplication and divergence of a single primordial gene. After duplication, divergence occurs via mutations

and rearrangements of the copies of the primordial gene, resulting in a gene family with a broad spectrum of related functionalities. For instance, the hemoglobin family of genes, in humans and mice, produces five slightly different proteins. All of them are capable of transporting oxygen, but each one of them is specialized for some particular oxygen transportation need necessary at some stage of development. All of them are believed to have evolved from a primordial gene by means of duplication, mutation, and recombination. Gene duplication followed by mutation and recombination is the underlying evolutionary force that produces new genes, hence, new proteins with new functionalities. One partial explanation of the Cambrian Explosion ¹ lies in the hierarchical organization of biological information into chromosomes. Primordial genes duplicate and diverge onto gene families. Gene families similarly produce supergene families with hundreds of related genes and so on.

3.1.6 Location Independence in Genomes

There is a fundamental difference in what and how information is encoded in biological evolutionary systems and that encoded in computational evolutionary systems. Computational encodings tend to focus on the order of information. Biological encodings also emphasize the existence of information.

Ordered encodings are, of course, an important method by which biological systems store and express information. Proteins are defined by the order of their component amino acids. Genes are defined by the order of their component DNA molecules. The order in which genes appear on a genome can be important, for example, the mammalian β -globin gene cluster consists of one embryonic, two fetal, and two adult genes arranged on the chromosome in the order in which they are expressed during the development of the organism [WGW92].

From the overall genome viewpoint, however, the job of a genome is to specify the collective and relative amounts of genes needed within a cell. The expressed genes produce proteins which interact to create life. The type and amount of protein available for interaction directly affects the expressed phenotype of a cell or organism. Biological genomes are not divided into fields, each of which is assigned to produce a specific protein. Rather, they are more open-ended, with genes appearing where ever a viable encoding is found. The number of genes that comprise a solution is bounded only by the genome size and the number of different “encodings” is virtually endless. This viewpoint suggests that the existence or nonexistence of a gene has as much or more impact than the location of a gene. After all, the location of a gene can only matter if it exists.

¹The relatively short evolutionary time of 10 million years during which multicellular life forms diversified into countless different organisms, after it took eukaryotic cells almost 2 billion years to evolve from prokaryotic cells and multicellular organism 0.75 billion years to evolve from single-celled eukaryotes.

There are clear examples in biology which indicate that the existence of a gene allows it to produce a required protein regardless of its location. Curtis [Cur83] gives an example where a genetic defect (deletion) is masked by an extra copy of the missing gene on a different chromosome. The defect is expressed in the offspring which do not have an extra copy of the missing gene, possibly due to receiving that part of the chromosome from the other parent. Studies on the expression of X chromosome genes support the importance of accurate gene expression. Female mammals have two X chromosomes to male mammals' XY chromosomes. To prevent females from receiving a double dosage of X chromosome genes, one X chromosome is suppressed in all female cells [BYF62, Ohn70].

3.2 Genomics

Relatively recent technological advances in molecular biology, particularly in DNA sequencing have produced a shift of interest from the study of single genes to a more ambitious endeavor: the study of the entire set of genes that define an organism—the genome. Genomics was born as a science devoted to genome analysis. Genome analysis has gone from genome sequencing to genome expression and function. In this sense, Genomics is a natural progression from Genetics.

Two additional reasons have been argued for the shift from the study of genes to the study of genomes. First, unlike genomes, genes are difficult to define. At the beginning of the 1960's, a gene was a DNA fragment that coded for a protein. Nowadays, this definition has become more flexible to accommodate various facts. Among them are that some proteins are made from several genes; that genes are fragmented over wide genome spaces (exons and introns); that those fragments, in some cases, belong to various genes (regulatory sequences); and that several different proteins, each of them with different functionality, can be made from the same gene (differential splicing, multiple reading frames, RNA editing [Sco95]). Genomes, by contrast, have a simple definition: the complete biological information of an organism [Mor01]. Second, no gene acts alone. A gene was originally thought to be a single hereditary unit that produces a single observable feature in a organism. This simple definition of gene functionality has since been abandoned in favor of a looser interpretation of gene function. Genes can act cooperatively or antagonistically, with the final observable traits being the result of a complex interaction of the entire set of genes that composes the genetic material of an organism [Rid00]. Based on these reasons, some researchers [Beu95, Mor01] have argued for the use of the concepts of genomes and bits of genomes instead of genes.

The primary factor for the flourishing of genomics, however, is not the gene's identity crisis, but the advent of the Genome Era heralded by the remarkable success of the numerous genome sequencing projects and the wealth of information they have produced. Sequencing an organism's genome means spelling out the information encoded on its genome in terms of the four letter DNA alphabet. Projects of this nature were made possible by novel DNA

sequencing techniques that became available near the end of the twenty-first century. The first entire genome of a free-living organism was published on 1995 [PD01]² and, as of today, there are 92 complete genome sequences in the public domain and 475 genomes under analysis [BEK] The most remarkable genome sequenced, the *homo sapiens* genome, was published on 2001 independently by Venter and Lander [VO01, Lan01]. The success of these genome sequencing projects and the public availability of the data have triggered efforts to analyze the information now available. This information is already being exploited in a number of ways. Comparative genomics compares the gene sequences of a particular organism with all other genomes in order to identify differences that could account for important properties; structural genomics seeks to relate protein structure to gene sequences; and functional genomics tries to elucidate gene expression and function.

3.3 Proteomics

The huge amount of information produced by these projects, as well as the new techniques developed, have fueled not only the new science of Genomics but also the emerging field of Proteomics.

According to Palzkill [Pal02], proteomics is:

...a branch of functional genomics that has risen in response to the inevitable question posted by the genome sequencing projects, i.e., what are the functions of all the proteins? Proteomics can be defined as the large-scale study of protein properties such as expression levels, post-translational modifications and interactions with other molecules to obtain a global view of cellular processes at the protein level.

Proteomics focuses on analyzing the entire protein complement or *proteome* of an organism, as opposed to the genomics focus on the entire genetic content. The term proteome was first used circa 1995 by various researchers [WC95] in order to describe the protein complement of a genome³

²In the 95th meeting of the American Society of Microbiology, May 21–25, Washington DC, simultaneously by Craig Venter (TIGR) and Hamilton Smith (Johns Hopkins)

³The term genome was coined by H. Winkler in 1920. Winkler created it from the words GENes and chromosOME to denote the complete set of chromosomes and their genes. While artificial at its beginnings, genome has since found its way into the English language. The American Heritage Dictionary of the English Language defines it as “The total genetic content contained in a haploid set of chromosomes in eukaryotes, in a single chromosome in bacteria, or in the DNA or RNA of viruses” or simply as “An organism’s genetic material”. The word proteome, however, is still artificial and denotes, in accordance with the term genome, the complete set of chromosomes and their encoded proteins.

The proteome, unlike the genome, is not static. Each cell contains all the information necessary to build a new organism, but only a subset of the proteins encoded on its genome. Some genes that code for proteins that are essential for cell function are expressed in almost all cells, while other specialized proteins are only expressed in specific cell types [Lie02] Thus, every organism has one genome but multiple proteomes. We can say that the proteome is the genomic expression for a given set of environmental factors such as developmental state (time) and cell differentiation (space).

3.3.1 Proteome and Complexity

Genome sequencing projects have flooded the scientific community with vast amounts of data and often unexpected and revealing results. With regard to proteins and complexity, we found the following ones particularly intriguing: the rice genome contains more genes than the human genome [GO02]; humans and their closest evolutionary relative, chimpanzees, have 98.7 percent identical genomes [FO02]; and differences in brain composition among humans and other primates are not so much due to differences in genetic make up as to differences on gene expression [EO02]. The first two imply that the complexity of the organism is not correlated with the number of genes on its genome. The last one clearly correlates the proteome (expression of genes) with the organism's complexity.

3.3.2 Separation of Roles

The natural process of evolution can be seen as a never ending search for a set of genes that will fit a constantly changing environment. Nature developed a separation of functions 4 billion years ago when it evolved DNA and the genetic code. From that point on, this separation of labor between DNA and proteins has proven to be useful tool for evolution. Nature has found a competent information storage and replication structure, namely the genome, and a competent functionality builder structure, the proteome, as well as a way to map between them, the genetic code. The Proteomics Approach to Evolutionary Computation reflects this perspective.

3.4 From Genotypes to Phenotypes

Genetics, since Mendel’s early work on inheritance [Men65] , has focused on the study of genes.⁴ When Watson and Crick discovered the double helix shape of the DNA molecule (1953) and its self-replicating properties, the idea of genes found a concrete foundation at the molecular level as a unit of information storage and transmission. In this gene centered approach, a gene creates a phenotype (i.e., “visible” character). Alteration of a particular gene by mutation produces different “gene types” or alleles that can create different “phenotypes” or variations of the visible character associated with that gene. The set of all possible alleles for a given gene is known as the *genotype*, and the set of all variations in visible characters produced by the genotype is known as the *phenotype*.

According to the Oxford Dictionary of Genetics [KS97], the term phenotype is defined as the observable properties of an organism produced by the genotype in conjunction with the environment. In contrast with classical genetics where phenotypes are considered to be observable properties such as the color and shape of pea seeds, modern genetics defines phenotypes at various observable levels of gene expression. As a result, we can now distinguish between morphological, physiological, biochemical, and molecular phenotypes. Molecular phenotypes includes *protein phenotypes* and mRNA phenotypes. The above definition of a phenotype seems to imply a linear and progressive genotype-phenotype relationship, but this implication is not true. While the relation between a gene and its expression at the mRNA and protein levels is more or less linear,⁵ the relationship between genotypes and higher levels of expression, i.e. physiological or morphological phenotypes, is not. In fact, a phenotypic trait can be the product of more than one gene. Most likely, it is the product of several genes that contribute to the trait in different degrees in conjunction with environmental factors in a non-linear complex way. Furthermore, there are multiple levels of feedback between the genotype and phenotype. A gene expressed as a protein can interact with other molecules in the environmental vicinity to activate the expression of a different gene. This cycle of gene expression changing the environment and environment triggering expression of other genes contribute to the complexity and non linearity of the genotype to phenotype mapping. For instance, see [Ott04].

If we were, however, to define the functionality of a particular gene, on which level of expression, from mRNA to external traits, should we look? In other words, what is the basic functional structure associated with a given gene? As explained by Klose [Klo00], the DNA level—the information structure—provides us with no functional information. The mRNA level begins to provide some functional information because the concentration of mRNA strands correlates with the expression activity of their corresponding genes. This mRNA concentration however does not necessarily correlate with protein production. Hence, quantitatively, mRNA is also not very informative with regard to gene function. The next

⁴Gregor Mendel call them “factors”.

⁵In the sense that every protein is synthesized from one gene; however, if we take into consideration gene regulation, the production of proteins (expression of a genes) is regulated by multiple other genes.

level of gene expression, the protein level, exhibits a high degree of correlation with gene function, providing a significant amount of functional information. Proteins provide the molecular structures and properties needed to achieve functionality. Above the protein level, the process of gene expression becomes more complex. The path from genotype to phenotype enters the network of gene regulation and, in a more general sense, the network of metabolic pathways. These networks obscure the contributions of a particular genotype to a given phenotype. Many genes contribute to cascades of metabolic reactions that lead to higher level phenotypes, eventually leading to external genetic traits.

Consequently, while genes are the fundamental units of information storage and replication, proteins seem to be the most adequate functional structures associated to a given gene. In the path from genotypes to phenotypes, the functional importance of proteins is twofold. First, in isolation, proteins reflect to a great extent the basic functionality of their corresponding genes. Second, in conjunction with other proteins, they contribute, as basic building blocks, to the construction of increasingly complex functionalities.

CHAPTER 4

PROTEOMICS APPROACH

“The development of our species is more the result of changes in gene expression than the acquisition of new, specifically human genetic information”

“...The success of the first transgenic experiments thus came as a complete surprise: in most cases, if a gene was inserted somewhere—anywhere—into the genome, it had the same effect as in its endogenous position...These results have since been confirmed over and over again.”

–MICHEL MORANGE, “The Misunderstood Gene”, 2001

Representational approaches to the EC complexity problem share the following underlying principle. They propose representations based on *simple building blocks that self-organize into complex functions*. The proteomic approach makes this underlying principle explicit by characterizing these basic representational building blocks as proteins, the basic building blocks of life. Therefore, the proteomic approach serves as a unifying framework for many biologically-inspired representational approaches such the ones based on development, morphogenesis, and self-organization.

4.1 The Proteomics Approach to Evolutionary Computation

The proteomic approach to evolutionary computation is a representational approach to the EC complexity problem. It is inspired by the central dogma of molecular biology and based on the fact that proteins are the basic building blocks of life. This approach attempts to capture the way in which nature has evolved extremely complex solutions in response to complex environments with apparent ease.

4.1.1 Representation

Traditionally, EC problem representation is confined to defining the mapping between an internal representation space and the space of solutions (Figure 4.1). This mapping is crucial

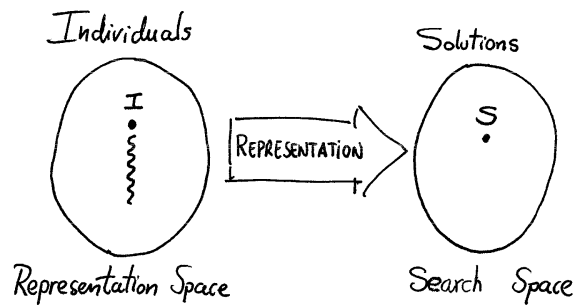


Figure 4.1: Typical representation mapping. Problem representation is based on the individual's genome.

to the success of evolutionary techniques since it determines the complexity of the space that the algorithm will explore as well as the complexity of the members of that space. In the proteomic approach, problem representation is not a simple mapping but rather a complex set of interactions among represented components (Figure 4.2). The encoded information in a genome are building blocks that assemble into a solution via these interactions. Therefore, there is a fundamental need for an interaction space that allows representational building blocks to self-organize; this interaction space for proteins is defined by the physics and chemistry at the molecular level. The simplest possible way to realize this space for initial studies is a multiset of symbols.

4.1.2 Functional Units

In traditional EC, it is the arrangement and value of the genes that completely determines candidate solutions. In the proteomic approach, the arrangement of the genes has no effect. Each gene encodes one type of functional representational building block. The interaction of these functional building blocks, in the interaction space, assembles candidate solutions. The amount of functional building blocks in the interaction space is determined by how many times a given gene is expressed. The required proportion of these building blocks is determined by gene expression mechanisms and the assembly of the candidate solution is done via self-organization of these functional building blocks in a pre-defined interaction space with predefined interaction rules.

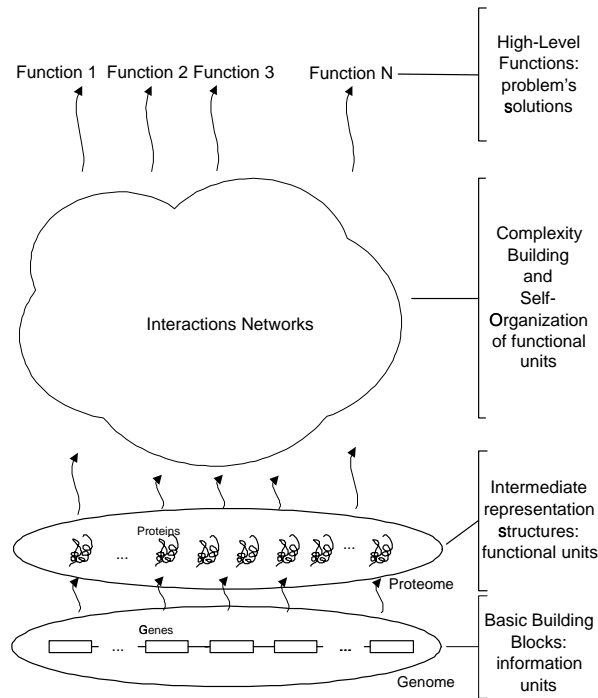


Figure 4.2: Evolutionary algorithms encode instances of solutions in artificial genomes. Designing this encoding is known as the problem representation. In the proteomics approach to evolutionary computation, we propose intermediate structures analogous to a proteome and related interaction networks. Basic building blocks are represented in the genome but the ultimate functionality of the representation will emerge from the building block network of interactions.

4.2 Fundamental Aspects of the Proteomics Approach

The proteomic approach captures the underlying principle of many representational approaches: basing problem representation on basic representational building blocks that interact and self-organize into complex functions or designs. This principle is made explicit by using protein-analogous structures as basic building blocks for problem representation.

The following are the two fundamental aspects of the proteomics approach:

- Proteins interact in a three dimensional medium analogous to a soup or multiset.
- Proteins are functional structures.

This thesis focuses on the study of the first issue for several reasons. First, the first aspect is foundational for the understanding of the second aspect because the functionality of the proteins is expressed as interactions in this protein multiset. Second, the way in which proteins interact to build complex biological functionality is currently subject of intense study [OB02, JTA00, Rav02, Rav02] but the principles involved are not yet well understood. What is well established is the central dogma of molecular biology upon which we base the study of the first aspect. Finally, there is a practical mathematical realization for the first aspect of the genome to proteome transition: string to multiset mapping.

4.3 Proteome-Based Location Independent Representations

Incorporating the first aspect into EC problem representation causes a fundamental departure from traditional EC methods. Using a “content” oriented structure such a multiset of proteins instead of a traditional “order” oriented structure such a string of genes as the basis for problem representation results in a completely location independent representation in which the location of the genes in the genome has no effect on the fitness of the solutions. We call these representations *proteome-based location independent* representations. In these representations, each individual is represented using a multiset of symbols from a given alphabet, and no functional aspects of proteins or their possible interactions are considered (see Figure 4.3). In the next chapter, we offer a detailed description of an implementation of a proteome-based location independent representation for rational numbers using a genetic algorithm. We call the resulting implementation the *proportional genetic algorithm (PGA)*. This implementation will be used throughout this thesis as a framework to study proteome-based location independent representations.

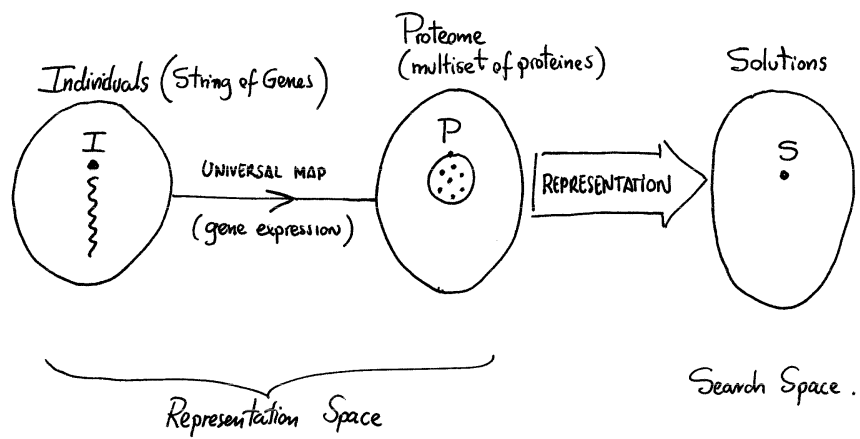


Figure 4.3: Proteome-based location independent representation: problem representation is based on the individual's proteome (multiset of proteins).

CHAPTER 5

THE PROPORTIONAL GENETIC ALGORITHM

“Gerrymandering is one of the great political curses of our single-member district plurality system and one that can only truly be lifted by adopting proportional representation.”

–DOUGLASS AMY, “Real Choices, New Voices”, 1993

“The towns you give ... are to be given in proportion to the inheritance of each tribe: Take many towns from a tribe that has many, but few from one that has few.”

–NUMBERS 35:8, “The Bible”

We introduce a GA with a new representation format which we call the proportional GA (PGA). Like a traditional GA, a PGA encodes solutions as linear strings. A PGA, however, is a multi-character GA that relies on the existence or non-existence of genes to determine the information represented. The information represented by a PGA individual depends only on what is present on the individual and not on the order in which it is present. As a result, the order of the encoded information can evolve in response factors other than the value(s) of the solution, for example, in response to the search for tightly linked building blocks. The PGA representation achieves location independence without the additional overhead of “tags” or “location markers” that many existing location independent encodings must use. In addition, a PGA is able to dynamically adapt the resolution of encoded information.

5.1 The Proportional GA Representation

If one considers the DNA level of encoding information in a genome, it appears to be very similar to a traditional GA encoding. The system is looking for a desired sequence or pattern of “characters” whose encoding provides a desired value. Characters are chosen from a fixed alphabet. The biological alphabet size happens to be four; the computational alphabet size is typically two. A correctly ordered sequence contributes positively to an individual’s fitness. An incorrect sequence contributes partial or no credit to an individual’s fitness. Order-based

encoding is an essential component of DNA information representation. All of the basic units of information that DNA can represent, i.e. genes, are encoded as ordered sequences.

If, however, we make the assumption that these basic units of information exist, and move up to a genome level viewpoint, we see an entirely different picture of what is being encoded. A biological genome encodes the collection of genes necessary to create a life form. A gene is expressed when it is translated into a protein product and can only be expressed if it exists on the genome. Multiple copies of a gene produces more of the protein for which it encodes. The interaction of the expressed proteins within a cell makes life possible. At this level, the key point is not the order of the encoded information, but the combination or content of the encoded information on a genome. This viewpoint requires, of course, that all of the “machinery” or encoding for the components (genes) that make up the content is already there. By focusing on a more abstract view of the genome as a description of what is available to interact within a cell, however, the existence or nonexistence of a gene has a much greater impact than the location of a gene.

The PGA representation is based on this idea that it is the content rather than the order of the encoded information that matters. We extend the idea of location independent atomic units from section 2.2.1.3 to a more generally usable encoding for integer and floating point numbers. Specifically, the PGA assigns one or more unique characters to each parameter or component of a solution. The value of a parameter is determined from the proportion of the characters assigned to that parameter as compared with the total number of characters in the individual, or from the relative proportions of the assigned characters of that parameter. Thus, characters that exist are “expressed” and, consequently, interact with other expressed characters. Characters that do not exist are “not expressed” and simply do not participate in the interactions of the expressed characters. This representation may be further extended with the addition of *non-coding* characters that are not associated with any parameters. These non-coding regions are beneficial for fine tuning purposes.

5.1.1 PGA1

Initial examination of this representation suggests that it is a perfect fit for resource allocation problems. Suppose we have a common pool of resources that must be divided among n users. With a PGA, we would use an n -character alphabet with one character assigned to each user. Each individual, regardless of length represents the total available resources. The proportion of each user’s character on an individual gives the percentage of resource allocated to each user. We will call this representation strategy PGA1.

For example, given a five user resource allocation problem, a PGA would require a minimum alphabet size of $a = 5$. We can assign characters as shown in table 5.1. The percentage

Table 5.1: PGA1 character assignment in a five user resource allocation problem.

| User U | Character assignment, $char(U)$ |
|----------|---------------------------------|
| U_1 | a |
| U_2 | b |
| U_3 | c |
| U_4 | d |
| U_5 | e |

Table 5.2: Allocation of resources specified by example individual of length 50.

| User U | Number of $char(U)$ | Allocation |
|----------|---------------------|------------|
| U_1 | 9 a's | 0.18 |
| U_2 | 15 b's | 0.3 |
| U_3 | 5 c's | 0.1 |
| U_4 | 11 d's | 0.22 |
| U_5 | 10 e's | 0.2 |

of total resource allocated to each user, $U_i, i = 0, \dots, a - 1$ is calculated by the equation

$$P_{PGA1}(U_i) = \frac{\text{number of } char(U_i) \text{ on individual}}{\text{length of individual}}.$$

As a result, the sum of all of the allocated resources must equal 1.0 (100% of the resources). A typical individual such as

accaebdbbeeddbabbddebaabdddebebadbecabebbedaacced

would generate the allocations shown in table 5.2. This individual is 50 characters long. Note that the location of the characters play absolutely no role in the evaluation of the individual. Thus, multiple individuals may generate the same solution and the individual

aaaaaaaaabbbbbbbbbbbbbbbcc

encodes the exact same solution as the first individual above.

5.1.2 PGA2

Unfortunately, many problems cannot be represented as pure resource allocation problems where the sum of the components is constrained to a fixed value. As a result, additional

Table 5.3: PGA2 character assignment in a five value problem.

| Value V | $positive_char(V)$ | $negative_char(V)$ | V_{min} | V_{max} |
|-----------|---------------------|---------------------|-----------|-----------|
| V_1 | A | a | 0 | 10 |
| V_2 | B | b | 0 | 10 |
| V_3 | C | c | 0 | 10 |
| V_4 | D | d | 0 | 10 |
| V_5 | E | e | 0 | 10 |

Table 5.4: Allocation of resources specified by example individual of length 50.

| Value V | Number of $positive_char(V)$ | Number of $negative_char(V)$ | $pct(V)$ | Expressed value |
|-----------|-------------------------------|-------------------------------|-----------|-----------------|
| V_1 | 9 A's | 2 a's | $9/(9+2)$ | 8.18 |
| V_2 | 3 B's | 9 b's | $3/(3+9)$ | 2.5 |
| V_3 | 3 C's | 5 c's | $3/(3+5)$ | 3.75 |
| V_4 | 4 D's | 4 d's | $4/(4+4)$ | 5.0 |
| V_5 | 7 E's | 4 e's | $7/(7+4)$ | 6.36 |

modifications must be made to the PGA representation to encode solutions for other types of problems.

We focus on the common problem format involving a search for a vector of values (either integer or floating point). Given a problem in which we are searching for $a = 5$ parameter values, $V_i, i = 0, \dots, a - 1$ and the range of each value is given by $V_{i,min}$ and $V_{i,max}$. We define a second PGA strategy called PGA2 in table 5.1.2. The number of “positive” and “negative” characters on an individual are used to calculate

$$pct(V_i) = \frac{positive_char(V_i)}{positive_char(V_i) + negative_char(V_i)}$$

where $i = 0, \dots, a - 1$ and $0.0 \leq pct(V_i) \leq 1.0$. The value of each parameter is calculated by the equation

$$P_{PGA2}(V_i) = V_{i,min} + pct(V_i) \times (V_{i,max} - V_{i,min}).$$

A typical individual of length 50 such as

AccBdDeeEbAbBDEccaAAAEebbEEEECCdbbbABCDEedcbaAAaddbA

would generate the parameter values shown in table 5.4. As with PGA1, the evaluation of a PGA2 individual is completely independent of the arrangement of characters. Thus, the individual

Table 5.5: Allocation of resources specified by example individual of length 50.

| Value V | Number of $positive_char(V)$ | Number of $negative_char(V)$ | $pct(V)$ | Expressed value |
|-----------|-------------------------------|-------------------------------|----------|-----------------|
| V_1 | 9 A's | 2 a's | 2/9 | 2.22 |
| V_2 | 3 B's | 9 b's | 3/9 | 3.33 |
| V_3 | 3 C's | 5 c's | 3/5 | 6.0 |
| V_4 | 4 D's | 4 d's | 4/4 | 1.0 |
| V_5 | 7 E's | 4 e's | 4/7 | 5.71 |

AAAAAAAAAaaBBBBbbbbbbbbCCCCccccDDDDddddEEEEEEEEeeee

also evaluates to the values shown in table 5.4.

5.1.3 PGA3

We attempt to simplify PGA2 by modifying the equation for $pct(V_i)$. In the new method called PGA3,

$$pct(V_i) = \begin{cases} \frac{positive_char(V_i)}{negative_char(V_i)} & \text{if } positive_char(V_i) < negative_char(V_i) \\ \frac{negative_char(V_i)}{positive_char(V_i)} & \text{otherwise} \end{cases}$$

where $i = 0, \dots, a - 1$ and $0.0 \leq pct(V_i) \leq 1.0$. The value of each parameter is still calculated by the equation

$$P_{PGA3}(V_i) = V_{i,min} + pct(V_i) \times (V_{i,max} - V_{i,min}).$$

With PGA3, both individuals

AccBdDeeEbAbBDEccaAAAEebbEEEECCDbbbABCDEedcbaAAAddbA

AAAAAAAAAaaBBBBbbbbbbbbCCCCccccDDDDddddEEEEEEEEeeee

generate the parameter values shown in table 5.5. Initial experiments suggest that the behavior of PGA2 and PGA3 are very similar.

5.1.4 PGA Further Enhancements

The adaptability of all three of the PGA variations described above can be further enhanced in two ways.

5.1.4.1 Variable Length Individuals

Because of the location independence of the PGA representation, it can easily be modified to work with variable length individuals. Variable length individuals allows a PGA to control the amount of computational effort it expends during a run. Using shorter individuals requires less computational effort. In addition, this modification would allow a PGA to control the resolution at which it encodes values. Longer individuals would allow finer resolution. The minimization of computational effort would likely need to be balanced against the need for adequate resolution.

5.1.4.2 Non-coding Regions

Non-coding regions can easily be added by including, as part of a PGA's alphabet, characters that are not assigned to any parameter values. These characters would not be directly involved in encoding information on an individual; however, they may affect the values represented by an individual by changing the overall length of individuals. Non-coding regions are expected to help a PGA "fine-tune" the values encoded on its individuals. In a fixed length system where the length is unsuitable for the required resolution, non-coding regions may be introduced to compensate for suboptimal length. This effect is expected to be more prevalent in PGA2 and PGA3 than PGA1 because of the nature of the problems on which PGA1 can be applied.

CHAPTER 6

MATHEMATICAL ANALYSIS

“Combinatorics ... began in ancient times. According to legend the Chinese Emperor Yu (c. 2200 B.C.) observed the magic square $\begin{bmatrix} 4 & 9 & 2 \\ 3 & 5 & 7 \\ 8 & 1 & 6 \end{bmatrix}$ on the back of a divine tortoise.”

—HERBERT RYSER, “Combinatorial Mathematics”, 1963, pp.1

This chapter provides answers for the following questions:

- Question: In order to represent solution spaces of equal size, what are the resources needed—in terms of genome length and alphabet—by proteome-based representations as compared to traditional genome-based representations? What is the role of the mapping function g (growth function) in this context?

Method: Quantitative analysis of strings to multisets transformations using basic combinatorics.

Relevance: Quantization analysis provides the basis for a fair comparison among approaches with different intermediate mappings. We state that, for a fair comparison, the algorithms should both target problem spaces that are identical in size regardless of the representation space size. In other words, they should be searching among the same number of solutions regardless of the number of repetitive representations available for a single solution.

- Question: What are the probabilities of finding a solution on a genome-based representation space as compared with a proteome-based representation space? Based on NFL results [WM95], if any, what are the problem classes that characterize proteome-based representations, i.e., problem classes for which proteome-based representations outperform other search approaches?

Method: Statistical analysis using random sampling.

Relevance: This analysis characterizes the influence of neutral classes on the proteome-based representation space. It also shows the differences of comparing binary and proteome-based representations in the case both have the same genotype space size and when both have the same phenotype space size. Finally, corroborates the no free

Table 6.1: Possible solutions using a PGA with binary alphabet and length 8.

| Number of 0's | Number of 1's |
|---------------|---------------|
| 8 | 0 |
| 7 | 1 |
| 6 | 2 |
| 5 | 3 |
| 4 | 4 |
| 3 | 5 |
| 2 | 6 |
| 1 | 7 |
| 0 | 8 |

lunch theorem, in the sense that, when randomly sampling representation spaces, both representations have in average the same chances to find solutions.

6.1 Resolution Analysis

We would like to compare the performance of a PGA with that of a canonical GA; however, their fundamentally different representations make it difficult to determine what would be a fair comparison. For example, a GA using a binary alphabet and individuals of length 8 can encode $2^8 = 256$ different solutions. Table 6.1 shows that a PGA using the same alphabet and length can encode 9 unique solutions. Obviously it will be easier to find a solution with the PGA; however, the PGA is representing a much smaller and coarser solution space, i.e. trying to solve an easier problem. Increasing the alphabet size of the PGA to three increases the number of unique solutions to 45 (see Table 6.2). The selected length and alphabet size affects the resolution of the solution spaces that each algorithm can represent.

We assert that, to fairly compare two search algorithms, they must search for a solution from among the same number of potential solutions, that is, within equal sized solution spaces. As a result, we will focus on determining metrics that would allow a GA and a PGA to represent a problem at the same resolution. To do so, we analyze the effects of the multiset approach of the PGA on its genotype-phenotype mapping; we analyse the formation of *neutral classes*; and produce an equation for determining the GA and PGA solution structures required to represent the same solution space.

Table 6.2: Possible solutions using a PGA with alphabet size 3 and length 8.

| Number of A's | B's | C's | Number of A's | B's | C's | Number of A's | B's | C's |
|---------------|-----|-----|---------------|-----|-----|---------------|-----|-----|
| 8 | 0 | 0 | 3 | 5 | 0 | 1 | 6 | 1 |
| 7 | 1 | 0 | 3 | 0 | 5 | 1 | 1 | 6 |
| 7 | 0 | 1 | 3 | 4 | 1 | 1 | 5 | 2 |
| 6 | 2 | 0 | 3 | 1 | 4 | 1 | 2 | 5 |
| 6 | 0 | 2 | 3 | 3 | 2 | 1 | 4 | 3 |
| 6 | 1 | 1 | 3 | 2 | 3 | 1 | 3 | 4 |
| 5 | 3 | 0 | 2 | 6 | 0 | 0 | 8 | 0 |
| 5 | 0 | 3 | 2 | 0 | 6 | 0 | 0 | 8 |
| 5 | 2 | 1 | 2 | 5 | 1 | 0 | 7 | 1 |
| 5 | 1 | 2 | 2 | 1 | 5 | 0 | 1 | 7 |
| 4 | 4 | 0 | 2 | 4 | 2 | 0 | 6 | 2 |
| 4 | 0 | 4 | 2 | 2 | 4 | 0 | 2 | 6 |
| 4 | 3 | 1 | 2 | 3 | 3 | 0 | 5 | 3 |
| 4 | 1 | 3 | 1 | 7 | 0 | 0 | 3 | 5 |
| 4 | 2 | 2 | 1 | 0 | 7 | 0 | 4 | 4 |

6.1.1 Notation

In the section, we will formalize the terms *individual* and *genotype* to refer, interchangeably, to fixed length strings over an alphabet. Typically, GA individuals are strings over a binary alphabet while PGA individuals are strings over a higher-arity alphabet.

We introduce the following notation: *representation (genotype) space*, $\mathcal{G}_{(l,\Sigma)} = \{g_i\}$, the set of all genotypes, g_i , of length l over alphabet Σ ; and *solution (phenotype) space*, $\mathcal{P} = \{p_j\}$, the set of phenotypes, p_j , which are strings of length l for a GA and *multisets*¹ of length l for a PGA. The mapping between spaces is $\mathcal{M} : \mathcal{G}_{(l,\Sigma)} \mapsto \mathcal{P}$, hence $p_j = \mathcal{M}(g_i)$. Figure 6.1 illustrates this notation.

Let \mathcal{A} be a GA with individuals of length $l_{\mathcal{A}}$ over an alphabet $\Sigma_{\mathcal{A}} = \{\sigma_1, \sigma_2, \dots, \sigma_{n_{\mathcal{A}}}\}$. Let the cardinality of the alphabet be $|\Sigma_{\mathcal{A}}| = n_{\mathcal{A}}$. Let \mathcal{A} 's representation space, solution space, and mapping be denoted by $\mathcal{G}_{(l_{\mathcal{A}},\Sigma_{\mathcal{A}})}$, $\mathcal{P}_{\mathcal{A}}$, and $\mathcal{M}_{\mathcal{A}}$, respectively.

Similarly, let \mathcal{B} be a PGA with length $l_{\mathcal{B}}$ and alphabet $\Sigma_{\mathcal{B}} = \{\sigma_1, \sigma_2, \dots, \sigma_{n_{\mathcal{B}}}\}$ with cardinality of $|\Sigma_{\mathcal{B}}| = n_{\mathcal{B}}$. Let \mathcal{B} 's representation space, solution space and mapping be denoted accordingly by $\mathcal{G}_{(l_{\mathcal{B}},\Sigma_{\mathcal{B}})}$, $\mathcal{P}_{\mathcal{B}}$, and $\mathcal{M}_{\mathcal{B}}$.

¹Multisets—also known as *bags*—denoted on this work by $\{|\cdot, \cdot, \cdot|\}$, are analogous to sets, except that they may contain multiple copies of identical elements. The number of occurrences of an element in the multiset is called the *multiplicity* of the element.

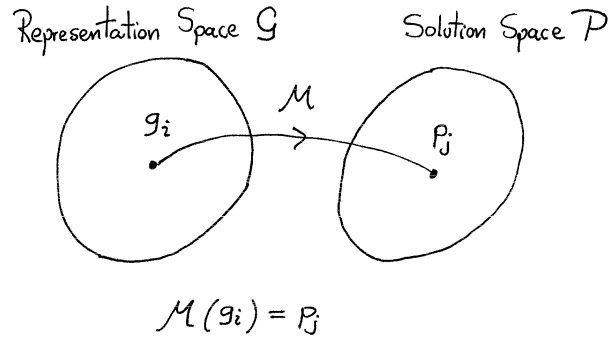


Figure 6.1: Representation and Solution Spaces

6.1.2 The PGA Mapping

For the GA, \mathcal{A} , the representation space and the solution space are the same size, $(n_{\mathcal{A}})^{l_{\mathcal{A}}}$, and the genotype-phenotype mapping, $\mathcal{M}_{\mathcal{A}}$, is trivial—one-to-one and onto from strings to strings—in fact, the identity mapping. For the PGA, \mathcal{B} , the size of the representation space is similarly $(n_{\mathcal{B}})^{l_{\mathcal{B}}}$, but the genotype-phenotype mapping, $\mathcal{M}_{\mathcal{B}}$, will map strings onto multisets instead of strings to strings. This mapping is, in general, many-to-one and onto, and leads to a solution space that is smaller than its corresponding representation space. The number of elements in each of these multisets is the same as the genotype length. The mapping, $\mathcal{M}_{\mathcal{B}}$, represents the essence of the PGA idea: every string is mapped according to the multiplicity of its elements regardless of the arrangement of the elements. In general, multiple strings will map onto the same multiset. The only exceptions are those strings that contain only one kind of element; i.e., for $l = 4$ and $\Sigma = \{a, b\}$, the string “*aaaa*” is the only one that maps onto the multiset $\{[a, a, a, a]\}$, while all of the following strings: “*aaab*”, “*aaba*”, “*abaa*”, and “*baaa*”, map onto the same multiset, $\{[a, a, a, b]\}$.

Using basic combinatorics results [Rys63], we can calculate the exact number of strings that map to each multiset as function of their multiplicities. For this effect, let $\eta(p_j)$ denote the number of strings on the genotype space which map to the multiset p_j as shown in Figure 6.2.

It is easy to show that:

$$\eta(p_j) = \frac{(l_{\mathcal{B}})!}{\prod_{i=1}^{n_{\mathcal{B}}} (|p_j|_{\sigma_i})!} \quad (6.1)$$

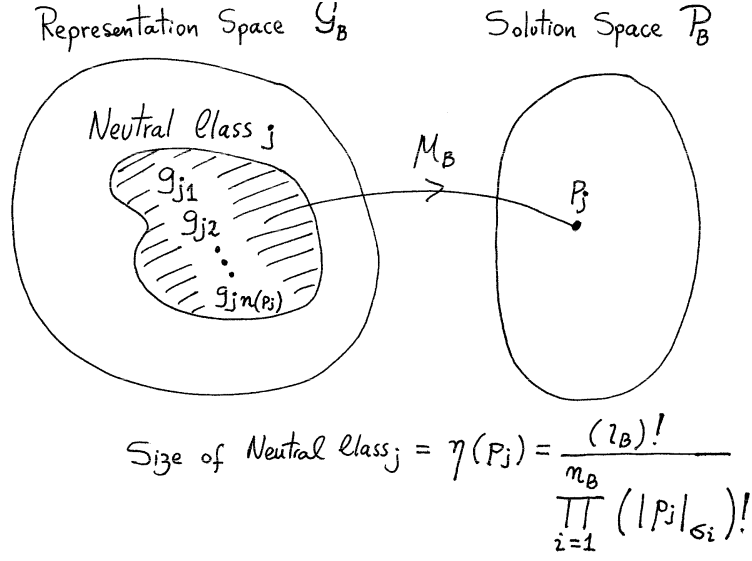


Figure 6.2: Neutral Classes in the PGA mapping

where, $p_j = \{\varsigma_{j1}, \varsigma_{j2}, \dots, \varsigma_{jk}, \dots, \varsigma_{jl_B}\}$ is the j^{th} multiset in \mathcal{P}_B ; $\varsigma_{jk} \in \Sigma_B$ for $k = 1, \dots, l_B$; $|p_j|_{\sigma_i}$ denotes the multiplicity (number of occurrences) of σ_i in the multiset p_j ; and $\sigma_i \in \Sigma_B = \{\sigma_1, \sigma_2, \dots, \sigma_{n_B}\}$.

In the same way, we can also calculate the total number of multisets produced by the PGA mapping for a given representation space. Using basic combinatorics, it is easy to show that the total number of multisets produced by the PGA mapping

$$\mathcal{M}_B : \mathcal{G}_{(l_B, \Sigma_B)} \mapsto \mathcal{P}_B$$

is

$$\binom{n_B + l_B - 1}{l_B}$$

As the set of all the multisets obtained by \mathcal{M}_B constitute the phenotype space \mathcal{P}_B , the size of this space, denoted $|\mathcal{P}_B|$, is given by:

$$|\mathcal{P}_B| = \binom{n_B + l_B - 1}{l_B} \quad (6.2)$$

6.1.3 Neutral Classes

The information represented by a PGA individual depends only on what is present on the individual and not on the order in which it is present. As a result, the PGA representation

is *fully redundant*, since every possible rearrangement of the same genes, and also any other arrangement that preserves the original *proportion* among genes expresses the same information². These neutral networks raise into consideration the scenario of an evolutionary dynamic in which populations rather than being driven to climb hills by a demanding selection pressure, traverse along connected networks of genotypes with equal fitness. Selection pressure becomes, in this scenario, the driving force behind sporadic jumps between these networks.

PGA redundancy allows neutral classes to form naturally in the PGA representation space. A *neutral class* is defined as the set of genotypes that maps to the same phenotype. The number of neutral classes as well as their corresponding sizes are determined by the mapping between spaces, \mathcal{M} . Each element of the phenotype space has a unique associated neutral class and vice versa. Note that the set of neutral classes is a partition of the genotype space.

For the PGA mapping, $\mathcal{M}_{\mathcal{B}}$, the size of the neutral class associated with the phenotype p_j is given by $\eta(p_j)$ in Equation (6.1), and the number of neutral classes formed in the genotype space by this mapping equals the size of the phenotype space, $|\mathcal{P}_{\mathcal{B}}|$, given by Equation (6.2).

6.1.4 Resolution

Let us consider a problem in which we are searching for a single parameter value with a given resolution, i.e., searching for a number between 0 and 100 with a resolution of 0.1. In this problem, the solution space would need to consist of at least $100 \times 10 = 1000$ elements to accommodate the required resolution of the search. As a result, an evolutionary algorithm applied to this problem must be able to represent at least 1000 elements in its solution space to find an accurate solution. In order to fairly compare two algorithms set to solve this problem, both of them should have the same solution space size.

Therefore, in order to compare GA \mathcal{A} and PGA \mathcal{B} , their solutions spaces should be of the same size, that is:

$$|\mathcal{P}_{\mathcal{A}}| \equiv |\mathcal{P}_{\mathcal{B}}| \tag{6.3}$$

The size of the phenotype space for GA \mathcal{A} is:

$$|\mathcal{P}_{\mathcal{A}}| \equiv (n_{\mathcal{A}})^{l_{\mathcal{A}}} \tag{6.4}$$

We use (6.2) and (6.4) to rewrite (6.3) and obtain:

²Note that since we only require the conservation of the relative gene proportions, the size of each neutral network produced is virtually unbounded for an unbounded variable length representation.

$$(n_{\mathcal{A}})^{l_{\mathcal{A}}} = \binom{n_{\mathcal{B}} + l_{\mathcal{B}} - 1}{l_{\mathcal{B}}} \quad (6.5)$$

Equation (6.5) is important because it describes the relationship between the lengths and alphabet sizes of an arbitrary GA and PGA that would ensure resulting solution spaces of equal size.

6.1.5 Examples

The resolution analysis verifies two facts about the PGA proteome-based location independent representation. First, this representation is redundant. As a result, PGA representations need larger representation spaces than non-redundant representations such binary encoding. Second, the PGA proteome-based location independent representation redundancy is not uniform over the representation space but is seriously biased. It over-represents individuals that contain the greatest possible diversity of symbols and under-represents individuals with low symbolic diversity. The following examples serve to illustrate these points.

Example 1. Consider a binary representation with individuals of length $l_{\mathcal{A}} = 8$ over an alphabet $\Sigma_{\mathcal{A}} = \{0, 1\}$, and $|\Sigma_{\mathcal{A}}| = n_{\mathcal{A}} = 2$. It is evident that the representation space and the solution space for this case are of size $2^8 = 256$. Now let us consider a PGA with a comparable solution space. Using equation 6.5, we calculate one possible solution pair: $l_{\mathcal{B}} = 10$ and $n_{\mathcal{B}} = 4$. Let us choose without loss of generality a particular four-letter alphabet: $\Sigma_{\mathcal{B}} = \{\alpha, \beta, \gamma, \delta\}$. For this setting, the size of the solution space is:

$$\binom{n_{\mathcal{B}} + l_{\mathcal{B}} - 1}{l_{\mathcal{B}}} = \binom{4 + 10 - 1}{10} = 286$$

This solution space is slightly larger than in the binary case (256) but a good approximation considering that $l_{\mathcal{B}}$ and $n_{\mathcal{B}}$ have to be integers. The question now is: how large is the resulting representation space? The answer is:

$$(n_{\mathcal{B}})^{l_{\mathcal{B}}} = 4^{10} = 1,048,576$$

Therefore, in this example, the PGA needs a representation space 3,666 times larger than a binary GA to represent the same number of points in the solution space due to redundancy.

Example 2. In the example above, what is the largest neutral class? And what is the smallest? We know from equation 6.7 that each string determines a different neutral class of a particular size. For instance, let's consider the following strings $\alpha\alpha\alpha\alpha\alpha\alpha\alpha\alpha$ and $\alpha\alpha\alpha\beta\beta\beta\gamma\gamma\delta\delta$. Using equation 6.7 we calculate the sizes of the neutral classes associated

with these strings:

$$\eta(\alpha\alpha\alpha\alpha\alpha\alpha\alpha\alpha\alpha) = \frac{(l_{\mathcal{B}})!}{\prod_{i=1}^{n_{\mathcal{B}}} (|\alpha\alpha\alpha\alpha\alpha\alpha\alpha\alpha\alpha|_{\sigma_i})!} = \frac{10!}{(10!)(0!)(0!)(0!)} = 1$$

The neutral class is of size one; therefore, there is no redundancy for this string. This result is obvious as there is only one possible ordering for a string of all α 's.

$$\eta(\alpha\alpha\alpha\beta\beta\beta\gamma\gamma\delta\delta) = \frac{(l_{\mathcal{B}})!}{\prod_{i=1}^{n_{\mathcal{B}}} (|\alpha\alpha\alpha\beta\beta\beta\gamma\gamma\delta\delta|_{\sigma_i})!} = \frac{10!}{(3!)(3!)(2!)(2!)} = \frac{3628800}{144} = 25200$$

The neutral class associated with string $\alpha\alpha\alpha\beta\beta\beta\gamma\gamma\delta\delta$ has 25200 elements because there are 25200 alternative orderings for this string. All of them map to a single point in the solution space, which is characterized by three α 's, three β 's two γ 's and two δ 's, regardless of their order.

The two strings we have chosen are in fact representatives of the smallest and largest neutral classes for this example. It is easy to show that the smallest neutral classes are always of size one and that they are characterized by equal-symbol strings such as $\alpha\alpha\alpha\alpha\alpha\alpha\alpha\alpha\alpha$. It is also easy to show that the largest neutral classes are characterized by the string that has the greatest possible number of distinct symbols present in equal proportion, such as $\alpha\alpha\alpha\beta\beta\beta\gamma\gamma\delta\delta$ in our example.

6.2 Random Sampling Analysis

Consider the point $p_s \in \mathcal{P}$ to be a distinguished point in the phenotype space as shown in Figure 6.3. Let us call it the solution point.³

Let $g_s \in \mathcal{G}$ be a point in the representation space that maps to p_s . Let $\mathcal{N}(p_s)$ be the set of all such points, $g_{s_i} \in \mathcal{G}$, that map to p_s . Therefore, $\mathcal{N}(p_s)$ is the *neutral class* associated with p_s .

Since there is exactly one neutral class associated with each element of \mathcal{P} , we have $|\mathcal{P}|$ neutral classes due to the mapping. The number of elements in each neutral class, $|\mathcal{N}(p_j)|$, depends on the specifics of the mapping \mathcal{M} .

³This point represents the phenotype with highest fitness, given a fitness function defined over the space \mathcal{P} .

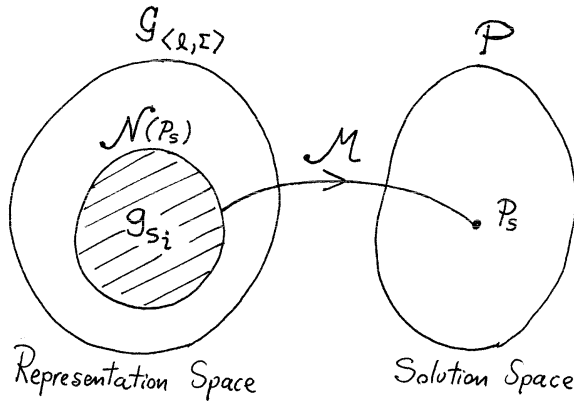


Figure 6.3: Solution point p_s and its associated neutral class $\mathcal{N}(p_s)$

6.2.1 Probabilities for Finding a Solution

The probability of selecting a genotype that maps to p_s by random sampling is given by:

$$Pr(p_s) = \frac{\text{Number of genotypes that map to } p_s}{\text{Total number of genotypes}}$$

which can be rewritten as:

$$Pr(p_s) = \frac{|\mathcal{N}(p_s)|}{|G_{(l, \Sigma)}|} \quad (6.6)$$

where $|\mathcal{N}(p_s)|$ is the size of the neutral class associated with p_s .

Let us consider again GA \mathcal{A} and PGA \mathcal{B} . From the previous discussion we know that the number of the neutral classes is the same as the number of elements in \mathcal{P} . As a result, $|\mathcal{P}_{\mathcal{A}}|$ and $|\mathcal{P}_{\mathcal{B}}|$ give us the number of neutral classes for \mathcal{A} and \mathcal{B} .

The number of strings in each class for \mathcal{A} is trivial to compute. Since the mapping, $\mathcal{M}_{\mathcal{A}}$, is one-to-one and onto, there is exactly one string in each class. For \mathcal{B} , the size of a neutral class is given by Equation (6.1) and depends on the multiplicity of the associated multiset. This multiplicity is the same as the multiplicities of each of the strings in the class defined by the multiset. Hence, we can expand our definition of η to account for multisets as well as for strings in the following way:

$$\eta(g_{s_k}) = \frac{(l_{\mathcal{B}})!}{n_{\mathcal{B}} \prod_{i=1} (|g_{s_k}|_{\sigma_i})!} \quad (6.7)$$

where $g_{s_k} \in \mathcal{G}_{\langle l_B, \Sigma_B \rangle}$.

The size of the neutral class associated with p_s is given by:

$$|\mathcal{N}(p_s)| = \eta(g_{s_k}) \quad (6.8)$$

We use Equation (6.6) to obtain the probabilities $Pr_{\mathcal{A}}(p_s)$ and $Pr_{\mathcal{B}}(p_s)$:

$$Pr_{\mathcal{A}}(p_s) = \frac{1}{(n_{\mathcal{A}})^{l_{\mathcal{A}}}} \quad (6.9)$$

$$Pr_{\mathcal{B}}(p_s) = \frac{\eta(g_{s_k})}{(n_{\mathcal{B}})^{l_{\mathcal{B}}}} \quad (6.10)$$

6.2.2 The PGA Hypothesis

In order for PGA \mathcal{B} to have an equal or better probability of finding a solution (p_s) than GA \mathcal{A} , the following inequality must hold:

$$Pr_{\mathcal{B}}(p_s) \geq Pr_{\mathcal{A}}(p_s) \quad (6.11)$$

We call Equation (6.11) the *PGA hypothesis*. It simply states that the probability of randomly selecting a genotype that maps to the solution is equal or higher in a PGA than in a GA.

Substituting (6.9) and (6.10) into (6.11) and rearranging the terms we get:

$$\eta(g_{s_k}) \geq \frac{(n_{\mathcal{B}})^{l_{\mathcal{B}}}}{(n_{\mathcal{A}})^{l_{\mathcal{A}}}} \quad (6.12)$$

which is a general condition for the PGA hypothesis to hold. This equation depends on parameters such as length and alphabet cardinality, as well as depends on the choice of the desired solution p_s .

Observe that for the case in which \mathcal{A} and \mathcal{B} have the same sized representation space, $|\mathcal{G}_{\langle l_{\mathcal{A}}, \Sigma_{\mathcal{A}} \rangle}| \equiv |\mathcal{G}_{\langle l_{\mathcal{B}}, \Sigma_{\mathcal{B}} \rangle}|$, Equation (6.12) reduces to:

$$\eta(g_{s_k}) \geq 1 \quad (6.13)$$

which always holds. Hence, in this case, subject to the choice of p_s , a PGA always has a better or equal chance of finding a solution than a GA.

In the case where $|\mathcal{P}_A| \equiv |\mathcal{P}_A|$, Equation (6.12) reduces to:

$$\eta(g_{s_k}) \geq \frac{(n_{\mathcal{B}})^{l_{\mathcal{B}}}}{\binom{n_{\mathcal{B}}+l_{\mathcal{B}}-1}{l_{\mathcal{B}}}} \quad (6.14)$$

In this case, either the PGA or the GA could have the advantage, depending on the choice of p_s .

6.2.3 Average Probabilities Over All Solutions

Since the previous results depend heavily on the choice of p_s , let us analyze the average of all possible choices of p_s . Note that p_s affects the above equations by changing the size of the associated neutral class. Consequently, to analyze the average behavior of all possible choices of p_s , we will compute the average size of all of the neutral classes (all possible choices of p_s).

Let $\hat{\eta}$ denote the average size of all neutral classes for a given representation space:

$$\hat{\eta} = \frac{\sum_{\{g_r \mid \mathcal{M}_{\mathcal{B}}(g_r) \neq \mathcal{M}_{\mathcal{B}}(g_t) \forall r \neq t\}} [\eta(g_k)]}{\binom{n_{\mathcal{B}}+l_{\mathcal{B}}-1}{l_{\mathcal{B}}}}$$

Since the sets of all neutral classes are a partition of the representation space, the summation of all of the neutral class sizes is the size of the space:

$$\sum_{\{g_r \mid \mathcal{M}_{\mathcal{B}}(g_r) \neq \mathcal{M}_{\mathcal{B}}(g_t) \forall r \neq t\}} [\eta(g_k)] = |\mathcal{G}_{\langle l_{\mathcal{B}}, \Sigma_{\mathcal{B}} \rangle}|$$

Thus, $\hat{\eta}$, the average size of all neutral classes over a representation space, is given by the following equation as function of the parameters defining the space itself:

$$\hat{\eta} = \frac{(n_{\mathcal{B}})^{l_{\mathcal{B}}}}{\binom{n_{\mathcal{B}}+l_{\mathcal{B}}-1}{l_{\mathcal{B}}}} \quad (6.15)$$

At this point, we can replace in our previous equations the specific size of a neutral class, η , with the average size over all possible neutral classes in a representation space, $\hat{\eta}$, to yield results that are independent of p_s .

Replacing η with $\hat{\eta}$ on Equation (6.12) and rearranging terms we obtain the condition for the PGA Hypothesis to hold in the average case:

$$(n_{\mathcal{A}})^{l_{\mathcal{A}}} \geq \binom{n_{\mathcal{B}} + l_{\mathcal{B}} - 1}{l_{\mathcal{B}}} \quad (6.16)$$

Equation (6.16) is independent of the choice of p_s (solution point) because η has been averaged over all possible choices of p_s . Note from this equation that the PGA hypothesis will hold if and only if:

$$|\mathcal{P}_{\mathcal{B}}| \leq |\mathcal{P}_{\mathcal{A}}| \quad (6.17)$$

Equation (6.17) tells us that, in order for a PGA to have an advantage, the PGA solution space cannot be larger than the GA counterpart. But equation (6.3) states that both solution spaces should be equal for a fair comparison. Hence, the probability that a PGA will find a solution is, on average, equal to the probability that a GA will find a solution when both solution spaces are equal in size. In order for a PGA to have a greater probability of finding a solution, the PGA must be searching in a solution space that is smaller than that of the competing GA.

Equation (6.13) considers the case of $|\mathcal{G}_{(l_{\mathcal{A}}, \Sigma_{\mathcal{A}})}| \equiv |\mathcal{G}_{(l_{\mathcal{B}}, \Sigma_{\mathcal{B}})}|$. Using the average value of η , $\hat{\eta}$, we obtain:

$$\hat{\eta} \geq 1$$

Replacing $\hat{\eta}$ with its value from Equation (6.15), we get:

$$\frac{(n_{\mathcal{B}})^{l_{\mathcal{B}}}}{\binom{n_{\mathcal{B}} + l_{\mathcal{B}} - 1}{l_{\mathcal{B}}}} \geq 1$$

This equation holds only for the inequality. Thus, the PGA hypothesis holds as an inequality. As a result, when a PGA and GA have representation spaces of same size, a PGA will have, on average, a better chance of finding a solution than a GA.

Equation (6.14) examines the case where $|\mathcal{P}_{\mathcal{A}}| \equiv |\mathcal{P}_{\mathcal{B}}|$. Using $\hat{\eta}$ we can now obtain:

$$\hat{\eta} \geq \frac{(n_{\mathcal{B}})^{l_{\mathcal{B}}}}{\binom{n_{\mathcal{B}} + l_{\mathcal{B}} - 1}{l_{\mathcal{B}}}}$$

$$\frac{(n_{\mathcal{B}})^{l_{\mathcal{B}}}}{\binom{n_{\mathcal{B}} + l_{\mathcal{B}} - 1}{l_{\mathcal{B}}}} \geq \frac{(n_{\mathcal{B}})^{l_{\mathcal{B}}}}{\binom{n_{\mathcal{B}} + l_{\mathcal{B}} - 1}{l_{\mathcal{B}}}}$$

In this case, the equation holds only for the equality. Thus, the PGA hypothesis holds as equality. With solution spaces of the same size, a PGA has, on average, similar chances of finding a solution as a GA.

6.3 Discussion

This chapter provides a resolution analysis and a random sampling analysis of PGA proteome-based location independent representation and the GA binary representation. Using combinatorics, we provided closed form expression for the relationship between individual lengths and alphabet sizes that will result in comparable solution spaces. This analysis indicates that in order to encode the same amount of information, proteome-based location independent representations require significantly larger alphabets of symbols than traditional representations. This chapter also provides a closed form expression for the sizes of the neutral classes and observes that the redundancy introduced over-represents individuals with high symbolic diversity and under-represents individuals with low symbolic diversity.

Using these two expressions it calculates the probability of finding a solution in either representation space by random sampling. This analysis indicates that proteome-based location independent representations and binary representations have, on average, similar chances of finding a solution. It also shows that for a particular solution, one or the other representation may have the advantage depending of the size of the neutral class associated with the chosen target solution.

CHAPTER 7

EMPIRICAL ANALYSIS

This chapter experimentally compares the three variations of the PGA described in section 5.1 with a canonical GA. Specific questions addressed in the experiments include:

1. How does a PGA compare to a traditional GA? Can a PGA perform at least as well as a GA?
2. Does a PGA use non-coding regions as a means of “fine-tuning” the values it encodes?
3. Can a PGA regulate the length of its individuals to minimize computational effort while maximizing performance?
4. Does a PGA encourage the formation of building blocks?

7.1 Experimental Setting

7.1.1 Test Problems

We study three types of problems in the experiments described here.

1. Resource allocation
2. Number matching
3. Symbolic regression

All of these problems require a GA to find five values.

The resource allocation problem is a search for a series of values that sum to a predefined value. This type of problem appears to be a perfect fit for the PGA encoding. We compare the behavior of PGA1 with a GA on this problem.

The second two problems search for a series of numbers that do not have to sum to a predefined value. Many real world problems can be mapped to one of these general problem types. In number matching, all values have equal weight. In symbolic regression, some values may have greater affect on fitness. We compare both PGA2 and PGA3 with a GA on these two problems.

Table 7.1: GA parameter settings.

| | | |
|-------------------------------|---|---|
| Population size | : | 200 |
| Maximum number of generations | : | 500 |
| Selection method | : | tournament |
| Crossover type | : | two-point (fixed len), homologous (variable len) |
| Crossover rate | : | 1.0 |
| Mutation rate | : | 0.01, 0.005 |

7.1.2 GA Configuration

Table 7.1 gives the parameter setting used in these experiments. Each experiment was run 100 times and the results averaged over all runs.

Our GA uses a binary representation consisting of a field of eight bits for each of the five values for a total length of 40 bits. We can use Equation (6.5) from section 6.1.4 to calculate the appropriate PGA lengths for our comparisons. For each of our five values, we have $n_{\mathcal{A}} = 2$ and $l_{\mathcal{A}} = 8$, as our GA is binary and eight bits are allocated per value. For the PGA we have $n_{\mathcal{B}} = 2$ as each parameter is represented with an alphabet size of two¹. Entering these values into Equation (6.5) and solving for $l_{\mathcal{B}}$, we obtain $l_{\mathcal{B}} = 255$. Therefore, 255 is the equivalent length that a PGA would require to encode a single parameter value with the same resolution as the GA. As a result, the PGA representation would need a genome length of $255 \times 5 = 1275$ to encode the same resolution as the GA.

In addition to the simple PGA representation, we also test extensions which include non-coding regions and variable length individuals. We test a total of six variations of each PGA:

PGA-40: Simple PGA with fixed length individuals. The length of the individuals is the same as the length of the individuals in the GA (40 bits).

PGA-255: Simple PGA with fixed length of 255 bits. As the full resolution of the PGA requires a genome length of 1275, PGA-255 is still somewhat penalized with respect to resolution; however, this PGA at least has the resolution of a single GA field. The location independence of the PGA allow us to overlap all five values on the same genome.

PGA-40-nc: PGA with fixed length individuals and non-coding regions. Length is same as in **PGA-40**. One or more non-coding characters are added to the PGA alphabet.

¹This is clear for PGA2 and PGA3 since the alphabets are separate; however for PGA1 we also have two groups of characters for each value: one distinguished character and one group undistinguished characters—the rest of the alphabet.

PGA-255-nc: Same as **PGA-40-nc** with length of 255.

PGA-var: PGA with variable lengthed individuals. The maximum allowed length is set at 2048 to ensure that the PGA would have as much resolution as it would need. Note that this value is bigger than the required length of 1275. No parsimony pressure is applied unless otherwise specified.

PGA-var-nc: PGA with variable lengthed individuals and non-coding regions.

7.2 Formation of Building Blocks and Regulation of Length and Resolution Analysis

7.2.1 Resource Allocation Test Problem

The resource allocation problem involves the allocation of a fixed pool of resources among five users.

The PGA1 encoding is described in section 5.1.1. A GA individual is decoded by first calculating the binary values encoded in each field. Each value is then divided by the sum of all of the values to produce a proportion between 0.0 and 1.0.

Given $a = 5$ target values or proportions, $T_i, i = 0, \dots, a - 1$ where $\sum T_i = 1.0$, and a users $U_i, i = 0, \dots, a - 1$, the encoded proportion assigned to each user is given by $P_{PGA1}(U_i)$ (see section 5.1.1). We first calculate the *ratio* of each corresponding pair:

$$ratio(i) = \begin{cases} \frac{T_i}{P_{PGA1}(U_i)} & \text{if } T_i < P_{PGA1}(U_i) \\ \frac{P_{PGA1}(U_i)}{T_i} & \text{otherwise} \end{cases}$$

This value gives an indication of how close each encoded value is to the corresponding target value. The fitness of an individual is the average of all ratios:

$$fitness = \frac{\sum_{i=0}^{a-1} ratio(i)}{a}.$$

A perfect match gives the maximum score of 1.0.

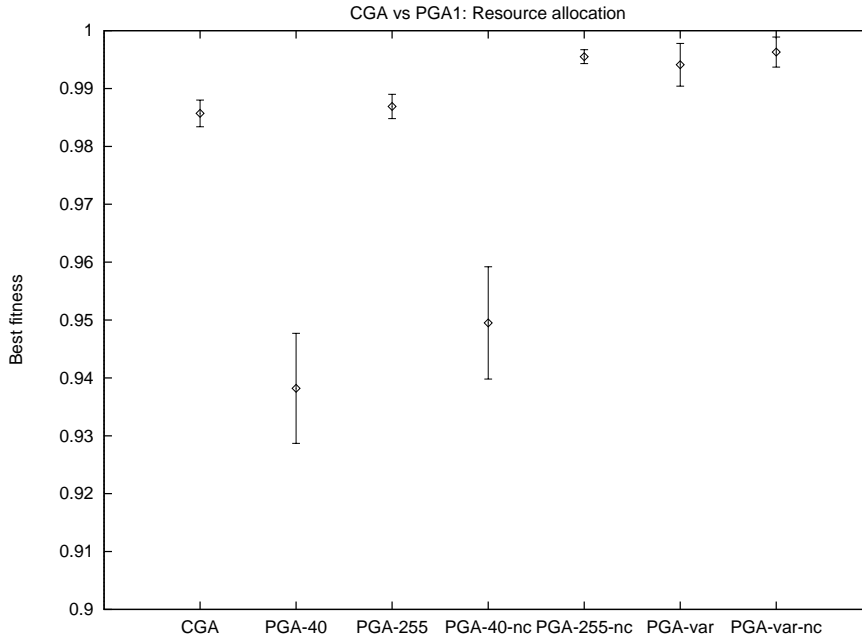


Figure 7.1: GA -vs- PGA1 on resource allocation: Fitness of best solution found averaged over 100 run and 95% confidence intervals.

7.2.2 Randomly Generated Target Allocations

Our first experiment simply compares the ability of a GA and PGA1 to find a set of target proportions. To eliminate any unexpected bias, we randomly generate a set of target values, T_i where $\sum T_i = 1.0$, at the beginning of each run. As a result, each of the 100 runs that make up this experiment are searching for a different set of target values. Figure 7.1 shows the average best solutions obtained from each algorithm tested with 95% confidence intervals. The GA performs well, with an average best fitness of 0.9857. All of the PGA variations except for the two using length 40 perform as well or significantly better than the GA. PGA-255 performance is consistent with GA performance despite its resolution penalty. Enhancing the PGA with either non-coding regions or variable length appears to significantly improve performance. Although the PGAs with length 40 performed significantly worse, they still provided respectable average fitnesses of 0.9382 and 0.9495. Table 7.2 shows detailed results from this experiment. In addition to the best fitness value, we also track the number of runs out of 100 in which a best fitness of 0.99 or higher was found — the number of “hits”. The GA finds fitness values above 0.99 in 53 out of 100 runs. Except for the PGAs of length 40, all other PGA runs are better able to achieve 0.99 fitness than the GA. The variable length PGAs evolve significantly longer individuals than the fixed length values tested. These lengths, however, are not unreasonable given the actual required resolution of 1275. Applying parsimony pressure, as described in [BDG98], results in average lengths that are about half of those shown above with essentially the same average best fitness value.

Table 7.2: Results of PGA1 versus GA on finding randomly generated allocation values. Average (and standard deviation) over 100 runs.

| Algorithm | Best fitness | Hits ≥ 0.99 | Genome length | Time in sec. |
|------------|-----------------|------------------|------------------|---------------|
| GA | 0.9857 (0.0120) | 53 | 40 | 13.96 (1.24) |
| PGA-40 | 0.9382 (0.0483) | 2 | 40 | 12.08 (0.65) |
| PGA-255 | 0.9869 (0.0110) | 59 | 255 | 15.75 (4.38) |
| PGA-40-nc | 0.9495 (0.0496) | 1 | 40 | 12.48 (1.02) |
| PGA-255-nc | 0.9955 (0.0063) | 94 | 255 | 16.12 (3.66) |
| PGA-var | 0.9941 (0.0189) | 91 | 1813.56 (585.68) | 46.86 (16.77) |
| PGA-var-nc | 0.9963 (0.0135) | 95 | 1810.30 (606.37) | 47.53 (13.72) |

All of our runs ran for 500 generations. The average clock time (in seconds) increases only slightly when genome length is increased from 40 to 255. Variable length PGAs, however, require significantly longer time to complete 500 generations of evolution.

7.2.3 Formation of Building Blocks

One of the expected advantages of the PGA encoding is that its complete location independence would encourage (or, at the very least, not impede) the formation of building blocks of related characters. Hypothetically, building blocks may be formed in two ways in the PGA. First, related characters (all of a particular character) are arranged in close proximity to minimize the chance of disruption by crossover. Second, all copies of each character are spread evenly across the entire individual making any particular segment of an individual essentially a building block (a miniature version of the full individual, albeit with coarser resolution). We examine individual PGA runs in detail to determine if such building block formation does occur.

We calculate the center of gravity and range for every character on every individual. The *center of gravity* (CoG) of a character is the average of the locations of all instances of that character on an individual. The *range* of a character is delimited by the two end-most copies of the character. We normalize the positions on an individual to range from 0.0 to 1.0 where 0.0 refers to one end of the individual and 1.0 refers to the opposite end. CoG values will then fall in between 0.0 and 1.0. Figures 7.2 and 7.3 show data from generation 500 of a PGA run.

The x-axis of these plots indicate each individual from the generation. The y-axis indicates position on an individual with normalized position values (positions range from 0.0 to 1.0). The Figure 7.2 gives the CoGs of all five characters in this PGA. The Figure 7.3 shows the CoG and range of one specific character. By generation 500, all CoG values have evolved

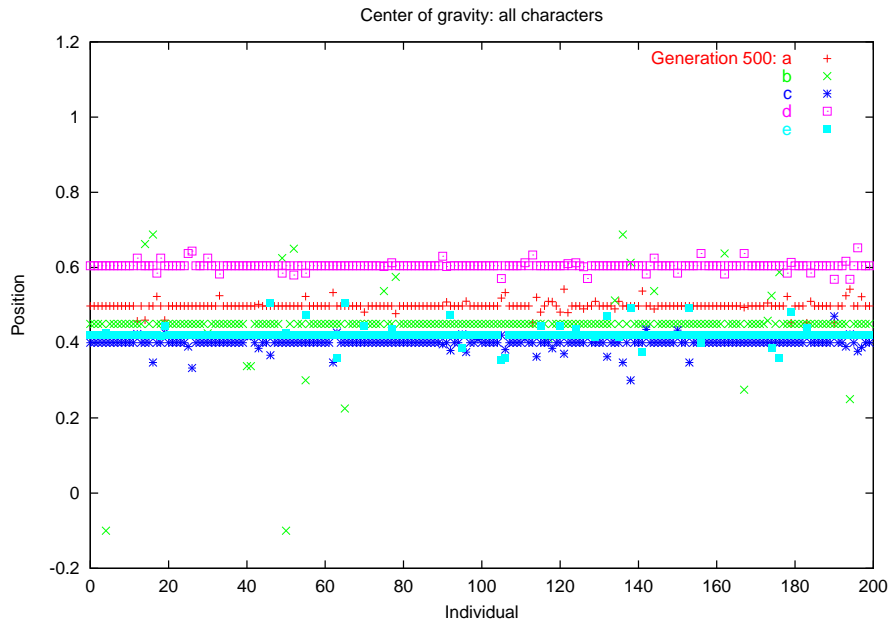


Figure 7.2: Center of gravity of characters on a population of PGA individuals.

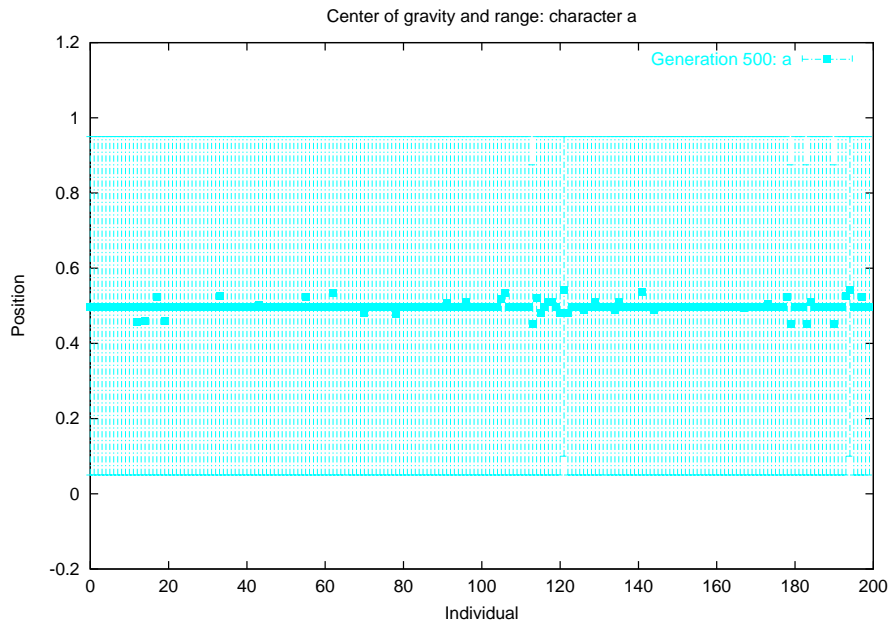


Figure 7.3: Range of characters on a population of PGA individuals.

Table 7.3: Results of PGA1 versus GA on finding fixed resolution allocation values. Average (and standard deviation) over 100 runs. A hit is a run that achieves the fitness of 0.99 or higher. All fitnesses are 0.95 or higher.

| Algorithm | Best fitness | Hits ≥ 0.99 | Genome length | Time in sec. |
|------------|-----------------|------------------|------------------|---------------|
| GA | 0.9911 (0.0009) | 80 | 40 | 5.23 (0.45) |
| PGA-40 | 1.0 (0.0) | 100 | 40 | 3.34 (0.48) |
| PGA-255 | 0.9922 (0.0) | 100 | 255 | 8.27 (0.45) |
| PGA-40-nc | 1.0 (0.0) | 100 | 40 | 3.76 (0.43) |
| PGA-255-nc | 1.0 (0.0) | 100 | 255 | 8.36 (0.48) |
| PGA-var | 0.9997 (0.0003) | 100 | 1460.33 (849.38) | 53.62 (8.87) |
| PGA-var-nc | 1.0 (0.0) | 100 | 1129.86 (933.11) | 50.63 (14.96) |

close to the center of the individuals. The range of the characters appears to be spread out over the entire individual. These results indicate that a PGA tends to form the second type of building block where characters are distributed across an entire individual. Examination of specific individuals supports this conclusion. Figure 7.4 shows an example individual from a PGA run. Similar, repeated sequences are aligned for easier recognition. A high degree of repetition and many similar building blocks are found throughout the individual. A shift in the alignment used in figure 7.4 by one character produces an entirely new set of building blocks containing approximately the same set of characters.

7.2.4 Regulation of Length and Resolution

To further investigate the PGA's ability to regulate length and resolution, we next perform an experiment with specific target values where the target allocations have the ratio 1:2:4:2:1. The goals of this experiment are to test whether a PGA allocates an appropriate number of characters to account for the desired resolutions and to test whether a PGA will vary or minimize length in response to the resolution of the target values. The minimum number of characters required to encode this distribution in a PGA is 10 characters. As a result, the 40 length used in our fixed experiments should be able to find a perfect solution.

Table 7.3 shows the results from this experiment. The GA is only able to find solutions above 0.99 fitness in 80% of its runs while all PGA runs were able to find solutions above 0.99 fitness. PGA-40 easily found the optimum distribution in all runs. Because a length that is a multiple of ten is required to achieve the optimum distribution, PGA-255 was not able to achieve a fitness of 1.0; A standard deviation of zero and examination of individual solutions, however, indicate that PGA-255 found the best possible solutions for its given

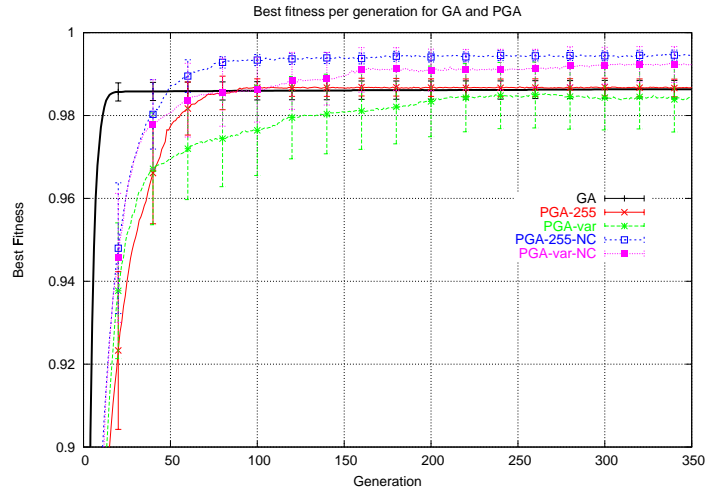


Figure 7.5: Best fitness per generation obtained by the GA, PGA-255, PGA-var, PGA-255-nc, and PGA-var-nc for the resource allocation problem using randomly generated allocation values.

length. Adding non-coding regions to PGA-255 alleviates this problem: PGA-255-nc finds all optimum solutions. Variable length PGAs, again, evolve significantly longer individuals but application of parsimony pressure reduces average lengths to around 240. Despite the large average length, two PGA-var runs generate solutions of length 10.

These results indicate that, given a problem whose resolution is achievable with the length used by a PGA, that PGA will be able to find a perfect solution and will outperform a GA. Furthermore, a PGA that has the ability to adjust its resolution, e.g. by varying length or amount of non-coding regions, will take advantage of these mechanisms to improve its solutions. Parsimony pressure can be used to minimize solution length with very little fitness penalty.

7.2.5 Generational Behavior of Best Fitness

Previous sections compare GA and PGA best fitness obtained over 500 generations. In this section we present the generational behavior of GA and PGA best fitness for the resource allocation domain. All results are averaged over 100 independent runs. Figure 7.5 shows the best fitness per generation obtained by GA, PGA-255, PGA-var, PGA-255-nc, and PGA-var-nc for the resource allocation problem with randomly generated allocations. We clearly observe that all PGA variants take longer time to converge. The GA converges at generation 25, while even by generation 350 the PGA variants do not appear reach convergence. At

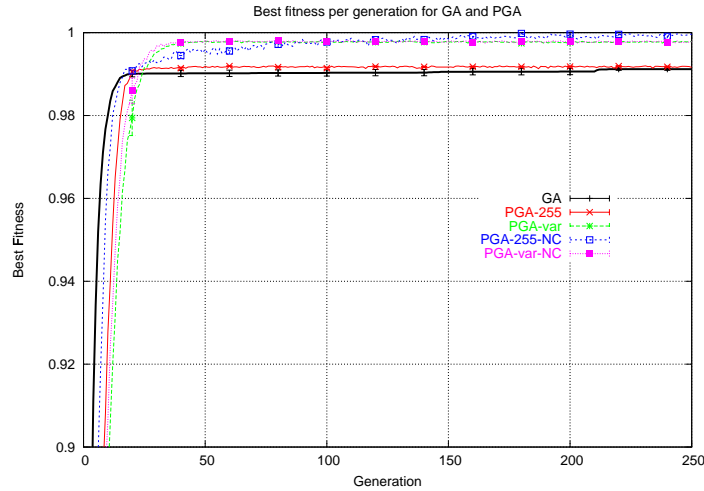


Figure 7.6: Best fitness per generation obtained by the GA, PGA-255, PGA-var, PGA-255-nc, and PGA-var-nc for the resource allocation problem using a fixed test allocation(1:2:4:2:1).

generation 25, all PGA variants have worst best fitness than the GA; however, 75 generations later, three PGA variants have already achieved better best fitness than the GA. The PGA-var never obtains better fitness than the GA. Figure 7.6 is analogous to the previous figure, but shows results for the resource allocation problem with fixed allocation values. In this case, the GA also has a faster initial best fitness improvement than the PGA. At generation 14, all PGA variants have worst best fitness than the GA, but less than a dozen generations later, all PGA variants have already achieved better best fitness than the GA. Appendix B gives a complete collection of generational best fitness behavior data for the GA and the PGA over the various problems discussed on this thesis.

From this experimental data we conclude that the PGA improves best fitness in a slower and smoother fashion than the GA. Nevertheless, while the GA seems to converge quickly to a fitness plateau, the PGA continues to improve best fitness and in many cases surpasses the fitness plateau of the GA.

7.2.6 Sensitivity Analysis

In the previous sections, we offer results using various PGA resolutions such as PGA-40, PGA-255, PGA-var, etc. In this section, we analyze the sensitivity of the PGA to various crossover and mutation rates. We set up 15 experiments with different crossover and muta-

Table 7.4: Crossover and mutation rates for the 15 experiments of the sensitivity analysis

| Experiment | Parameters | |
|------------|----------------|---------------|
| | Crossover rate | Mutation rate |
| e1 | 0.00 | 0.002 |
| e2 | 0.00 | 0.005 |
| e3 | 0.00 | 0.010 |
| e4 | 0.50 | 0.000 |
| e5 | 0.50 | 0.002 |
| e6 | 0.50 | 0.005 |
| e7 | 0.50 | 0.010 |
| e8 | 0.75 | 0.000 |
| e9 | 0.75 | 0.002 |
| e10 | 0.75 | 0.005 |
| e11 | 0.75 | 0.010 |
| e12 | 1.00 | 0.000 |
| e13 | 1.00 | 0.002 |
| e14 | 1.00 | 0.005 |
| e15 | 1.00 | 0.010 |

tions rates as shown in table 7.4. In each experiment, a GA and PGA1 are set to solve the resource allocation problem. For all experiments we use PGA-var ².

Figure 7.7 shows the fitness of the best individual over 20 runs for PGA1 and GA over the 15 experimental settings shown above. We observe that PGA manages to find the best fit individuals for all 15 settings. Figure 7.8 shows the average fitness with the 95% confidence intervals over the 20 runs. In this case, we observe that the GA average performance over various parameter settings is quite stable. On the other hand, the PGA average performance appears to depend heavily on the value of the mutation rate. PGA performs poorly on experiments e4, e8, and e12, where the mutation rate is set to zero.

The suitability of the PGA to resource allocation problems is clearly demonstrated by the above studies. The PGA is competitive and often outperforms the GA on such problems. Most problems, however, cannot be encoded or interpreted as a resource allocation problem. Thus for the PGA to be useful, we must be able to apply it to other types of problems. In the next section we investigate PGA performance on two problems onto which many real world problems may be mapped.

²for this experiments PGA-var has a fixed length of 1275, which is the exact comparable length as described in section 7.1.2

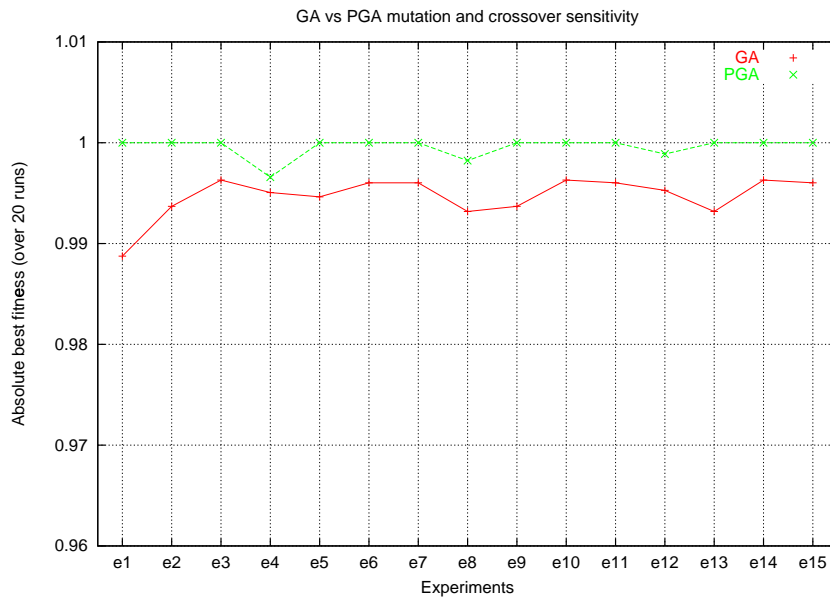


Figure 7.7: GA -vs- PGA1 on resource allocation: absolute best fitness found over 20 runs for experiments e1 to e15. Crossover and mutation rates used on each experiment are shown in table 7.4

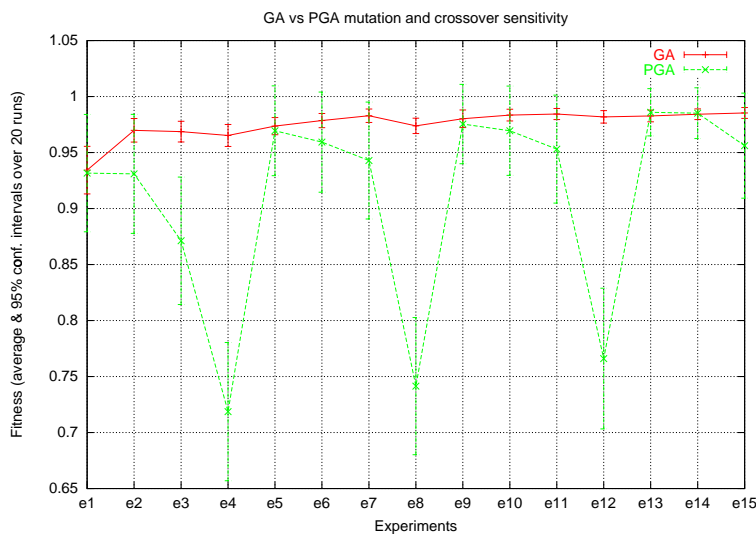


Figure 7.8: GA -vs- PGA1 on resource allocation: fitness of best solution found averaged over 20 runs and 95% confidence intervals for experiments e1 to e15. Crossover and mutation rates used on each experiment are shown in table 7.4

7.3 Performance Analysis

There are many problems which may be described as a search for a vector of values. These values may be completely independent or may have dependencies among themselves. We now examine PGA performance on two representative problems. The first problem is a simple number matching problem in which we search for a set of five independent numbers. The second problem is a symbolic regression problem in which we search for a set interdependent numbers. We test two PGA encodings called PGA2 and PGA3 (described in section 5.1.2 and 5.1.3) on both of these problems.

7.3.1 Number Match Test Problem

The number match problem is a search for $a = 5$ independent values. The a independent target values are randomly generated at the beginning of each run so that each run searches for a different set of values.

The fitness function for this problem is the same as the fitness function for the resource allocation problem. Given a target values, T_i , and a values decoded from a PGA individual, $P_{PGA}(V_i), i = 0, \dots, a - 1$, we first calculate the ratio of the smaller value divided by the larger value.

$$ratio(i) = \begin{cases} \frac{T_i}{P_{PGA}(V_i)} & \text{if } T_i < P_{PGA}(V_i) \\ \frac{P_{PGA}(V_i)}{T_i} & \text{otherwise} \end{cases}$$

The fitness of an individual is the average of all ratios:

$$fitness = \frac{\sum_{i=0}^{a-1} ratio(i)}{a}.$$

Again, a perfect match gives the maximum score of 1.0.

7.3.2 Symbolic Regression Test Problem

Given a set of p data points, $d_i, i = 0, \dots, p - 1$, the symbolic regression problem is a search for $a = 5$ coefficients that, when plugged into the equation

$$f(x) = \mathbf{a}x^2 + \mathbf{b}x + \mathbf{c} + \mathbf{d} \cos(x) + \mathbf{e} \sin(x),$$

most closely approximate the target equation. Instead of trying to match the encoded coefficient values to target values, the fitness function is calculated from the difference between

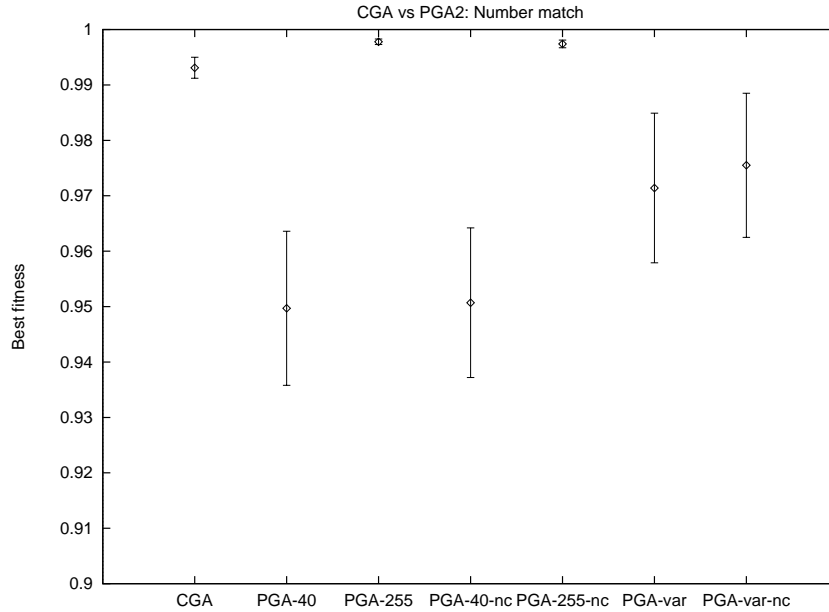


Figure 7.9: GA -vs- PGA2 on number match: Fitness of best solution found averaged over 100 runs and 95% confidence intervals.

the target data points and the function values generated using the encoded coefficient values. The $ratio(i)$ at each data point is

$$ratio(i) = \begin{cases} \frac{d_i}{f(x)} & \text{if } d_i < f(x) \\ \frac{f(x)}{d_i} & \text{otherwise} \end{cases}$$

The fitness is the average of all ratios:

$$fitness = \frac{\sum_{i=0}^{a-1} ratio(i)}{a}.$$

A perfect match gives a maximum score of 1.0. This problem differs from the number match problem in that the a values are interdependent. Changes in a single value can affect the impact of other values.

7.3.3 Results

Figure 7.9 and 7.10 compare the average best fitness and number of hits over 100 runs of a GA and PGA2 variations on the number match problem. In both comparisons, PGA2 significantly outperforms the GA only when it has length 255. A length of 40 results in

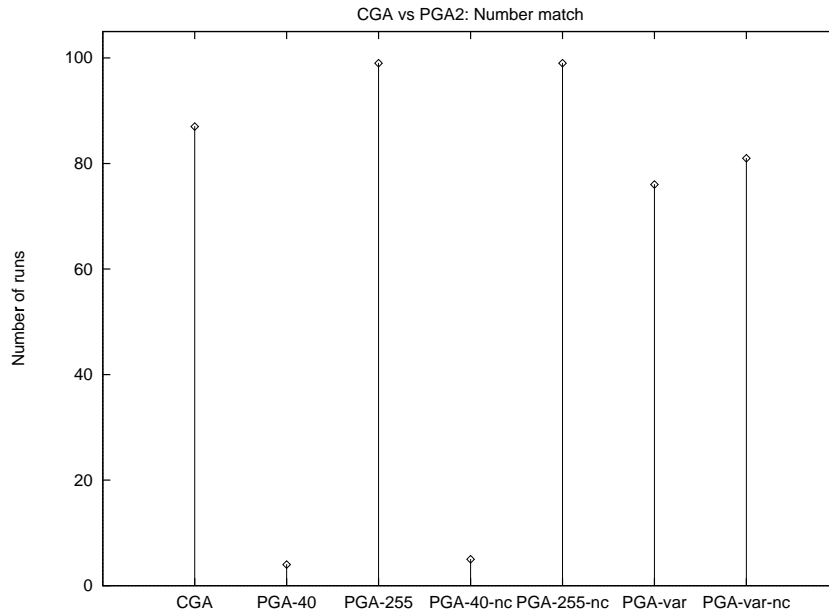


Figure 7.10: GA -vs- PGA2 on number match: Number of runs out of 100 that find solutions with fitness 0.99 or higher.

a significant performance decrease; however, fitness values still reach well above 90%. The addition of non-coding regions has little effect. A variable length PGA2 performs somewhat better than PGA-40, but remains worse than the GA. The fact that variable length is less beneficial with PGA2 than PGA1 is likely due to the more complex mapping from character proportions to function values in PGA2.

Figures 7.11 and 7.12 compare the average best fitness and number of hits over 100 runs of a GA and PGA3 variations on the number match problem. Again, the best PGA performance occurs when PGA3 has length 255. Those PGA3 runs reach equivalent fitness values as the GA and slightly outperform the GA in hit count. PGA3 with length 40 performs significantly worse as do the variable length PGA3s.

Figures 7.13 to 7.16 show the corresponding data for the symbolic regression problem. The relative performances of the algorithms and the GA performances are similar on both the symbolic regression and number match problems. All of the PGAs, however, perform significantly better on the symbolic regression problem. (Note the narrower y-axis range.) In particular, the PGAs with length 40 show a marked improvement in the number of hits. PGA-255 with and without non-coding regions is able to achieve a 100% hit rate in all but one experiment (which reaches 98).

In both the number match and symbolic regression problems, PGA2 performs slightly better than PGA3. Non-coding regions appear to have very little effect. Variable length individuals do give the PGA more flexibility, but appears to add too much complexity to the search space. Simply giving the PGA a fixed but reasonable amount of resource with which

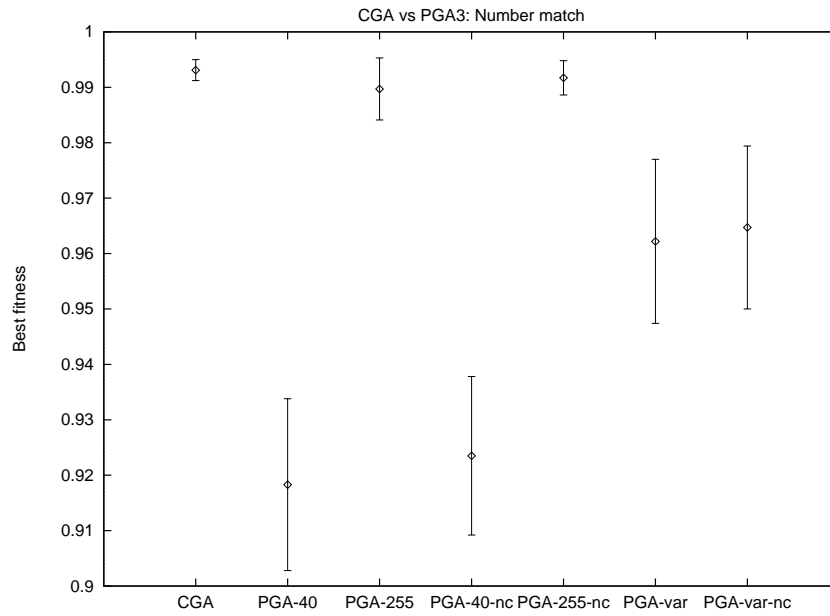


Figure 7.11: GA -vs- PGA3 on number match: Fitness of best solution found averaged over 100 runs and 95% confidence intervals.

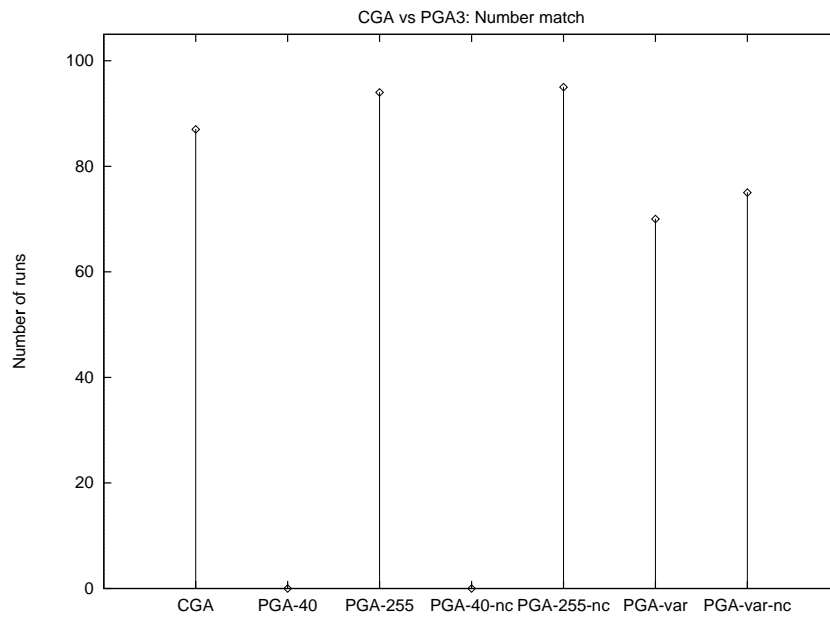


Figure 7.12: GA -vs- PGA3 on number match: Number of runs out of 100 that find solutions with fitness 0.99 or higher.

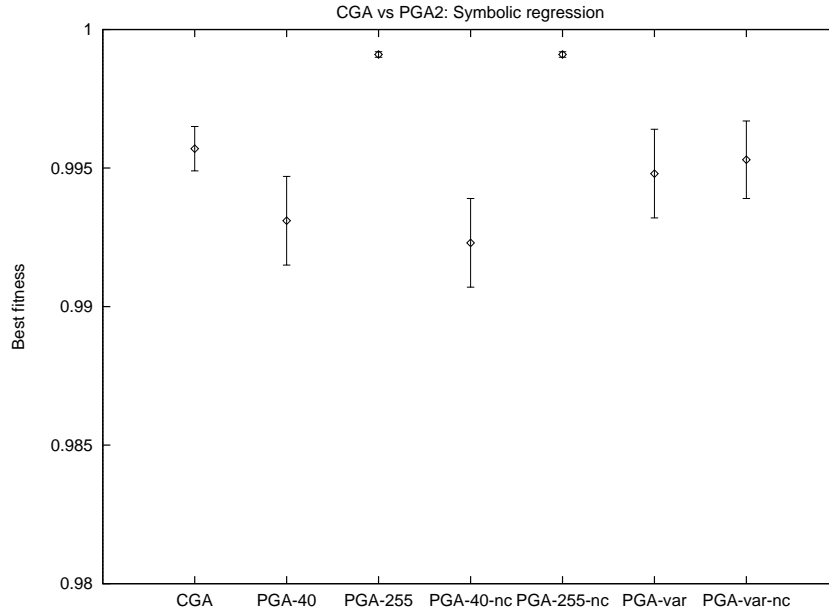


Figure 7.13: GA -vs- PGA2 on symbolic regression: Fitness of best solution found averaged over 100 runs and 95% confidence intervals.

to work seems to be the best, and relatively simple, solution. It is important to note that, according to the calculations in section 6.1.4, comparing a GA of length 40 with a PGA of length 255 still significantly penalizes the PGA. Presumably, increasing the length, and thereby the resolution, of the PGA would result in even better performance.

7.4 Discussion

This chapter introduces a GA with a new representation method which we call the proportional GA. The PGA is a multi-character GA that relies on the existence of genes rather than the order of genes to encode information. The inspiration for the PGA comes from the biological concept of gene expression: existing genes on a genome are expressed when they produce protein products; the combination of expressed protein products interact to produce life. Similarly, existing characters on a PGA individual are expressed and interact with the other expressed characters from that individual to produce a candidate solution. Information is represented in terms of the proportions rather than the ordering of the characters on an individual. Statistically, a fairly matched GA and PGA should have, on average, equal probabilities of finding a solution. Experimentally, the PGA appears to be able to generate comparable behavior even with lowered resolution of expression.

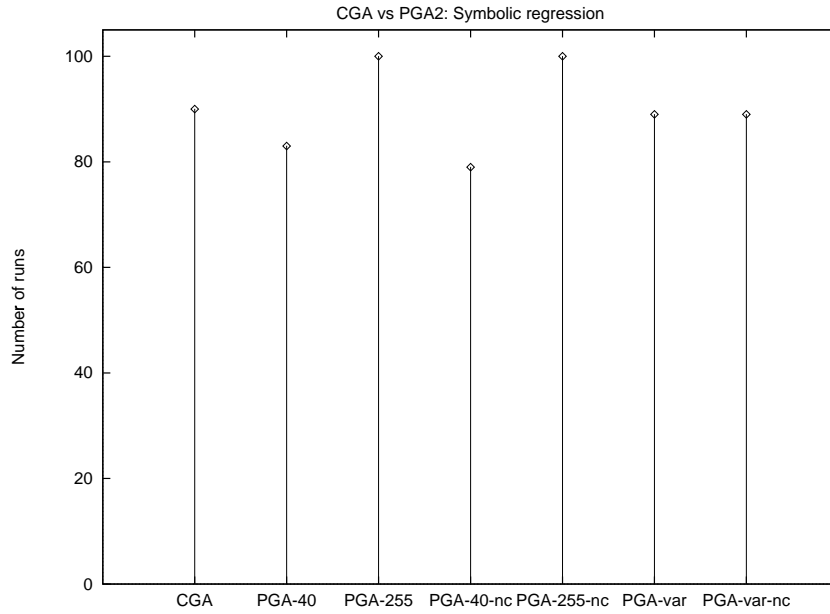


Figure 7.14: GA -vs- PGA2 on symbolic regression: Number of runs out of 100 that find solutions with fitness 0.99 or higher.

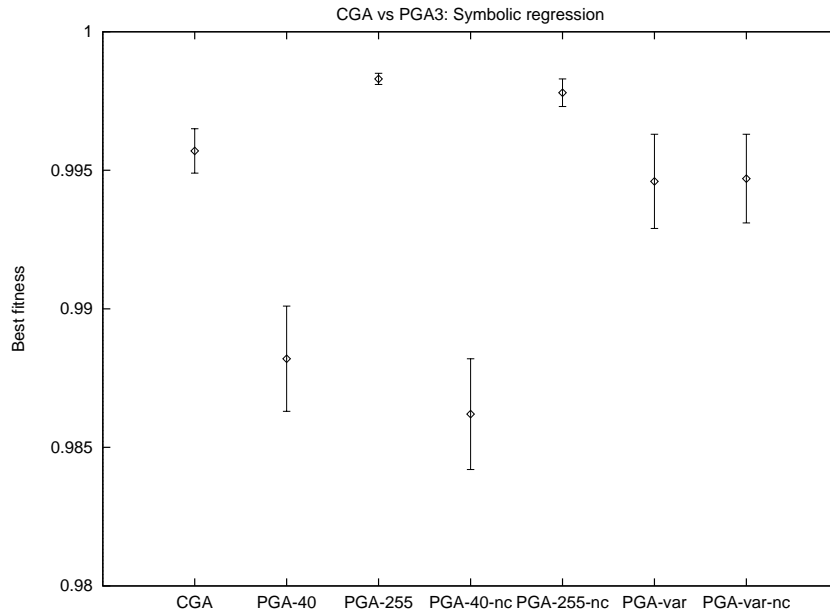


Figure 7.15: GA -vs- PGA3 on symbolic regression: Fitness of best solution found averaged over 100 runs and 95% confidence intervals.

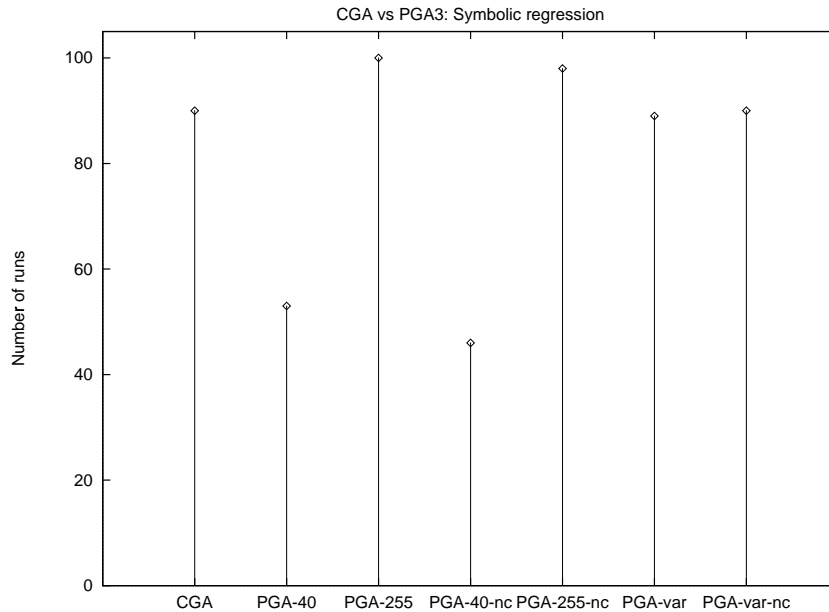


Figure 7.16: GA -vs- PGA3 on symbolic regression: Number of runs out of 100 that find solutions with fitness 0.99 or higher.

In section 7.1 we give a list of questions on which this chapter is focused. We now discuss our conclusions with respect to those questions.

1. How does a PGA compare to a traditional GA? Can a PGA perform at least as well as a GA?

Our initial studies indicate that the PGA can perform as well or better than a GA. The determining factor for PGA performance appears to be its resolution. Given a reasonable resolution, our PGA performances were comparable or better than the GA. “Reasonable”, in this case, can still be significantly less than GA resolution (in our experiments, five times less).

The PGA appears to be particularly well suited for resource allocation problems. A PGA that has enough resolution to encode the target values will almost always find an optimal solution. A PGA that does not have enough resolution, finds the best possible solution for its given resolution. Given the opportunity, a PGA will adjust the length of its evolved individuals to more closely match the required resolution of a solution. The similarities between the PGA problem representation and the resource allocation problem itself is thought to be a reason for the excellent PGA performance.

On the number match and symbolic regression problems, where the target values do not sum to 1.0, the PGA performs comparably or better than the GA if it is given enough resolution. The ability to vary individual length is less useful in these problems, presumably due to the more complex mapping between genotype and phenotype.

Higher resolution appears to be the best method for improving PGA performance. Interestingly, while GA performance is similar for both problems, the PGA performs significantly better on the symbolic regression problem. The explanation for this difference is unclear at this point, but we note that both the resource allocation and symbolic regression problems encode values that are dependent on each other.

2. Does a PGA use non-coding regions as a means of “fine-tuning” the values it encodes?

The impact of non-coding regions is unclear in these experiments. In general, the addition of non-coding regions did not produce any noticeable difference in PGA behavior. The only instance in which non-coding regions improved performance significantly is in the resource allocation problem when a PGA length that does not match perfectly to form the target resolutions. In that instance, the PGA appears to use non-coding regions as “padding” to achieve better target values.

3. Can a PGA regulate the length of its individuals to minimize computational effort while maximizing performance?

Given the opportunity, a PGA will attempt to evolve the length of its individuals to accommodate the required resolutions. Parsimony pressure can be used to minimize length with, apparently, very little impact on fitness. The PGA successfully uses variable lengths to solve the resource allocation problem. With the other problems, however, variable length appears to be less useful than simply providing sufficient resolution. As compared with insufficient resolution, however, variable length PGAs achieve significantly better fitness values. Overall, the addition of variable length individuals appears to be more beneficial than the addition of non-coding regions.

4. Does a PGA encourage the formation of building blocks?

The PGA forms building blocks by distributing copies of each character across an entire individual. In a sense, any segment of an individual is a “building block” representing a coarser version of the solution represented by the entire individual. As a result, the solution represented by a single individual is extremely robust in response to changes in the length as well as to crossover. Close examination of individual genomes finds many repeated regions. The PGA does not form building blocks that consist of tightly linked regions of the same character.

Our preliminary studies on the PGA have yielded very promising results and we plan to continue development and analyses of the PGA representation. Despite the lowered resolution of the PGA representation, PGA performance appears to be competitive with GA performance and PGA runs appear to be taking advantage of the flexibility provided by variable length individuals. Although well-suited to resource allocation problems, the PGA representation can be modified to work successfully on other types of problems.

CHAPTER 8

EMERGENCE OF GENOMIC SELF-ORGANIZATION

In this chapter, we use the Proportional Genetic Algorithm (PGA), as described in chapter 5, to study the emergent ordering of genomic symbols in the complete absence of selective pressure for a particular order. We hypothesize that self-similarity emerges because self-similar genomes are more robust with respect to crossover and mutation and because it favors positive correlations between form and quality of candidate solutions. The PGA is a Genetic Algorithm (GA) [Hol75a, Gol89] with a representation based on protein concentrations rather than on the usual gene ordering. A PGA translates strings of genes into multisets of proteins prior to fitness evaluation. As a result, there is no fitness pressure for any particular gene ordering and the order of the genes is free to evolve along with the candidate solutions that they encode. Section 7.2.3 shows that genomic symbols under these circumstances are evenly distributed throughout the genome (see also [WG02]) and that they appear to form building blocks of a peculiar type: coarse grained versions of the entire genome. In this chapter, we ask the fundamental question: *what is the emergent genomic ordering when there is no selective pressure for any particular ordering?* We use two very different methods to analyse the emergent genomic structure: equal-symbol correlation analysis, originally proposed by Voss [Vos93] to analyse DNA structure, and an experimental method of our making to analyse the self-similarity of genomic segments with respect to fitness. Our results can be summarized as follows:

1. The equal-symbol correlation on PGA genomes resembles white noise behavior [GW04].
2. The emergent genomic structure is self-similar with respect to fitness.

8.1 Emergent White Noise Behavior

It has been shown using standard spectral density measurement techniques that individual base positions in DNA sequences exhibit $1/f$ noise and long-range fractal correlations [Vos93]. From the evolutionary computation perspective, we pose the following question: What kind of behavior do genomic symbols present in successfully evolved individuals? After a brief

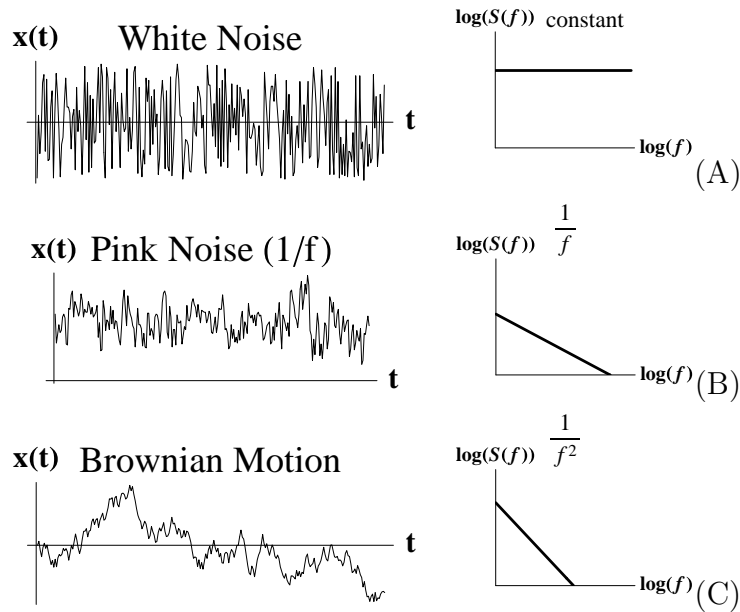


Figure 8.1: White(A), Pink(B) and Brownian Motion(C) noises and their spectral densities, $S(f)$.

analysis, it is easy to realize that for most evolutionary computation representations, which rely on an order-based encoding of information, this question has a trivial answer: the behavior exhibited by the evolved symbolic sequences is the one dictated by the chosen fitness function. For proteome-based location independent representations such as the PGA, this question becomes more interesting and less obvious. In proteome-based location independent representations, the encoding is based on which symbols are present on a genome and not on the order in which they are present [WG02]. Therefore, the order of genomic symbols is free to evolve along with the candidate solution they encode. We use spectral density measurement techniques to analyze the best individuals obtained by a PGA when applied to three problem domains: number matching, symbolic regression, and dynamical system control.

8.1.1 Symbolic Sequence Analysis

Autocorrelation and *spectral density* functions are widely used to analyze how fluctuations of a quantity, $X(t)$, are correlated between times t and $t + \tau$. Figure 8.1 shows examples of typical noises and their characteristic spectral densities $S(f)$. Voss [Vos93] has shown that this time correlation analysis can be successfully adapted to the analysis of symbolic

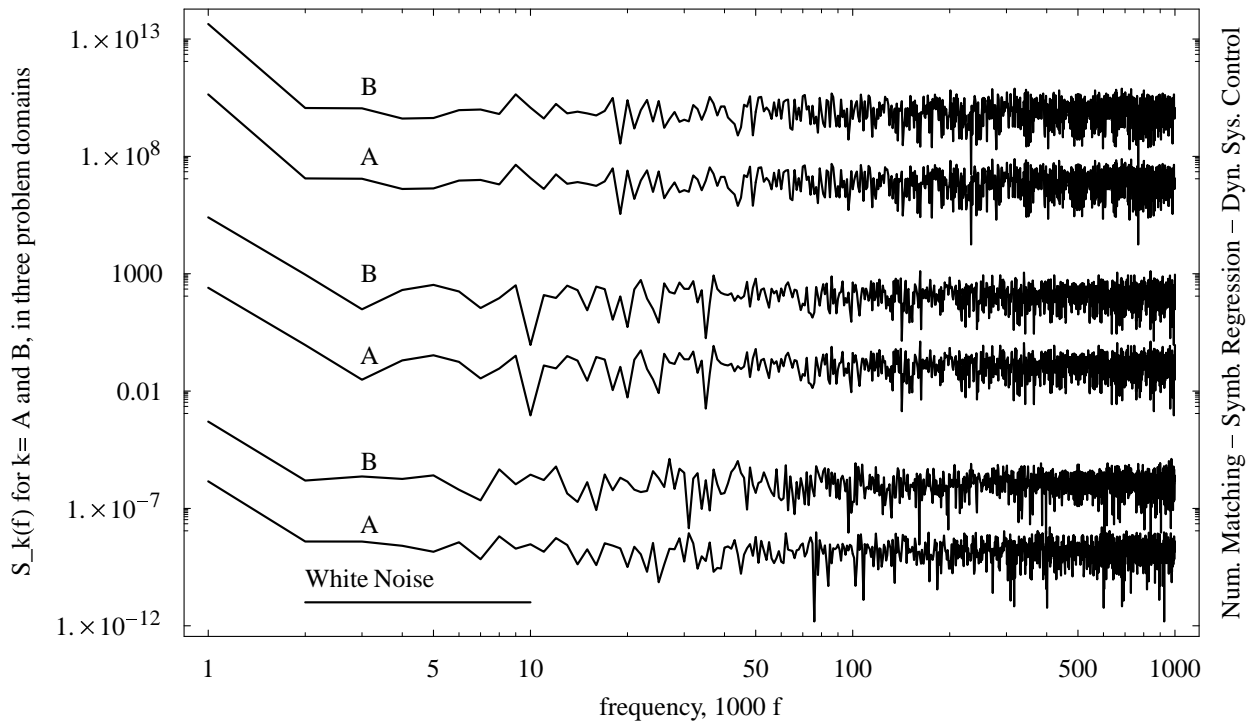


Figure 8.2: Log-log spectral density plots of the best of the last generation individuals. From the bottom up, symbols A and B for PGA on number matching, on symbolic regression and on dynamical system control problems. The graphics have been offset for clarity.

sequences such DNA. We use standard spectral density measurement techniques as adapted by Voss to analyze GA genomes using proportional representation. We analyze the best genomes from the last generation of a PGA run. These individuals are likely to be optimal or very close to optimal. Figure 8.2 shows the spectral density of the first two symbols of the alphabet (A and B) for three problems: number matching, symbolic regression, and control of dynamical systems. In all cases, we find that the emergent ordering of genomic symbols, according to our symbolic sequence analysis, resembles white noise. These results contrast with the original analysis of DNA sequences from GenBank data bank. In that study, $1/f$ noise and long-range fractal correlations are found over a broad range of DNA classifications such primate, invertebrate, plant, etc.

Regarding this contrast between the PGA and DNA spectral analysis, evolutionary computation processes are inspired by nature but do not pretend to model the complexities involved in biological evolution. Therefore, PGA is not a model to study DNA. Having said so, we think that the observed differences are due to the fact that DNA strings are a mixture of location dependent and location independent encodings, while the PGA is purely location

independent. Since the PGA is purely location independent, we expect that the emerging order is purely due to the dynamics of the algorithm, in our case white noise. In the DNA case, it seems that the pink noise arises due to long range dependencies in the encoded information.

8.2 Genomic Self-Similarity Analysis

The objective of this empirical study is to determine whether or not genomic self-similarity with respect to fitness emerges in the PGA proteome-based location independent representation. Based on previous PGA studies that suggest that PGA genome segments are coarse grained versions of an entire PGA genome, we expect the following behavior. Large genome segments will approximate the fitness of the entire genome very closely. The smaller the genome segments, the less they will resemble the fitness of the entire genome. There will be a cut-off point where a segment is too small to represent the required information and self-similarity is lost. As a result, we expect a gradual decrease in fitness as the segments get smaller until a cut-off point is reached at which point fitness will show a significant drop, as shown in Figure 8.3(C). As a baseline comparison, we use a regular GA in which genomic order is dictated by selection pressure and consequently not free evolve or self-organize. Exceeding our expectations, and as we will see shortly in the results section, PGA genomes resemble ideal self-similarity until the cut off point is reached.

The freedom of genomic elements to self-organize comes from the choice of representation. In the PGA proteome-based location independent representation the order on which the symbols are located on the genome, by definition, has no effect on the fitness of the individual. As a result, the genome is free to organize in response to other factors such the dynamics of the algorithms, operators, or building block processing. Because the PGA representation is based on a multiset, we can guarantee that no fitness function definition can distinguish between different orderings of the same symbols.

On the other hand, in a traditional representation where information is encoded on the order of the genomic symbols, there is no such freedom for genomic self-organization. On this case, fitness can potentially differentiate among various orderings and therefore the orderings can be subject to selective pressure. Unless the fitness function is specifically handcrafted to allow assure all possible orderings to have the same fitness value, a very unlikely scenario, on these cases the ordering of symbols will be determined by fitness. In our experiments, we do not handcraft such fitness for the traditional GA binary representation; hence, we do not expect self-similarity to emerge on GA genomes.

8.2.1 Self-Similarity Metric for Genomes

We use fitness as the metric for genomic self-similarity. A genome is self-similar if its fitness is approximately equal to the average fitness obtained by evaluating all of its genomic segments of a given segment length. More formally, let us introduce the following notation. A genome $g_{\langle 1,L \rangle} = g_1g_2g_3\dots g_L$, or simply g , is a string of length L over the genome alphabet Σ . A segment $s_{\langle i,j \rangle} = s_is_{i+1}s_{i+2}\dots s_j$ of $g_{\langle 1,L \rangle}$ is defined as the substring $g_i g_{i+1} g_{i+2} \dots g_j$, where $1 \leq i \leq j \leq L$. A fitness function $\mathcal{F}(g_{\langle 1,L \rangle})$ maps strings into real numbers and is defined for any genome or segment. Using the notation above, we define the average fitness of all segments of size r over genome $g_{\langle 1,L \rangle}$ as follows:

$$\hat{f}_r(g_{\langle 1,L \rangle}) = \frac{\sum_{i=1}^{L-r+1} \mathcal{F}(s_{\langle i,i+r-1 \rangle})}{(L-r+1)} \quad (8.1)$$

Note that for a segment equal to the entire genome ($r = L$), the expression above reduces simply to a fitness evaluation:

$$\hat{f}_r(g_{\langle 1,L \rangle}) = \mathcal{F}(s_{\langle 1,L \rangle}) = \mathcal{F}(g_{\langle 1,L \rangle})$$

Finally, we say that a genome is self-similar if the following expression is true:

$$\forall_r \left[\hat{f}_r(g_{\langle 1,L \rangle}) \cong \mathcal{F}(g_{\langle 1,L \rangle}) \right] \quad (8.2)$$

where r is the size of the genome segments used to analyze self-similarity.

The above equation implies the following. For an ideal case of genomic self-similarity, Equation 8.2 will hold indefinitely for any segment size r . In this case, genomic segments of any size will have the same fitness as the entire genome, as shown in Figure 8.3(A). This case is analogous to perfect fractal behavior. On the other hand, if there is no self-similarity, Equation 8.2 will not hold even for large segments. In this case, the fitness of the segments will not resemble the fitness of the entire genome. We thus expect random segments with average fitness equal to the median fitness of the problem as demonstrated in Figure 8.3(B).

8.2.2 Fitness Evaluation

For all experiments, we set the algorithms to solve a very simple problem: *number matching*. Number matching is a simple hamming distance type of problem for real numbers. The goal

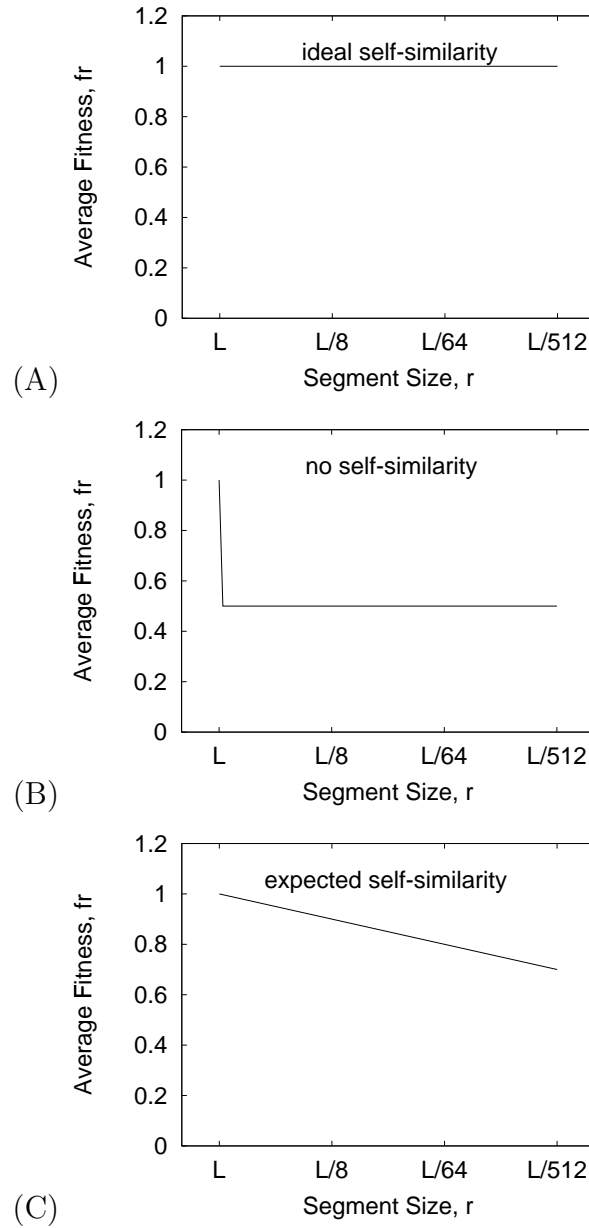


Figure 8.3: Predicted genomic self-similarity in populations that have converged to the optimal fitness of 1. (A) Ideal self-similar case: all genomic segments have fitness of 1 regardless of their size. (B) No self-similarity case: segments have random fitness with average equal to the fitness median. (C) PGA expected self-similarity: segments gradually reduce in fitness as they get smaller. Experimental results resemble the ideal case (see Figure 8.4).

g replacements
 Segment Size, r
 ts, $\hat{f}_r(g_{<1,L>})$

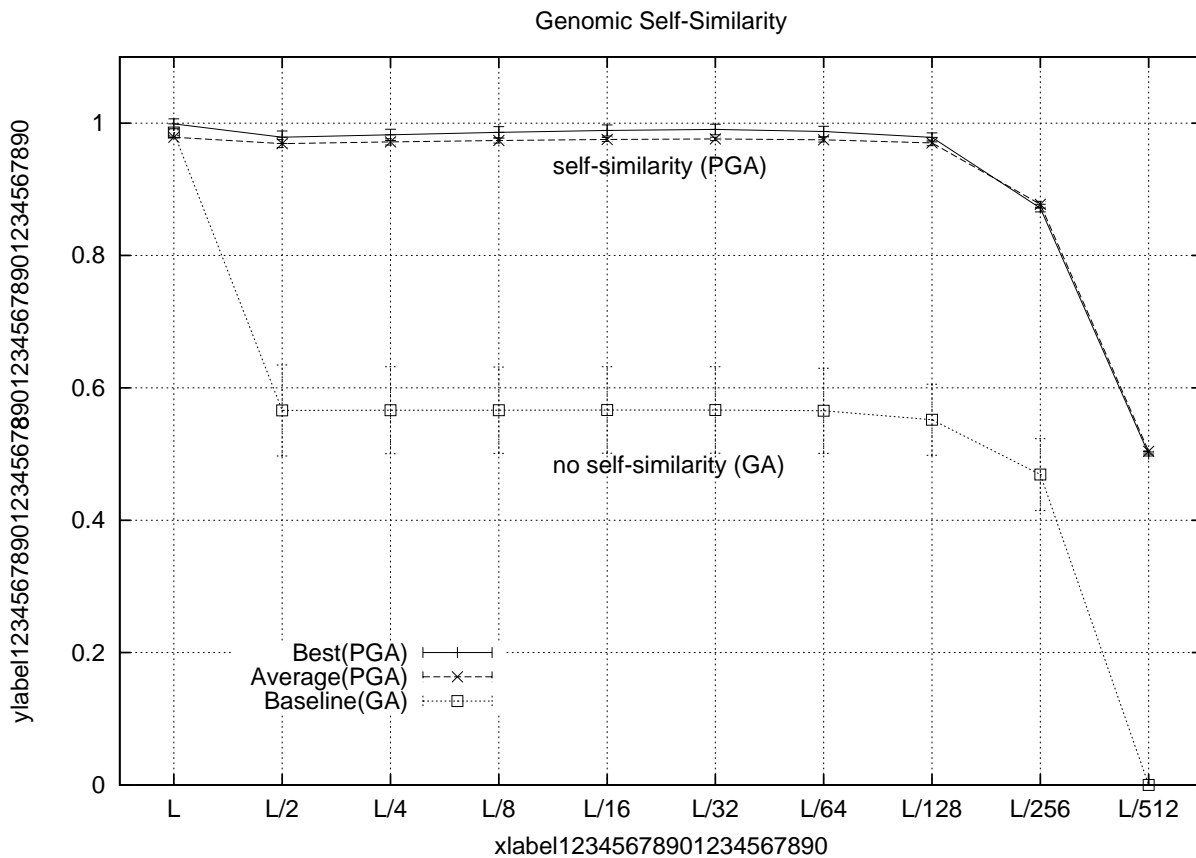


Figure 8.4: PGA best, PGA average, and GA baseline comparison of average fitness of genome segments($\hat{f}_r(g_{<1,L>})$) of sizes(r) L(entire genome), L/2(half genome), L/4, ... , and L/512 on a simple number matching problem averaged over 20 runs with 95% confidence intervals. (Top: self-similarity) PGA segment fitness remains approximately equal to the entire genome fitness for segment sizes of L/2, L/4, ... until L/128. (Middle: no self-similarity) No GA segment has average fitness close to the entire genome fitness for any segment size. Baseline note: for clarity, only GA average is shown; GA best is not shown but has similar behavior.

is to match, as closely as possible, a vector of target numbers. We use vectors of size two. For convenience of exposition, let us define the following function:

$$\delta(x, y) = \begin{cases} x/y & \text{if } x < y \\ y/x & \text{otherwise} \end{cases} \quad \forall x, y \in \mathfrak{R}^+.$$

Fitness is determined by the average difference, as measured by δ , between the values encoded in the individual's genome (or segment) and the target values:

$$\mathcal{F}(g_{\langle 1, L \rangle}) = \frac{\delta(v_{target}^1, v_{encoded}^1) + \delta(v_{target}^2, v_{encoded}^2)}{2}$$

Where, $V_{target} = \{v_{target}^1, v_{target}^2\}$ is the vector of target values, and $V_{encoded} = \{v_{encoded}^1, v_{encoded}^2\}$ is the vector of values encoded in g . For the PGA, the encoded values are given by:

$$V_{encoded}(g) = \left[\frac{|g|_a}{|g|_a + |g|_b}, \frac{|g|_c}{|g|_c + |g|_d} \right]$$

where the genomic alphabet is $\Sigma_{PGA} = \{a, b, c, d\}$, and $|X|_y$ returns the number of times a symbol y appears in the multiset associated with string X . Note that this definition is also valid for genomic segments. Note also that only the associated multisets are used to calculate the values. The first value is encoded simply as a proportion of the concentrations between symbols a and b , and the second value, between symbols c and d . The proportions are numbers between zero and one, but they can be easily scaled to represent any parameter range of values.

For the GA, the values encoded in the genome are interpreted in the usual way. The genomic alphabet is binary: $\Sigma_{GA} = \{0, 1\}$. The first half of the genome represents our first encoded value and the second half, our second encoded value. Note that this definition is also valid for genome segments.

8.2.3 Settings

For our experiments, we use a standard GA a PGA. The GA uses a binary alphabet; the PGA uses a multicharacter alphabet. Mutation, in the GA, is bit-flip mutation. The PGA mutation, randomly changes one alphabet symbol into another. The following parameter settings are common for all experiments: the genome length is $L=1000$ and segment sizes are $L/2, L/4, L/8, L/16, L/32, L/64, L/128, L/256, \text{ and } L/512$, the crossover type is two-point, the crossover rate is 0.8, the mutation rate is 0.005, the selection method is tournament of size 4, the population size is 250, and the number of generations is 500. We perform 20 trials for all experiments using new randomly generated targets for each run, and report average values with their 95% confidence intervals.

8.2.4 Results

Figure 8.4 shows plots of the average measured fitness for decreasing segment length. These results reveal genomic self-similarity with respect to fitness for the PGA and no self-similarity for the baseline case. The self-similarity is almost ideal (see Figure 8.3 (A)) for segments as small as $L/128 = 1000/128 = 7.81$ symbols. After this critical point, self-similarity is lost and fitness of the segments decreases significantly. Figure 8.5 shows the raw fitness of segments of size $L/2$ for the two encoded values over the 500 generations. In Figure 8.5 (A), the variance for the self-similar case is very small. This result indicates that all segments, not just their averages, must have fitness similar to the entire genome. In Figure 8.5 (B), the variance for the non-self-similar case is very high, indicating that the segments have a wide range of fitness values and are not similar to the entire genome.

8.3 Discussion

Empirical observations from section 7.2.3 and from this chapter indicate that, during the course of the evolutionary process, PGA genomes tend to self-organize into a self-similar state with respect to fitness. In this state, alphabet symbols are evenly distributed throughout the genome. The question, then, is: why does this particular organization emerge when there is no fitness pressure for any particular ordering?

Self-similarity with respect to fitness is possible because PGA looks for proportions. If the desired proportion of symbols α and β is 2 : 1, and if the symbols are evenly distributed throughout the genome, the genomic segments will have roughly the correct proportion. This self-similarity reduces building block disruption. Building block disruption occurs when a crossover operation destroys useful schemata. In the PGA case, a genome self-organizes into self-similar building blocks. These building blocks are not disrupted by typical crossover operations because a sub-segment is itself a representative of the entire building block or schema. As a result, crossover disruption is minimized, if not eliminated, on the PGA genomes once this self-similarity has emerged.

Self-similarity reinforces positive correlations between form and quality of candidate solutions. Let us analyze the effects of self-similar genomes on one-point crossover and single-symbol mutation. Assume that P_1 and P_2 are two individuals with perfect, self-similar genomes of length L over the alphabet $A = \{a, b\}$. P_1 and P_2 undergo one-point crossover at location l and, as a result, the individuals P_3 and P_4 are produced. The concentrations of symbols a and b on individuals P_1 and P_2 are given by:

$$a_{p1} = \frac{|P_1|_a}{L} \quad b_{p1} = \frac{|P_1|_b}{L}$$

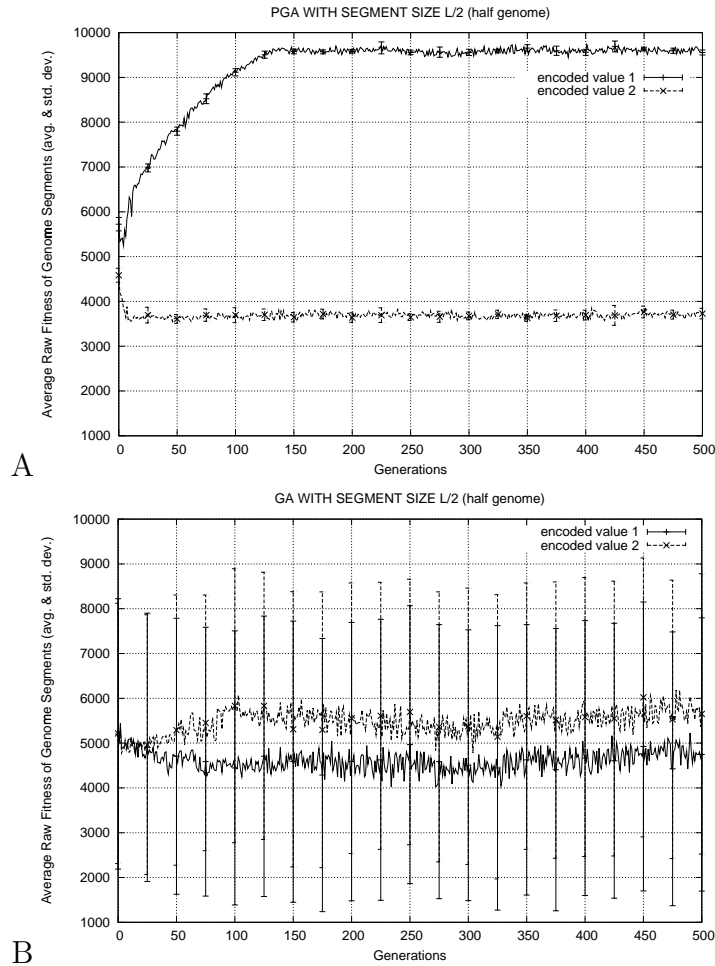


Figure 8.5: Average encoded values 1 and 2 of genomic segments of size $L/2$ (half-genome) over 500 generations. The plots show averages and standard deviations over 20 runs. (A) PGA encoded values 1 and 2 have tight standard deviations that indicate that encoded values of PGA genomic segments of size $L/2$ converge. From Figure 8.4 we observe that their fitness converges to a near optimal solution, similar to the fitness of the entire genome L . (B) GA encoded values 1 and 2 have wide standard deviations that indicate that GA genomic segments of size $L/2$ do not converge. Furthermore, the encoded values of these segments appear to be randomly distributed. From Figure 8.4 we observe that their average normalized fitness is not similar to the entire genome, but near to the median 0.5 instead. We observe similar behavior for segments of size $L/4$ to $L/128$.

$$a_{p2} = \frac{|P_2|_a}{L} \quad b_{p2} = \frac{|P_2|_b}{L}$$

It is easy to see that after crossover, the concentrations of the resulting offspring P_3 and P_4 are:

$$a_{p3} = \frac{\frac{|P_1|_a}{L} \times l + \frac{|P_2|_a}{L} \times (L - l)}{L}$$

$$a_{p4} = \frac{\frac{|P_1|_a}{L} \times (L - l) + \frac{|P_2|_a}{L} \times l}{L}$$

$$b_{p3} = \frac{\frac{|P_1|_b}{L} \times l + \frac{|P_2|_b}{L} \times (L - l)}{L}$$

$$b_{p4} = \frac{\frac{|P_1|_b}{L} \times (L - l) + \frac{|P_2|_b}{L} \times l}{L}$$

Hence, the symbol concentration of the offspring produced as a result of one-point crossover between parents with self-similar genomes is the weighted average of the symbol concentration of the parents. In terms of mutation, let us assume that P_1 undergoes single-symbol mutation in the following way: a single symbol is changed from a to b . Before mutation, the concentrations of symbols in P_1 are given above. After mutation, we have:

$$a'_{p1} = \frac{|P_1|_a - 1}{L} \quad b'_{p1} = \frac{|P_1|_b + 1}{L}$$

The change in concentrations is given by:

$$\Delta_M a_{p1} = \left| \frac{|P_1|_a}{L} - \frac{|P_1|_a - 1}{L} \right| = \frac{1}{L}$$

$$\Delta_M b_{p1} = \left| \frac{|P_1|_b}{L} - \frac{|P_1|_b + 1}{L} \right| = \frac{1}{L}$$

This concentration change equals the minimum possible change allowed by the encoding: $1/L$.

Clearly, the behavior of crossover and mutation applied to self-similar genomes produces smooth moves in the representation space. Crossover acts by averaging parent concentrations and mutation acts by changing an individual by the minimal difference. This smoothness of the proteome-based location independent representation will certainly not hinder, and will probably accentuate the key property required for stochastic search: positive correlation between the form and quality of solutions.

8.4 Summary

This chapter study the emergent organization of proteome-based location independent genomes using two methods: standar expectral density analysis of equal-symbol correlations and empirical fitness comparison of genomic segments to determine self-similarity with respect to fitness. The equal-symbol correlatio analysis shows that the order of genomic symbols resembles white noise. This confirm results on section 7.2.3 that show that in PGA genomes, symbols are evenly distributed throughout the entire genome. The empirical analysis offers evidence of the emergence of genomic self-similarity with respect to fitness in PGA genomes. When genomic order is free to evolve, the genome self-organizes in response to the dynamics of the evolutionary system. In the PGA case, it self-organizes into a fractal-like structure in which genomic segments have approximately the same fitness as the entire genome.

CHAPTER 9

CASE STUDY: ADVANCE LIFE SUPPORT SYSTEM

As technology advances in the space industry, the opportunity and need for long term extraterrestrial missions will increase. Such missions include both missions in space and extraterrestrial planetary habitats. In all cases, a necessary element for the success and productivity of a mission is an effective Advanced Life Support System (ALSS). A closed habitat such as a planetary or space habitat is composed of multiple components, e.g. crew, air, water, plants, climate, stores, and waste, all of which interact in a complex manner. An ALSS controls the necessary processes that regenerate basic life support products such as air, water, and food within a habitat. As a result, it minimizes the need to store large amounts of consumable products and minimizes the need for frequent resupply of such products.

Energy is required to produce food, regenerate air, purify water, and process waste. A closed habitat may be limited to a pre-defined amount of stored energy or may be exposed to a reliable, fixed energy source such as the sun or regular air currents. The task of an ALSS is to distribute available energy resources among the required tasks as efficiently as possible to maximize the productivity and lifetime of a habitat. The tight coupling of the system components and inherent unpredictability of some of them, e.g. the crew and climate, make this a complex task. Nevertheless, many of the decisions and tasks of an ALSS are routine and the development of automated or semiautomated systems can significantly reduce human effort on tedious monitoring tasks.

An ALSS is an example of a class of systems called coupled dynamical systems. Such systems consist of deterministic subsystems whose behaviors are easy to predict in isolation; but the behavior of the complete system is difficult to predict and, thus, difficult to control. The majority of work on ALSS control to date has focused on modeling and control of individual subsystems within a life support system, e.g. the water recovery system or the air revitalization system. Control algorithms are typically reactive to changing conditions but deliberative and tailored to a specific module or set of modules [BFG97, CPF00, STB02]. While deliberative control systems may suffice under static conditions, some element of learning or adaptability is likely to be necessary in real world scenarios. Kortenkamp et al. [KBS01] present the first study which successfully uses machine learning (ML) techniques to learn how to control an ALSS as a whole.

Table 9.1: Vector of control parameters, \vec{c}^t , of ALSS simulator.

| Parameter | Description | Values | Type |
|-------------------------|-----------------------------------|-------------------|---------|
| $c_{energy.to.air}^t$ | Energy alloc. to air processing | $0 \leq x \leq 5$ | Real |
| $c_{energy.to.water}^t$ | Energy alloc. to water processing | $0 \leq x \leq 5$ | Real |
| $c_{energy.to.food}^t$ | Energy alloc. to food production | $0 \leq x \leq 5$ | Real |
| $c_{water.to.crew}^t$ | Clean water allocated to crew | $0 \leq x \leq 5$ | Real |
| $c_{water.to.crops}^t$ | Clean water allocated to crops | $0 \leq x \leq 5$ | Real |
| $c_{activity.level}^t$ | Activity level of crew | 0, 1, 2, 3 | Integer |
| $c_{store.water}^t$ | Use resources from water store | 0, 1 | Integer |
| $c_{store.air}^t$ | Use resources from air store | 0, 1 | Integer |
| $c_{store.food}^t$ | Use resources from food store | 0, 1 | Integer |

In this paper, we extend previous work [KBS01] by performing a detailed comparison of genetic algorithm (GA) and stochastic hill-climbing (SH) approaches to ALSS control. Evaluation of candidate solutions is performed on a newer and more complete ALSS model than previously available. As a result, successful results lend even stronger support to the viability of ML techniques in the development of autonomous ALSS controls. For both algorithms, problem representation determines the shape of the landscape which determines the solutions that are reachable from any particular solution. We examine the performance of these algorithms using two significantly different types of problem representations.

9.1 Background

9.1.1 Advanced Life Support System Simulator

We have implemented an ALSS simulator that models the basic processes occurring in the Bioregenerative Planetary Life Support System Test Complex (BIO-Plex) developed by NASA Johnson Space Center [BCF99, KBS01, Tri99]. This ALSS simulator is used to evaluate the candidate control strategies generated by our algorithm. The ALSS simulator is a simplified model of the actual BIO-Plex simulator because of time and computational constraints; however, our learning algorithm should be easily linkable to the BIO-Plex simulator if desired.

In each time step, the ALSS accepts as input the vector of control parameters \vec{c}^t listed in Table 9.1. A continuous value from zero to five indicates energy allocation to water, air, and food processing, as well as water allocation to crew and crops. Larger values indicate more energy and water. There are four discrete activity levels for the crew: sleep (0), low activity (1), moderate activity (2) and high activity (3). A binary value each indicates whether or not water, air, and food stores will be used. At the end of a simulator run, the ALSS outputs

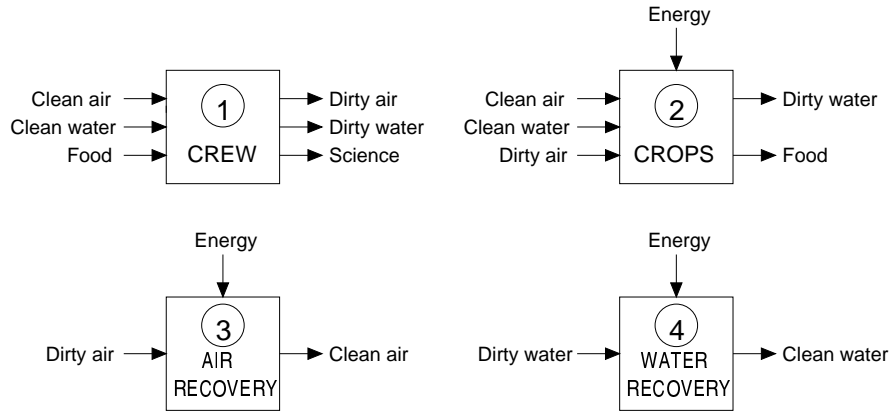


Figure 9.1: Basic transitions in the ALSS simulator.

measures of mission duration in simulation time steps or ticks (approximately one hour) and mission productivity in units of “science” for that particular run. A simulator run ends when resources are exhausted and the environment becomes unable to support human life. The diagrams in Figure 9.1 describe the basic processes and transformations that take place in the ALSS simulator in each time step.

1. The crew process consumes clean air, clean water, and food from the environment and produces dirty air, dirty water and *science*. The consumption and production rates are determined by the *activity level* input parameter. The higher the activity level, the greater the consumption of resources and production of resulting products. An internal simulation parameter, *crew status*, represents the health of the crew. A lack of resources for multiple consecutive time steps results in crew extinction and the end of the simulation.
2. The crops process consumes clean air, clean water, dirty air, and energy. It produces dirty water and food. Production of food depends on the *crop status* and on the availability of the consumed products. The *crop status* represents the current crop health, and decreases if there is insufficient air, water, or energy. Lack of consumables over multiple consecutive time steps results in crop expiration and the end of food production.
3. The air recovery process consumes energy and dirty air and produces clean air. The amount of clear air produced depends on the available energy. Lack of energy for three consecutive time steps shuts down this process (*shut down period*). If energy is restored, the process requires two consecutive time steps with adequate energy supply, (*warm up period*) to restart the production of clean air.
4. The water recovery process consumes energy and dirty water; and produces clean water. This process is analogous to the air recovery system, but with warm up and shut down periods of ten and five simulation time steps, respectively.

A detailed mathematical description of the simulator is given in Appendix A.

9.1.2 Related Work

Automated and semi-automated control of ALSS attempts to relieve the workload of human operators and crew members, moving them from a “vigilant monitoring” role to a “supervisory monitoring” role and allowing them more time to perform the more complex tasks of a mission [STB02]. Although live subject experiments have been and continue to be performed, mathematical modeling and simulation studies are also prevalent due to their significantly lower cost and high controllability. Such simulations use mathematical models to approximate and simulate the interactions that are expected to occur for any given set of input conditions [Fin99, FT01, GT99, KB03a]. Studies of autonomous or semi-autonomous control of various individual subsystems of an ALSS include a three-tier hierarchical control of water recovery and air regeneration systems [STB02], a market auction based approach to managing power allocation and surges for a partially modeled ALSS [CPF00], and various approaches for control of environmental management systems [Fin99, FT01, STB02].

Kortenkamp et al. [KBS01] are the first to evaluate ML methods for learning effective ALSS energy allocations using an early version of the BIO-Plex simulation. They find both the GA and reinforcement learning to be viable choices for maximizing either or both the mission duration and productivity of a single simulator run. Two different GA implementations are tested. In the first GA method, each GA individual specifies a single set of control parameters. The fitness of an individual is based on the duration or productivity of a run which uses that individual’s parameter setting in every time step. A fitness function that optimizes for mission duration produced an optimal run of 31 ticks with 148 units of science. A fitness function that optimizes for productivity generated a maximum of 184 units of science with mission length of 26 ticks. By comparison, reinforcement learning performs slightly better than the GA in terms of mission duration and about twice as good in terms of productivity. The second GA approach allows each GA individual to specify multiple, n , sets of parameter values. These n sets of values are used in n consecutive time steps in a simulation run. The fitness of an individual is based on the duration and productivity of a simulation over the n time steps it takes to use all parameter sets specified by that individual. The next individual to be evaluated executes its parameter values in the next n time steps of the same simulation. This multi-step GA produces a significant improvement in performance reaching a mission duration of 702 ticks (when optimizing for ticks) or productivity of over 1800 units of science (when optimizing for science). Increasing the number of sets of parameters, n , and the target fitness values over multiple runs appears to improve the quality of solutions generated.

Within the area of resource allocation, GAs have been successfully applied to number of different types of problems. For example, in wireless communications, GAs are able to

Table 9.2: Constant parameter settings for all GA runs.

| | |
|------------------|--|
| Population size | 100 |
| Generations | 500 |
| Selection method | tournament |
| Crossover type | two-point (fixed length), homologous (variable length) |
| Crossover rate | 1.0 |
| Mutation rate | 0.005 |

learn effective and adaptive frequency allocation strategies [SHN99, SHN00, Wil00]. Military applications have used GAs to allocate resources such as aircraft [ABB98], weapons [OS94], and sorties [PZS01]. Lau and Tsang [LT98] apply a hybrid algorithm combining a GA with Guided Local Search to solve the Generalized Assignment Problem. Callaghan et al. [CNL99] use a GA to optimize land allocation. Baglioni et al. [BSd00] describe an evolutionary approach to financial asset allocation. Cousins et al. [CLR98] describe a hybrid GA method for memory allocation in high performance computing systems. Papavassiliou et al. [PPT01, PPT02] examine the integration of agents and GAs for network management. The problem of optimizing resource allocation is closely related to scheduling and routing problems that have been widely studied in the GA field [Bru97, Mic98, Nis97].

9.2 Algorithm Descriptions

9.2.1 Genetic Algorithm

A genetic algorithm [Hol75b, Gol89] is a learning algorithm based on principles from natural selection and genetic reproduction as described in section 2.1.1. Table 9.2 gives the GA parameter settings used in this work. Values were experimentally selected to provide good performance.

9.2.2 Stochastic Hill Climbing

Hill-climbing [Ack87, Dav91] is a simple optimization algorithm that has proven to be a competitive alternative to GAs. If we visualize an optimization problem as a landscape on which each point corresponds to a solution and its height corresponds to the fitness of the solution, a hill-climbing algorithm searches for peaks by repeatedly moving to adjacent points of higher fitness. Stochastic hill-climbing (SH) is a probabilistic variant of hill-climbing in

which the search of adjacent points is stochastic and not deterministic. In the GA community, SH is a widely accepted baseline test of GA effectiveness for given problems [JW95, MHF94]. The SH algorithms used here adopt the *problem representation*, the *fitness function* and the *mutation operator* of their corresponding GA variant, but they all share the following search procedure:

1. Choose one individual at random. Call this individual the *best-individual*.
2. If termination condition satisfied, stop and return *best-individual*.
3. Mutate *best-individual*. If the mutation produces an individual with higher fitness, then set *best-individual* to mutated individual. Go to step 2.

The termination condition is set to 50,000 fitness evaluations—equal to the number of evaluations in each GA run. Mutation probability was experimentally optimized and set to 0.05.

9.3 Fitness Definitions

9.3.1 The Optimization Problem

The ALSS simulation, the optimization problem, and the fitness functions are formally defined in Appendix A. Informally, an ALSS simulation (Definition 1)¹ is a coupled dynamical system with a set of internal simulation states (Definition 3), a control strategy (Definition 5) and a transition function (Definitions 6 to 10). The next step of a simulation (Definition 11) is calculated deterministically using the transition equations that compute the next simulation state based on the current state and the current control variables provided by the control strategy. The ML problem is to find a controller or, equivalently, to find a set of control variables to be used in each simulation step that optimizes one or more outcomes of a simulation. A simulation ends when its environment is no longer able to support human life (Definition 13). In this paper, we study three different optimization problems of interest:

1. Finding a control strategy that maximizes mission productivity (Definition 15).
2. Finding a control strategy that maximizes mission duration (Definition 16).
3. Finding a control strategy that optimizes both mission productivity and duration (Definition 17).

¹All definitions are in Appendix A

9.3.2 The Control Strategies

The *control strategy* (Definition 5) throughout this paper uses n vectors of control parameters circularly within a simulation. Therefore, the optimization problem for the various algorithms we test—GA and SH with both binary and proportional representations—is restricted to learning the appropriate n vectors of control parameters that, when circulated in a simulation, produces a maximized outcome of mission productivity, duration, or both. We call this control strategy Γ_n . We perform experiments with $n = 1, 3, 5, 7,$ and 9 vectors of parameters and refer to these instantiations of the control strategy as $\Gamma_1, \Gamma_3, \Gamma_5, \Gamma_7, \Gamma_9,$ respectively. Each individual or a candidate solution is a set of n vectors of control parameters.

9.3.3 The Fitness Measures

The fitness of an individual is determined by running an ALSS simulation using an individual’s n vectors and control strategy Γ_n . We use the following fitness measures for our experiments:

1. The total amount of “science” produced by the crew when the simulation ends; when optimizing mission productivity (Definition 18).
2. The number of simulation time steps at which the simulation ends; when optimizing mission duration (Definition 19).
3. A weighted addition of the previous two measures; when optimizing both (Definition 20).

9.4 Problem Representation

How a problem is represented in an ML algorithm determines what can be expressed and how the solution space is connected which in turn determines the concepts that can or cannot be learned. How information is represented in a learning algorithm can affect an algorithm’s expressivity, efficiency, and readability [Sam94]. We compare the performance of a GA and SH on two drastically different problem representations: binary representation and proportional representation.

0010 0101 1000 1001 0111 11 1 0 1

Energy to water >
 Energy to air >
 Energy to food >
 Water to crew >
 Water to crops >
 Crew activity level >
 Use water store >
 Use air store >
 Use food store >

| Parameter | Expressed Value |
|---------------------|------------------------|
| Energy to water | $2/15 \times 5 = 0.67$ |
| Energy to air | $5/15 \times 5 = 1.67$ |
| Energy to food | $8/15 \times 5 = 2.67$ |
| Water to crew | $9/15 \times 5 = 3$ |
| Water to crops | $7/15 \times 5 = 2.3$ |
| Crew activity level | 3 |
| Use water store | 1 (Yes) |
| Use air store | 0 (No) |
| Use food store | 1 (Yes) |

Figure 9.2: A binary encoded individual and its corresponding encoded values.

9.4.1 Binary Representation

Binary representation is commonly used in algorithms that use linear problem representations such as GAs and SH. A typical binary representation consists of a field of bits for each encoded parameter value. For our problem, each vector of control parameters is a 25 bit string. We use four bits to represent each of the five real values shown in Table 9.1. Four bits provide a resolution of sixteen possible values which are scaled to a real value between zero and five. We use two bits to represent the “activity level”. The activity level ranges from zero to three and is indicated by the binary value of the two bits. We use a single bit for each binary value representing whether or not to use resources from the stores. With this representation, each individual is a $25 \times n$ bit string formed from the concatenation of n 25 bit strings, each representing one vector of parameters. Figure 9.2 shows an example individual for $n = 1$ and values that it encodes. The GA uses two-point crossover and bit flip mutation. The SH uses bit flip mutation.

The strengths of binary representations are that they are very space efficient and that they are logical and easy for humans to interpret. Their weaknesses include brittleness (missing, extra, or misplaced bits can severely change encoded values) and the fact that they are subject to positional biases found in order-based encodings [ECS89].

Table 9.3: Proportional representation character assignments for ALSS problem.

| Parameter value V | $p_char(V)$ | $n_char(V)$ | V_{min} | V_{max} |
|---------------------|--------------|--------------|-----------|-----------|
| Energy to water | A | a | 0 | 5 |
| Energy to air | B | b | 0 | 5 |
| Energy to food | C | c | 0 | 5 |
| Water to crew | D | d | 0 | 5 |
| Water to crops | E | e | 0 | 5 |
| Crew activity level | F | f | 1 | 3 |
| Use water store | G | g | 0 | 1 |
| Use air store | H | h | 0 | 1 |
| Use food store | I | i | 0 | 1 |

9.4.2 Proportional Representation

The proportional representation attempts to address the weaknesses of order-based encodings by eliminating the notion of order altogether. This representation was initially developed for a GA [WG02], but can also work with other algorithms such as SH. Encoded information depends solely on what does and does not exist on an individual and not on the order in which it is present. As a result, the order of the encoding is free to evolve in response to factors other than the expressed solution, for example, in response to the identification and formation of building blocks.

The proportional representation uses a linear genome with a multi-character alphabet. One or more unique characters are assigned to each parameter or component of a solution. The value of a parameter is determined from the relative proportions of the assigned characters of that parameter. Thus, characters that exist are “expressed” and, consequently, interact with other expressed characters. Characters that do not exist are “not expressed” and do not participate in the interactions of the expressed characters.

For example, in the ALSS problem, we are searching for n vectors \vec{V} of $a = 9$ parameter values, $V_i, i = 0, \dots, a - 1$. The range of these values is given in Table 9.1 and will be denoted by $V_{i,min}$ and $V_{i,max}$. We assign a “positive” character, p_char , and a “negative” character, n_char , to each parameter as shown in Table 9.3.

The number of positive and negative characters on an individual are used to calculate the proportion of positive characters, pct , as follows:

$$pct(V_i) = \frac{p_char(V_i)}{p_char(V_i) + n_char(V_i)}$$

where $i = 0, \dots, a - 1$ and $0.0 \leq pct(V_i) \leq 1.0$. The value of each parameter is then calculated by the equation

$$P_{PGA}(V_i) = V_{i,min} + pct(V_i) \times (V_{i,max} - V_{i,min}).$$

AccBfFhIiAAGgDEGGGaCFbIbABhFGgfCbhhAffeA

Figure 9.3: An example proportional representation individual of length 40 with a single vector ($n = 1$).

Table 9.4: Allocation of resources as specified by example individuals from Figures 9.3 and 9.4.

| Value V | $p_char(V)$ | $n_char(V)$ | $pct(V)$ | $P_{PGA}(V)$ | Expressed value |
|---------------------|--------------|--------------|-------------|--------------|-----------------|
| Energy to water | 6 A's | 1 a's | $6/(6 + 1)$ | 4.29 | 4.29 |
| Energy to air | 2 B's | 3 b's | $2/(2 + 3)$ | 2.0 | 2.0 |
| Energy to food | 2 C's | 2 c's | $2/(2 + 2)$ | 2.5 | 2.5 |
| Water to crew | 1 D's | 0 d's | $1/(1 + 0)$ | 5.0 | 5.0 |
| Water to crops | 1 E's | 1 e's | $1/(1 + 1)$ | 2.5 | 2.5 |
| Crew activity level | 3 F's | 4 f's | $3/(3 + 4)$ | 1.86 | 2 |
| Use water store | 5 G's | 2 g's | $5/(5 + 2)$ | 0.71 | 1 |
| Use air store | 0 H's | 4 h's | $0/(0 + 4)$ | 0.0 | 0 |
| Use food store | 2 I's | 1 i's | $2/(2 + 1)$ | 0.67 | 1 |

A typical single-vector individual ($n = 1$) of length 40 such as the one shown in Figure 9.3 encodes the values shown in Table 9.4. As the expressed values are completely independent of the arrangement of characters, the individual in Figure 9.4 also encodes the values shown in Table 9.4.

A fixed length GA uses two-point crossover. A variable length GA uses homologous crossover [BDG98] which randomly selects a crossover point on the first parent, then finds the region of highest homology (similarity) on the second parent. Crossover occurs within the region of homology. Mutation for both a GA and SH randomly switches one character to another character in the alphabet. All characters are equally likely.

AAAAAAaBBbbbCCccDEeFFFffffGGGGGgghhhhIIi

Figure 9.4: Another example proportional representation individual of length 40 with a single vector ($n = 1$). Encodes equivalent solution as individual from Figure 9.3.

9.4.3 How to Fairly Compare Binary and Proportional Representations

Although the proportional representation eliminates positional biases, it is clearly a less compact encoding than binary representation. Binary individuals are strings over a binary alphabet while proportional individuals are strings over a higher-arity alphabet. As a result, binary and proportional individuals of identical length will likely encode solution spaces of different sizes. In order to fairly compare the two representations we need to find a relationship among their individual lengths and alphabet sizes to ensure that both algorithms target solution spaces of comparable complexity².

A binary individual of length l_{bin} over an alphabet with cardinality n_{bin} encodes $(n_{bin})^{l_{bin}}$ different binary strings which represent $(n_{bin})^{l_{bin}}$ different solutions. The size of the search space and the solution space are equal for binary representation. A proportional individual of length l_{pro} over an alphabet with cardinality n_{pro} encodes $(n_{pro})^{l_{pro}}$ different multi-character strings. These strings are the search space of this proportional representation. According with section 6.1.4, they map to a solution space of $\binom{n_{pro}+l_{pro}-1}{l_{pro}}$ different solutions. Thus,

$$(n_{bin})^{l_{bin}} = \binom{n_{pro} + l_{pro} - 1}{l_{pro}} \quad (9.1)$$

must hold for a binary and proportional representation to encode solution spaces of the same size. Consequently, a proportional encoding length will typically be longer than an equivalent binary length.

A binary encoded individual requires a total length of 25 bits to represent a single vector of values and $25 \times n$ bits to represent n vectors. We can use Equation 9.1 to calculate the corresponding proportional representation lengths for each value in a vector, then sum over all values to obtain the total equivalent proportional representation length. For example, $n_{bin} = 2$ and $l_{bin} = 4$ for the binary representation of the first parameter—energy allocated to water processing. The proportional representation uses two characters to represent any single parameter, thus $n_{pro} = 2$. Using Equation 9.1, we calculate $l_{pro} = 15$ for the first parameter. We can compute the remaining equivalent parameter lengths for the proportional representation in the same way, to obtain a total equivalent proportional representation length of $l_{pro}(1) = 15 + 15 + 15 + 15 + 15 + 2 + 1 + 1 + 1 = 80$ for a single vector. The required proportional representation length for n vectors is $l_{pro}(n) = l_{pro}(1) \times n = 80 \times n$.

We compare binary representation to three proportional representations of differing lengths, Len , and consequently, differing resolutions:

1. *PSame*: Fixed length proportional representation in which the length of an individual is the *same* as the length of a binary representation individual, $Len = l_{bin}$. This length is 25 for a single vector of parameters and $25 \times n$ for n vectors of parameters. According

²We define the *search space* to be the space of all encodings and the *solution space* to be the space of all solutions.

to Equation 9.1, *PSame* is severely penalized in terms of available resolution due to its restricted genome length.

2. *PHalf*: Proportional representation with a fixed length of one *half* the equivalent length, $Len = l_{pro}/2$. This length is 40 for a single vector case and $40 \times n$ for n vectors. *PHalf* is still penalized in terms of available resolution; however, less so than *PSame*.
3. *PMax*: Proportional representation with the equivalent length, $Len = l_{pro}$. This length is 80 for the single vector case and of $80 \times n$ for the n vectors case. To save space, we allow the GA to evolve variable lengthed individuals with a maximum length of Len . No parsimony pressure is applied. The SH uses fixed length individuals of length, Len .

The proportional representation typically requires two characters for each parameter, thus, the total number of characters required to represent n vectors of nine parameters should be $2 \times 9 \times n$. As the precision of similar parameters is likely to be comparable, it is reasonable to use the same set of negative characters for all corresponding vector components. The resulting PGA requires only $9 \times (n + 1)$ characters to represent n vectors of nine parameters. For example, consider the case of $n = 3$ vectors. The first parameter of each vector represents the value of *energy to water*. Let all three values share the same negative character: a ; but have unique positive characters: A for the value in the first vector, J for the second and S for the third. Similarly, the second parameter (representing the value of *energy to air*) of all three vectors share the same negative character: b ; but have unique positive characters: B , K and T , respectively. We use this sharing strategy for all the experiments with proportional representation.

9.5 Comparative Experimental Analysis

We compare the performance of a GA and SH in finding control parameter values for the three optimization problems described in section 9.3.1 using the five control strategies described in section 9.3.2 for the ALSS simulation. In each set of experiments, we test eight algorithms: GA and SH each using four different representations, *Binary*, *PSame*, *PHalf*, and *PMax*, as described in section 9.4.3. In total, we perform 120 experiments. Each experiment is run 40 times and the results averaged over all runs. In each run, learning stops at 50,000 fitness evaluations. Each fitness evaluation consists of one execution of the ALSS simulator using the control parameters to be evaluated. When the ALSS simulation terminates, an appropriate fitness value is returned as described in section 9.3.3.

Figure 9.5 summarizes the results from these 120 experiments. For each experiment, we plot the fitness of the best solution found averaged over 40 runs along with 95% confidence intervals. The results are organized into three plots by optimization problem: (A) optimizing mission productivity, (B) optimizing mission duration, and (C) optimization of mission productivity and duration. The y -axes indicate the fitness of the solutions found: in units of science for (A), in simulation time steps or ticks for (B), and as a weighted combination

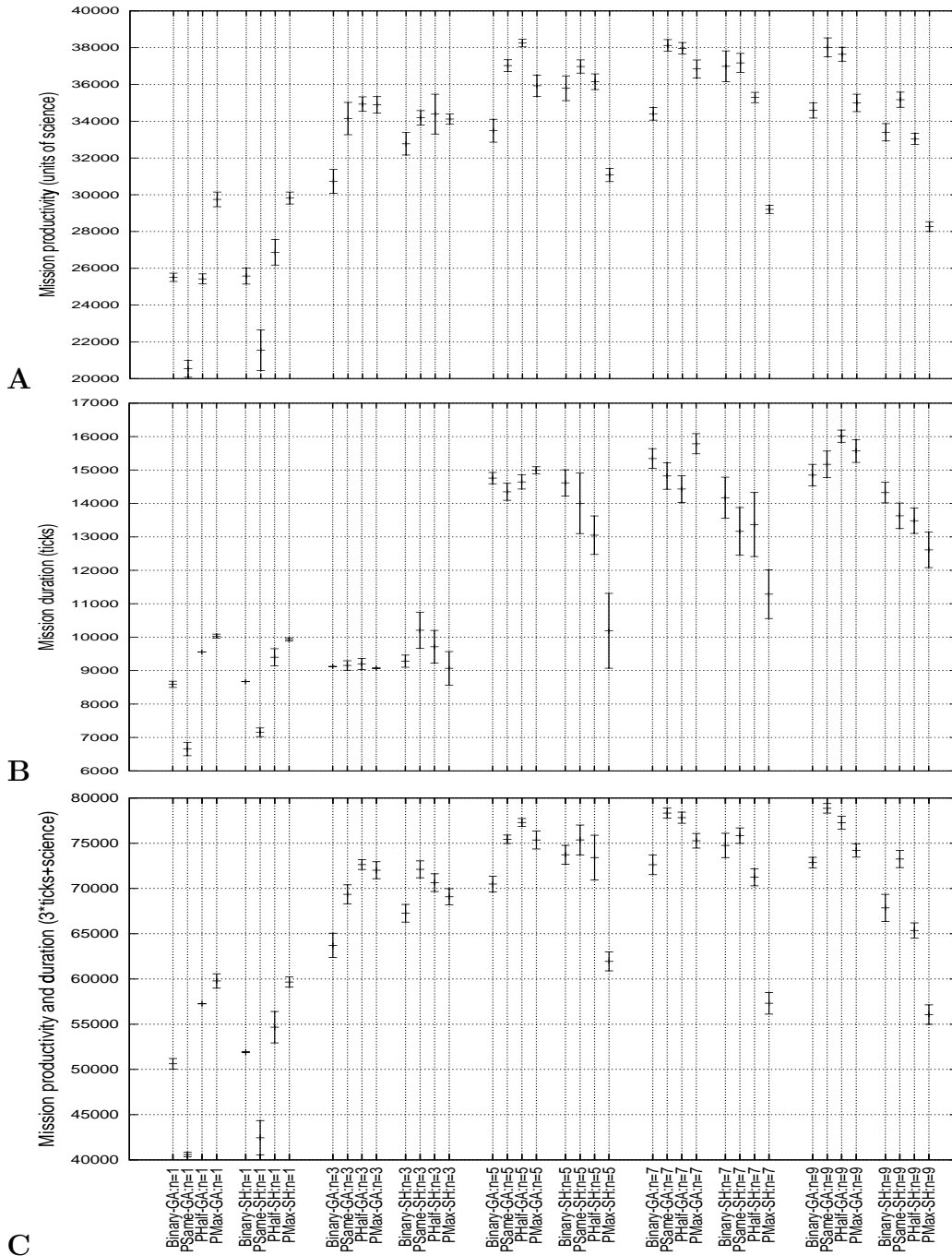


Figure 9.5: GA and SH with binary, proportional-same, proportional-half, and proportional-max representations using control strategies Γ_1 , Γ_3 , Γ_5 , Γ_7 , and Γ_9 to optimize ALSS (A) mission productivity, (B) mission duration, and (C) both. Fitness of best solution found averaged over 40 runs and 95% confidence intervals.

of both for (C). The x -axis indicates the algorithms used. On this axis, from left to right, we show a group of eight algorithms working with strategy $\Gamma_1(n = 1)$, then a group of eight algorithms working with strategy $\Gamma_3(n = 3)$, and so on until $\Gamma_9(n = 9)$. Within each group, the eight algorithms are, from left to right: *Binary-GA*, *PSame-GA*, *PHalf-GA*, *PMax-GA*, *Binary-SH*, *PSame-SH*, *PHalf-SH*, and *PMax-SH*.

Figure 9.5 (A) plots the results for optimizing mission productivity. Overall, performance increases as the number of control vectors n used by the control strategy Γ_n increases. For $n = 1$, performance ranges from an average of 20,535 units of science generated by *PSame-GA* to statistically undistinguishable averages of 29,822 and 29,748 units generated by *PMax-SH* and *PMax-GA*. For $n = 3$, performance fluctuates between 30,731 and 34,931 units of science generated by *Binary-GA* and *PHalf-GA*, respectively. *PMax-GA* performance of 34,894 is statistically undistinguishable from that of *PHalf-GA*. For $n = 5$, performance fluctuates between 31,076 and 38,251 units of science generated by *PMax-SH* and *PHalf-GA*. For the higher values of n , $n = 7$ and $n = 9$, performance ranges from 29,205 to 38,118 and from 28,267 to 38,016 with the worst and best performances in both cases generated by *PMax-SH* and *PSame-GA*, respectively. The best performance for this set of experiments (A) is 38,251 units of science, achieved by *PHalf-GA* with $n = 5$. *PSame-GA* with $n = 7$ and *PSame-GA* with $n = 9$ achieve results that are statistically equivalent to the best. The best performance obtained by a SH algorithm is of 37,173 by *PSame-SH* with $n = 7$. However close, it is, nevertheless, statistically worse than the best GA performance. Overall, for all the values of n in (A), the best performing algorithms belong to the group of GAs using proportional representations with the only exception for $n = 1$, where the performances of *PMax-SH* and *PMax-GA* are statistically equivalent.

Figure 9.5 (B) plots the results for optimizing mission duration. Again we see an increase in performance with increasing n , a trend that appears to be independent of the optimization function used. For $n = 1$, performance ranges from an average of 6,654 time steps by *PSame-GA* to an average of 10,030 time steps by *PMax-GA*. For $n = 3$, the range is an average of 9,067 time steps by *PMax-SH* to an average of 10,207 time steps by *PSame-SH*. For $n = 5$ and $n = 7$ performance ranges from 10,189 to 14,993 and from 11,287 to 15,786, respectively, with *PMax-SH* performing the worst and *PMax-GA* performing the best in both cases. For $n = 9$, performance ranges from 12,610 by *PMax-SH* to 16,012 by *PHalf-GA*. The best performance achieved in set (B) is an average of 16,012 simulation time steps by *PHalf-GA* in the $n = 9$ case, and *PMax-GA* with $n = 5$ performance of 15,786 is statistically indistinguishable from this best. The best performance obtained by a SH algorithm is an average of 14,615 achieved by *Binary-SH* with $n = 5$ which is significantly worse than the best GA performance. Overall, for all n in (B), the best performing algorithms are again GAs using proportional representations, with an exception in the case of $n = 3$. For $n = 3$ the best performing algorithm is an SH using proportional representation.

Figure 9.5 (C) plots the results for optimizing both mission productivity and duration. These results exhibit similar trends as seen in the previous plots. Performance increases with the increase of control vectors n used by control strategy Γ_n . Performance ranges are:

- for $n = 1$, 40,592 by *PSame-GA* to 59,781 by *PMax-GA*;
- for $n = 3$, 63,713 by *Binary-GA* to 72,641 by *PHalf-GA*;
- for $n = 5$, 61,945 by *PMax-SH* to 77,296 by *PHalf-GA*;
- for $n = 7$, 57,312 by *PMax-SH* to 78,626 by *PSame-GA*; and
- for $n = 9$, 56,062 by *PMax-SH* to 78,860 by *PSame-GA*.

The best performance obtained in this set (C) is 78,860 by *PSame-GA* with $n = 9$. The performance of 78,626 by *PSame-GA* with $n = 7$ is statistically indistinguishable from the best. The best performance obtained by a SH algorithm is 75,843 by *PSame-SH* with $n = 7$ which is significantly worse than the best GA performance. Overall, for any n in (C), the best performing algorithms are again GAs using proportional representations.

All of these experiments use the parameters settings show in Table 9.2; however, additional experiments have been performed with other mutation rates and population sizes to ensure that the qualitative nature of the results in this section is robust and not an artifact of a particular set of parameters.

9.6 Discussion

According to the NFL theorem [WM97], there is no universally best algorithm; however, there are algorithms that are best suited for particular problem classes. Given an algorithm, the choice of representation can have a significant impact on its effectiveness. Simpler representations may be inadequate in the amount of information that can be expressed [DM83]. More complex representations result in larger search spaces which can increase the difficulty of a problem and the computational cost of a learning algorithm [Sam94]. Different types of representations may have biases that make them more or less easily manipulated by a learning algorithm [CS92, LS87]. In this paper, we examine the performance of two ML algorithms using two different types of problem representations applied to the optimization of an ALSS.

9.6.1 Algorithms: GA vs. SH

In comparing all of the GA and SH variants that we test, we find that GAs tend to have better overall performance than SHs for the ALSS problem. GAs generate the best performance in all groups (as defined by optimization objective and control strategy) with only once exception: optimizing mission duration with $n = 3$. GA performance consistently increases with increasing n . SH becomes less competitive with GAs as n increases and the worst performance is consistently and noticeably found using *PMax-SH*. GA performance remains

relatively stable as the problem size scales up while SH shows noticeable decline as problem size grows. We attribute the weaknesses of SH to the simplicity of the SH search strategy – move in single steps only toward higher fitness points – which can easily become trapped at local optima. In addition, the SH algorithm we selected does not allow random walks among solutions of equivalent fitness which can further exacerbate the problem of becoming trapped (we plan to eliminate this restriction in future experiments). The more complex operators and population based approach of a GA appear to provide a better search strategy for this type of problem. Nevertheless, the simple SH search strategy is still a competitive strategy for an ALSS as it is usually only outperformed by a small, albeit statistically significant, margin.

9.6.2 Control Strategies: $n = 1, 3, 5, 7, 9$

The plots from Figure 9.5 show a clear trend: performance improves as the number of control vectors, n , used in the optimization increases. This result remains fairly robust throughout our experiments and supports and extends previous work [KBS01] where this same trend is reported for a simple GA. We attribute this behavior to the fact that more control vectors allows for a more subtle control of the ALSS, leading to noticeably improved performance. There is, however, a trade-off between the better control allowed by a high n and the consequent increase in the search space size. Increasing n increases the search space exponentially. As a result, we expect to see a performance drop at high values of n due to searchability issues. We do not observe such expected behavior for high n which we speculate is due to the fact that we work with relatively low values of n (from 1 to 9). We do observe, however, that after $n = 5$, increases of n yield diminishing performance returns.

9.6.3 Representations: Binary vs. Proportional

The proportional representation is inspired by the natural process of gene expression and the concept that, within a genome, what is most important is whether or not the necessary genes exist. Of lesser impact is the order in which existing genes are arranged. This concept led to the development of a content-based representation for GAs instead of the traditional order-based representation. Previous investigations on the impact of proportional representation on GAs have shown advantages to using a proportional representation over traditional binary representation on several simple problems: number matching, resource allocation, and symbolic regression [WG02]. In this paper, we extend those results by showing that this advantage persists and is even more noticeable on a significantly more complex problem, control of a coupled dynamical system. The best performing algorithms for each set of experiments are consistently proportional algorithms. It is important to note is that coupled dynamical

systems are very sensitive to initial conditions. We suspect that the automatic adjustment of resolution in the proportional representation and its ability to allocate genomic resources in response to fitness payoffs have an important role in the suitability of this representation this problem. This is clearly an area for further study.

Despite the close analogy to genetics, there is no a priori rationale that restricts the proportional representation to be used only with evolutionary algorithms. We implement and test this same representation in a SH. Results indicate that, in contrast with the performance increase seen in a proportional GA, there is not consistent performance improvement of a proportional SH over a binary SH. The performance of a proportional SH is only comparable to the performance of traditional binary SH. We believe that a likely reason for this difference is that the SH algorithm that we selected only moves to points of greater fitness in the search space. As a result, it does not allow random walks on neutral networks of points with equal fitness. Proportional representation is known to have a high redundancy which forms neutral networks in the search space that arguably can improve the search abilities of algorithms; but for this to be beneficial, an algorithm would have to allow random walks. We plan to re-examine SH performance while allowing random walks. We should also note that, while SH uses an operator analogous to mutation to move within a search space, SH lacks an operator analogous to GA recombination.

9.6.4 Resolution: P-Same vs. P-Half vs. P-Max

In Figure 9.5, the x -axis orders the tested algorithms into groups of four: the *Binary*, *PSame*, *PHalf*, and *PMax* representations. Consecutive foursomes indicate GA and SH algorithms for the five control strategies: $n = 1, 3, 5, 7, 9$.

We previously noted that, overall, the best performing algorithms tend to be GAs using a proportional representation. Closer examination of the best performing proportional algorithm within each foursome suggests that the value of n can affect the type of proportional representation that performs best. For low n ($n = 1$) the best performing algorithms all use the *PMax* representation. For high n ($n = 7, 9$) the best performing algorithms tend to use the *PSame* representation. For middle values of n ($n = 3, 5$) the best performing algorithms tend to use *PHalf* or *PMax* for a GA and *PSame* or *PHalf* for an SH. We believe that this variation is due to the resolution-searchability trade-off described in section 9.6.2: low resolution limits the quality of solutions, but reduces the search space allowing solutions to be found faster; high resolution increases solution quality at the cost of increasing the size of the space to search. Low n generates a small search space on which there is advantage to having the maximum possible resolution, *PMax*, in the representation. High n results in large search spaces, making the limited resolution and relatively smaller search space of *PSame* more attractive because of its increased searchability.

Table 9.5: Summary of experimental results of comparing GA with binary and proportional representations to optimize the ALSS simulator in 45 different configurations..

| | Comparing Proportional-GA and Binary-GA (number of experiments; each experiment = 40 runs) | | |
|--------|--|------------------------------|--|
| | Proportional-GA outperforms Binary-GA | No statistical difference | Proportional-GA performs worse than Binary-GA |
| P-Same | 8(53%) | 4(27%) | 3(20%) |
| P-Half | 11(73%) | 3(20%) | 1(7%) |
| P-Max | 11(73%) | 4(27%) | 0(0%) |
| Total | 30(67%) | 11(24%) | 4(9%) |

9.6.5 Performance: Comparing Binary and Proportional Representations

In order to compare the relative performance of proportional and binary representations, we calculate, for all the experiments reported in Figure 9.5, the number of times that an algorithm using a proportional representation is statistically better, comparable, or worse than the same algorithm using a binary representation. Table 9.5 presents our results for the GA and Table 9.6 for the SH.

In Table 9.5, *PSame-GA* uses the same genomic length as *Binary-GA* which, according to equation 9.1, puts *PSame-GA* at sharp disadvantage. Despite this limitation, *PSame-GA* performs better than *Binary-GA* on 53% of the experiments and worse on 20%. *PHalf-GA* uses a longer genome than *Binary-GA*, but is still theoretically at a disadvantage because it has only *half* of the genomic length required by equation 9.1 to ensure comparable resolutions. Despite this theoretical limitation, *PHalf-GA* performs better than *Binary-GA* on 73% of the experiments, and worse on only 7% of the experiments. *PMax-GA* uses a variable length genome with the maximum length equal to the required genomic length given by equation 9.1. As a result, *PMax-GA* and *Binary-GA* are comparable in terms of available resolution. *PMax-GA* perform better than *Binary-GA* 73% of the time and never performs worse than the Binary-GA. Overall, for all of the cases compared in Table 9.5, proportional GAs perform better than binary GAs on 67% of the experiments and worse on 9% of the experiments.

In Table 9.6, we observe a very different situation for the SH algorithms. The *PSame-SH* representation outperforms *Binary-SH* in 40% of the experiments but performs worse in 20%. The situation becomes worse as we increase the resolution of the proportional representation with 33% better and 27% worse for *PHalf-SH*, and 33% better and 60% worse for *PMax-SH*. There is a clear preference for compact representations over high resolutions. Overall, the proportional SHs perform better than binary SHs on 35.5% of the experiments, worse on 35.5% of experiments, and show no statistical difference on 29% of experiments. As a

Table 9.6: Summary of experimental results of comparing Proportional-SH and Binary-SH to optimize the ALSS simulator in 45 different configurations.

| | Comparing Proportional-SH and Binary-SH (number of experiments; each experiment = 40 runs) | | |
|--------|--|---------------------------|---|
| | Proportional-SH outperforms Binary-SH | No statistical difference | Proportional-SH performs worst than Binary-SH |
| P-Same | 6(40%) | 6(40%) | 3(20%) |
| P-Half | 5(33%) | 6(40%) | 4(27%) |
| P-Max | 5(33%) | 1(7%) | 9(60%) |
| Total | 16(35.5%) | 13(29%) | 16(35.5%) |

result, there does not appear to be a clear advantage to using either binary or proportional representation in an SH.

9.7 Summary

Advanced Life Support Systems are a crucial component for successful space exploration. An ALSS is a coupled dynamical system whose overall behavior is difficult to predict and whose behavior is highly sensitive to initial conditions and control parameters. The unpredictability of such systems limits the effectiveness of deliberative and hand-crafted control algorithms. We investigate the performance of two well known ML techniques, genetic algorithms and stochastic hill-climbing, on the problem of learning how to control an ALSS.

The selection of representation can have a significant impact on the effectiveness and efficiency of a learning algorithm. The same problem encoded with two different representations can appear to an ML algorithm to be two entirely different problems [MW94]. We compare the performance of a GA and SH using two different types of representations: a traditional binary representation and a novel proportional representation. The proportional representation, originally developed for GAs, is a content-based representation derived from the concept of gene expression. Its ability to dynamically adapt the distribution of genomic resource along with a solution allows it to naturally evolve both parameters values and their appropriate resolutions. We hypothesize that this unique ability may make the proportional representation particularly suitable for the highly sensitive nature of coupled dynamical systems.

We perform experiments on three ALSS optimization problems using five control strategies. For each of these experiments, we compare the performance of a GA and SH, each using a binary representation and three variations of proportional representations: *PMax* which has an encoding resolution equal to that of the binary representation, *PHalf* which has an

encoding resolution that is one half of *PMax*, and *PSame* which has a severely penalized encoding resolution as compared to the binary representation.

Throughout our experiments, we observe that the GA consistently performs as well or better than the SH. The top performance is achieved by a GA variant in all the fifteen experiments, with only one exception: optimizing mission duration with strategy Γ_3 . In fact, all of the best performers use the proportional representation, supporting our hypothesis that a proportional representation is competitive. For lower values of n , *PMax-GA* appears to be the strongest algorithm, and for higher values of n , *PSame-GA* appears to be the strongest algorithm. We attribute the differences to the trade-off between the size of the search space and the resolution of the representation. For lower n , the search space is small enough to afford a full resolution of parameters, while for high n and its large search space, preference is given to limited resolution representations to favor searchability over resolution.

We observe that all algorithms show increasing performance as the number of vectors, n , in the control strategy Γ_n increases. We attribute this behavior to the fact that more control vectors are likely to allow a more precise control of the system. This result extends previous results obtained for a simple GA to the eight algorithm-representation combinations tested in this paper.

Finally, we compare the relative performance of the three proportional representations variants with binary representation for each algorithm. When using a GA, all of the proportional representation variants tend to be more successful than binary representation. On average, the proportional representation GAs outperform the binary GA 67% of the time and perform worse than the binary GA 9% of the time. The *PMax-GA* representation appears to be strongest, outperforming *Binary-GA* 73% of the time; exhibiting statistically equivalent performance the remaining 27% of the time. *PMax-GA* is never outperformed by *Binary-GA*. Comparison of proportional and binary representations for the SH yields a very different picture. On average the proportional SHs outperform the binary SH 35% of the time, and they are outperformed by the same percentage. As a result, while proportional representation yields a strong improvement in performance for a GA, it does not significantly affect the performance of the SH algorithms.

CHAPTER 10

FUTURE WORK

This thesis begins by proposing to study whether self-organization of protein-analogous structures at the representation level can increase the degree of complexity and novelty of the solutions obtainable using evolutionary search techniques. Then it identifies two fundamental aspects of this proteomics approach to EC: (1) proteins interact in multisets, and (2) proteins are functional structures. This thesis analyzes the first one. There are many directions of future work on the second. Three of them are listed below.

10.1 Self-organization in Representations

One area for future research is to explore domain-independent methods for representing complex solutions with relatively simple self-organizable building blocks. Self-organization is present in almost every level of natural evolution: gene regulation networks, protein interaction networks, metabolic pathways, cellular organization, etc, but it is not usually present in evolutionary algorithms. Perhaps the self-organization of genotypic instructions into phenotypes is a key ingredient to unleashing the evolution of complex and scalable solutions with desirable emergent phenomena such: scale-free, adaptability, innovativeness, evolvability, and robustness.

10.2 Proteins as Representational Building Blocks

Another area of future search is whether or not new recent discoveries on protein function has something to teach us about functional building block design for EC problem representation.

To design of representations that combine simple functional building blocks into increasingly complex solutions, basic understanding of the required characteristics of these building blocks and their interactions is required. Some relevant questions include: How many building blocks should be produced? How do they function? How do they interact? There is a clear need for methods of design and evaluation for self-organizable, functional building blocks for problem representation. The problem of designing basic building blocks that self-reproduce, self-assemble, and self-organize into increasingly complex functionalities has already been solved by nature. DNA is the information storage structure, proteins are

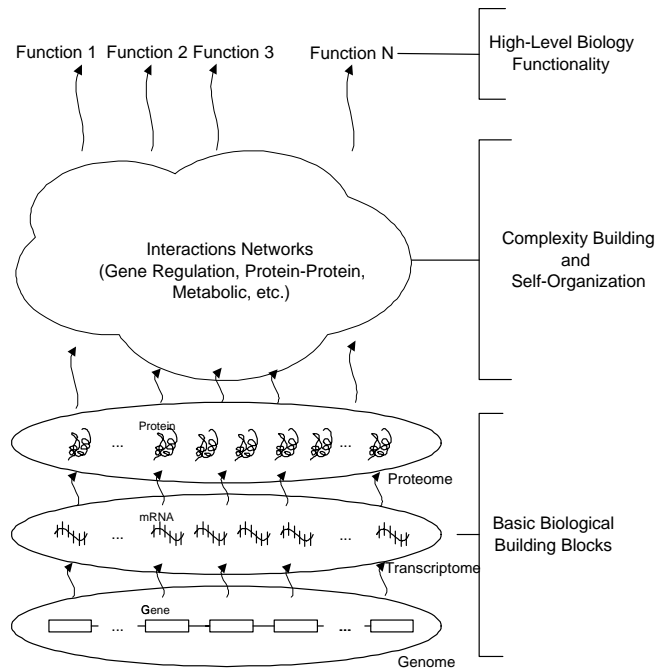


Figure 10.1: Genes, mRNA, and proteins as the basic building blocks of biological function. These basic components interact at many levels. Gene regulation networks, protein-protein interaction networks, and metabolic networks among others describe their interactions. From these interactions, complex high-level biological functionality emerge. Recently, it has been shown that these basic biological components self-organize into scale-free hierarchical networks with embedded modularity. It has also been shown that these type of networks are robust and error tolerant. These results seem to suggest that scale-free hierarchical networks with embedded modularity are a potentially good architecture for combining building blocks into high-level functionality in general.

the basic building blocks of biological functionality, and the universal genetic code defines the mapping between them. Beyond the protein level, the problem becomes increasingly complex and less well understood. At this level, we start talking about gene regulatory networks and, in a wider sense, about metabolic pathways. Proteins cooperate, compete, and antagonize in order to ultimately produce complex high-level functions. Proteins can self-organize into bigger molecular machinery and can trigger the production of more proteins on demand. In isolation, proteins implement to a great extent the basic functionality of their corresponding genes and, in conjunction with other proteins, they contribute as basic building blocks to the construction of an organism's increasingly complex functionalities (see Figure 10.1).

10.3 Building Blocks Interactions

A third area for future research focuses on how representational building blocks should interact. The relatively recent availability of actual genomic sequences has opened a new perspective: the study of a biological organism as a whole. Genomics and proteomics are dedicated not only to produce but also to analyze all of this newly obtained data. New approaches for unraveling large-scale biological organization are emerging thanks to this wealth of information. One of the simplest approaches charts interaction graphs from the available data and topologically analyzes these graphs to discover hidden properties or underlying organizational principles that could potentially improve understanding of these complex systems. The nodes of these interaction graphs (or interaction networks) are typically basic biological building blocks (genes, mRNAs, proteins, and metabolites) and the edges represent functional relationships. For example, gene regulation networks relate genes with the proteins that promote their expression; protein-protein interaction networks chart functionally related proteins; and metabolic networks relate the reactant and product substrates of metabolic reactions. The information provided by these networks is at the organism level and its analysis provides information about the large-scale organization of the interactions among basic biological building blocks. Recent work has found that these networks of interaction share many properties such as presence of recurrent patterns, scale-free (*preferential attachment*), and hierarchically modular topological organization, as shown in Table 10.3. This suggests that the underlying principles that govern functional building block interaction in nature could be used to engineer competent systems to combine functional building blocks for EC representations.

Table 10.1: Summary of recent results related to biological building block organization, over 43 different organisms representing all three domains of life.

| Interaction Properties of Biological Building Blocks | |
|--|---|
| Property | Further information |
| <p>Recurrent patterns: <i>basic network building blocks that frequently reappear, network motifs.</i> [OB02]</p> | <p>Autoregulation, multicomponent loop, feedforward loop, single and multi input motifs, regulator chain [Lee02, Mil02, SMM02].</p> |
| <p>Scale-free: <i>connectivity that follows the power law</i> [Pri76] <i>with preferential attachment property.</i> [BA99]</p> | <p>Small-world [WF02], and scale-free [BA99, JTA00, JBT01, Wuc02].</p> |
| <p>Hierarchical modularity: <i>building blocks are organized hierarchically into functional modules.</i> [HHL99]</p> | <p>Hierarchical organization [OB02, Rav02], modular organization [HHL99, BDR03, CD02], and robustness [AJB00, Jeo01].</p> |

CHAPTER 11

CONCLUSION

As the complexity of our society and computational resources increases, so does the complexity of the problems that we approach using evolutionary search techniques. There are recent approaches to deal with the problem of scaling evolutionary methods to cope with highly complex difficult problems. Many of these approaches are biologically inspired and share an underlying principle: a problem representation based on basic representational building blocks that interact and self-organize into complex functions or designs. The observation from the central dogma of molecular biology that proteins are the basic building blocks of life and the recent advances in proteomics on analysis of the structure, function and interaction of entire protein complements, lead us to propose a unifying framework of thought for these approaches: the proteomics approach.

This thesis identifies two fundamental aspects of transition from traditional approaches to this new approach: (1) proteins interact in a three dimensional medium analogous to a multiset; and (2) proteins are functional structures. The first aspect is foundational for the study of the second. This thesis provides an analysis of the first aspect. In carrying out this analysis, we observe that there is a fundamental departure from typical EC methods: using a multiset of proteins as an intermediate mapping results in a completely location independent problem representation where the location of the genes in the genome has no effect on the fitness of the solutions. These are proteome-based location independent representations.

This thesis gives a formal analysis of the proteome-based location independent genotype to phenotype mapping. Using combinatorics, we provide close form expression for the relationship between individual lengths and alphabet sizes that will result in comparable solution spaces. Proteome-based location independent representations require significantly larger alphabets of symbols in comparison with traditional representations to encode the same amount of information. This is expected as we encode information in the symbols of the alphabet instead of in the location of these symbols. In a way, this is the price to pay for complete location independence. This thesis also analyzes the neutral classes formed by this representation. A neutral class, for this representation, includes all the possible permutations of the genes in an individual genome. It is obvious that these neutral classes can grow very easily. Again, this is the price to pay for truly complete location independence.

Contrary to what we would expect from these theoretical limitations, we provide a significant amount of experimental evidence that proteome-based location independent representations in a simple GA are, in many cases, better than or, at worst, comparable to a traditional GA using a binary representation. For all experiments, we use a simple genetic algorithm with a proteome-based location independent representation that we call the pro-

portional genetic algorithm. The question then is: what causes the proteome-based location independent representation to perform well?

This thesis provides a possible explanation. Among the causes that make these representations effective, it singles out three. First, these representations accentuate the key property required to perform stochastic search. Second, once genomic self-similarity has emerged, the schemata disruption cause by crossover is minimal. Finally, by definition, these representations are not subject to negative effects associated with the arrangement of the genes on a genome such positional bias.

Proteome-based location independent genomes are free to self-organize because there is no selective pressure for any particular genomic ordering. These genomes self-organize into structures resembling white noise that are self-similar with respect to fitness. We start this thesis by pointing out that stochastic search techniques, such EC, rely on a key property that is common to many difficult problems: there is a positive correlation between the form and quality of candidate solutions. The proteome-based location independent representation appears to accentuate this property in the following way. First, crossover becomes an “averaging” operator. Because of the self-similarity property, each segment being crossed over is a valid representative of the fitness of the entire individual. As information is encoded in protein concentrations, it does not matter where in an individual the segment being crossed over ends up, it will always have the same effect: offspring are essentially weighted averages of their parents. The weights are directly proportional to the sizes of the segments. Second, the effect of mutation is also guaranteed to be smooth. The effect of mutating a single symbol is to subtract one unit from one protein concentration and add one unit to another. In traditional representations the crossover operator disrupts a building block or a schema if the crossover point splits the schema. In self-similar genomes, when crossover split a schema, the resulting halves are valid representatives of the original one. As a result, once the self-similarity with respect to fitness has emerged, disruptions of schema caused by crossover are minimal.

This thesis concludes that the use of protein analogous structures as an intermediate representation in evolutionary computation is not only feasible but in some cases advantageous. Finally, it provides a foundation for further research on proteins as functional self-organizing structures capable of building increasingly complex functionality, and as basic units of problem representation for evolutionary computation.

11.0.1 Limitations

Completely location independent representations also introduce new problems of their own such as the need for large alphabets of symbols and the theoretical need for larger representation spaces than traditional approaches. The need of large alphabets can be overcome partially by introducing order oriented encoding for the alphabet symbols themselves. For instance an alphabet of 65,536 symbols can be defined by using 8-symbol strings from an

auxiliary alphabet of size 4. The problem of larger representations spaces is more fundamental. This problem is linked to the definition of complete location independence itself and is a limitation for this type of representational approaches. The larger a string of genes, the larger the number of strings that will contain the same information on different orderings. In order to enforce complete location independence, all these strings need to map to the same neutral class. The growth of these neutral classes triggers the growth of the representation space. As a result, algorithms using complete location independent approaches require more computational space than traditional algorithms for the same problem complexity. Therefore, if space is a concern the use of traditional approaches is recommended.

11.0.2 Contributions

This thesis contributes to the field of Evolutionary Computation in the following ways:

- It proposes a new approach to EC based on the central dogma of molecular biology. There are many approaches to scale EC to high complexities based on biologically inspired principles such as embryogenesis and developmental biology. The proteomic approach, however, captures and makes explicit the underlying principle of many of these representational approaches: *basing problem representation on basic building blocks that interact and self-organize into complex functions or designs*. This thesis provides the first comprehensive study of one of the basic aspects of this approach: proteome-based location independent representations.
- It defines an empirical framework with which to study proteome-based location independent representations using a genetic algorithm, the proportional genetic algorithm. The PGA proteome-based location independent representation, by definition, does not suffer from traditional EC hurdles associated with the arrangement of the genes or positional effect in a genome. This improvement is possible simply because proteome-based representations are based on “genomic content” whereas traditional representations are based on “genomic order”. Eliminating these hurdles delivers an algorithm that is less sensitive to implementation decisions and therefore more robust. Such a representation has the ability to self-organize into a genomic structure that appears to favor positive correlations between the form and quality of represented solutions. Additionally, it provides experimental evidence of some positive effects such as automatic adjustment of genomic resources and formation of self-similar building blocks.
- It provides theoretical and empirical evidence that a protein based representation is feasible and effective for practical problems. It shows theoretically the relationship between comparable genotypic and phenotypic space sizes needed for a fair comparison and provides substantial empirical evidence (380 million fitness evaluations over various problems and parameters settings) of the competence of this approach.

- It presents a case study for the proteomic approach to EC on a nontrivial and difficult real world problem: optimizing control of Advance Life Support Systems (ALSS), part of the NASA BioPlex project. This case study illustrates the effects of this approach on evolutionary algorithms and compares it to the effects of this approach on hill-climbing algorithms. This thesis gives a formalization of this problem, its optimization objectives, and its strategies in a framework given in Appendix A.

APPENDIX A
ADVANCED LIFE SUPPORT SYSTEM
SIMULATOR

The simulator used in this paper is the Advanced Life Support System simulator (ALSS), based on the NASA BioPlex simulator [Tri99]. The ALSS is a coupled dynamical system. The behaviors of its components or modules are completely deterministic and can be characterized by the state of the variables in any time step of a simulation. Transition equations determine the next simulation step for any given set of these state variables. Nevertheless, the behavior of the entire system of coupled modules is difficult to predict. The following is a mathematical description of the ALSS simulator used in this paper.

A.1 ALSS Simulation Mathematical Description

Definition 1 (ALSS) Formally an ALSS simulation \mathcal{A} is denoted by a 4-tuple:

$$\mathcal{A} = \langle \mathcal{Q}, \vec{q}_0, \mathcal{C}, \delta_{\mathcal{C}} \rangle$$

where:

\mathcal{Q} is the set of simulation states,

\vec{q}_0 in \mathcal{Q} is the initial state of the simulation,

\mathcal{C} is the control strategy that defines a vector of control parameters for each simulation time step,

$\delta_{\mathcal{C}}$ finite set of transition equations ¹

Definition 2 (Simulation Variables) Let \mathcal{V} denote the set of all the variables associated with the simulation \mathcal{A} , then:

$$\mathcal{V} \stackrel{\text{def}}{=} \{ \vec{\mathbf{e}}, \vec{\mathbf{r}}, \vec{\mathbf{o}}, \vec{\mathbf{a}}, \vec{\mathbf{w}} \}$$

where:

$\vec{\mathbf{e}} \stackrel{\text{def}}{=} \langle e_{\text{energy}}, e_{\text{air.o2}}, e_{\text{air.co2}}, e_{\text{water.clean}}, e_{\text{water.dirty}}, e_{\text{food}}, e_{\text{science}}, e_{\text{store.water}}, e_{\text{store.food}}, e_{\text{store.o2}}, e_{\text{store.co2}} \rangle$
, vector of **environmental** variables,

$\vec{\mathbf{r}} \stackrel{\text{def}}{=} \langle r_{\text{status}}, r_{\text{food}}, r_{\text{water}}, r_{\text{air.o2}}, r_{\text{starvation}}, r_{\text{dehydration}}, r_{\text{asphyxiation}} \rangle$, vector of **crew** variables,

$\vec{\mathbf{o}} \stackrel{\text{def}}{=} \langle o_{\text{status}}, o_{\text{energy}}, o_{\text{water}}, o_{\text{low.energy}}, o_{\text{low.water}}, o_{\text{age}}, o_{\text{biomass}}, o_{\text{o2.released}}, o_{\text{co2.absorbed}} \rangle$,
vector of **crop** variables,

$\vec{\mathbf{a}} \stackrel{\text{def}}{=} \langle a_{\text{energy}}, a_{\text{low.energy}}, a_{\text{recovery.time}} \rangle$, vector of **air recovery** variables,

¹Due to space constraints, we do not include the water, air, and food store equations in this mathematical description, nor the biomass related equations. For a complete mathematical description of the ALSS simulator used in this paper please refer to [GW03].

$\vec{w} \stackrel{\text{def}}{=} \langle w_{in}, w_{low.energy}, w_{uptime}, w_{potable}, w_{energy} \rangle$, vector of **water recovery** variables,

Definition 3 (Simulation State) A simulation state, \vec{q} in \mathcal{Q} , is a particular assignment of values for all the simulation the variables in \mathcal{V} . The simulation state at time t is:

$$\vec{q}_t \stackrel{\text{def}}{=} \langle \vec{e}_t, \vec{r}_t, \vec{o}_t, \vec{a}_t, \vec{w}_t \rangle$$

Definition 4 (Initial State) The initial simulation state denoted by \vec{q}_0 in \mathcal{Q} , is a defined as follows:

$$\vec{q}_0 \stackrel{\text{def}}{=} \langle \vec{e}_0, \vec{r}_0, \vec{o}_0, \vec{a}_0, \vec{w}_0 \rangle$$

where:

$$\vec{e}_0 \stackrel{\text{def}}{=} \langle e_{energy} = 10000, e_{air.o2} = 3000 \times 0.2095, e_{air.co2} = 3000 \times 0.0003, e_{water.clean} = 10, e_{water.dirty} = 0, e_{food} = 4, e_{science} = 0, e_{store.water} = 500, e_{store.food} = 500, e_{store.o2} = 50000, e_{store.co2} = 50000, \rangle,$$

$$\vec{r}_0 \stackrel{\text{def}}{=} \langle r_{status} = 1, r_{food} = 0, r_{water} = 0, r_{air.o2} = 0, r_{starvation} = 0, r_{dehydration} = 0, r_{asphyxiation} = 0 \rangle,$$

$$\vec{o}_0 \stackrel{\text{def}}{=} \langle o_{status} = 1, o_{energy} = 0, o_{water} = 0, o_{low.energy} = 0, o_{low.water} = 0, o_{age} = 0, o_{biomass} = 0, o_{o2.released} = 0, o_{co2.absorved}=0 \rangle,$$

$$\vec{a}_0 \stackrel{\text{def}}{=} \langle a_{energy} = 0, a_{low.energy} = 0, a_{recovery.time} = 0 \rangle,$$

$$\vec{w}_0 \stackrel{\text{def}}{=} \langle w_{in} = 0, w_{low.energy} = 0, w_{uptime} = 10, w_{potable} = 0, w_{energy} = 0 \rangle,$$

Definition 5 (Control Strategy) Let \mathcal{C} denote a control strategy and be defined as follows:

$$\mathcal{C} \stackrel{\text{def}}{=} \left\{ \vec{c}^t \mid (0 \leq t \leq T_{max}) \bigwedge t \in \mathbb{N} \right\}$$

where:

$$\vec{c}^t \stackrel{\text{def}}{=} \langle c_{energy.to.air}^t, c_{energy.to.water}^t, c_{energy.to.food}^t, c_{water.to.crew}^t, c_{water.to.crops}^t, c_{activity.level}^t \rangle$$

is the vector of **control** parameters values for time t ,

$\mathcal{C} \in \mathbf{C}$, \mathbf{C} is the set of all possible control strategies.

Definition 6 (Transition Equations) The set of transition equations denoted by $\delta_{\mathcal{C}}$, defines how to obtain the next simulation state ($t + 1$) from the current state (t) for a given control strategy \mathcal{C} . The simulation, \mathcal{A} , is a coupled dynamical system with four components: crew, crops, air revitalization, and water revitalization. Therefore we have four sets of equations:

$$\delta_{\mathcal{C}} \stackrel{\text{def}}{=} \langle \delta_{\mathcal{C}_{crew}}, \delta_{\mathcal{C}_{crops}}, \delta_{\mathcal{C}_{airRevitalization}}, \delta_{\mathcal{C}_{waterRevitalization}} \rangle$$

where definitions for $\delta_{\mathcal{C}_{crew}}$, $\delta_{\mathcal{C}_{crops}}$, $\delta_{\mathcal{C}_{airRevitalization}}$, $\delta_{\mathcal{C}_{waterRevitalization}}$ are given below.

Definition 7 (Crew Equations) $\delta_{C_{crew}}$:

$$(\mathbf{r}_{status})_{t+1} = \begin{cases} 0 & \begin{aligned} & \text{if } [(\mathbf{r}_{status})_t = 0] \\ & \vee [(\mathbf{r}_{starvation})_t > \mathbf{R}_{F.Del}] \\ & \vee [(\mathbf{r}_{dehydration})_t > \mathbf{R}_{W.Del}] \\ & \vee [(\mathbf{r}_{asphyxiation})_t > \mathbf{R}_{A.Del}], \end{aligned} \\ (\mathbf{r}_{status})_t + \begin{aligned} & -0.01 \times (\mathbf{r}_{status})_t \times [|(\mathbf{r}_{food})_t| < 0.25 \times \mathbf{R}_{F.Req}] \\ & -0.02 \times (\mathbf{r}_{status})_t \times [|(\mathbf{r}_{water})_t| < 0.25 \times \mathbf{R}_{W.Req}] \\ & -0.1 \times (\mathbf{r}_{status})_t \times [|(\mathbf{r}_{air.o2})_t| < 0.25 \times \mathbf{R}_{A.Req}] \end{aligned} & \text{otherwise.} \end{cases}$$

$$(\mathbf{r}_{starvation})_{t+1} = \begin{cases} (\mathbf{r}_{starvation})_t + 1 & \text{if } [(\mathbf{r}_{food})_t < 0.25 \times \mathbf{R}_{F.Req}], \\ 0 & \text{otherwise.} \end{cases}$$

$$(\mathbf{r}_{dehydration})_{t+1} = \begin{cases} (\mathbf{r}_{dehydration})_t + 1 & \text{if } [(\mathbf{r}_{water})_t < 0.25 \times \mathbf{R}_{W.Req}], \\ 0 & \text{otherwise.} \end{cases}$$

$$(\mathbf{r}_{asphyxiation})_{t+1} = \begin{cases} (\mathbf{r}_{asphyxiation})_t + 1 & \text{if } [(\mathbf{e}_{air.o2})_t < 0.25 \times \mathbf{R}_{A.Req}], \\ 0 & \text{otherwise.} \end{cases}$$

$$(\mathbf{e}_{air.o2})_{t+1} = \begin{cases} (\mathbf{e}_{air.o2})_t - 537.6/24 & \text{if } [c_{activity.level}^t = SLEEP], \\ (\mathbf{e}_{air.o2})_t - 832.8/24 & \text{if } [c_{activity.level}^t = LOW], \\ (\mathbf{e}_{air.o2})_t - 1200/24 & \text{if } [c_{activity.level}^t = MED], \\ (\mathbf{e}_{air.o2})_t - 1600.8/24 & \text{if } [c_{activity.level}^t = HIGH]. \end{cases}$$

$$(\mathbf{e}_{air.co2})_{t+1} = \begin{cases} (\mathbf{e}_{air.co2})_t + 643.2/24 & \text{if } [c_{activity.level}^t = SLEEP], \\ (\mathbf{e}_{air.co2})_t + 996.0/24 & \text{if } [c_{activity.level}^t = LOW], \\ (\mathbf{e}_{air.co2})_t + 1433.7/24 & \text{if } [c_{activity.level}^t = MED], \\ (\mathbf{e}_{air.co2})_t + 1910.4/24 & \text{if } [c_{activity.level}^t = HIGH]. \end{cases}$$

$$(\mathbf{e}_{water.clean})_{t+1} = \begin{cases} (\mathbf{e}_{water.clean})_t - 0.75 \times (\mathbf{r}_{water})_t & \text{if } [c_{activity.level}^t = LOW], \\ (\mathbf{e}_{water.clean})_t - 0.85 \times (\mathbf{r}_{water})_t & \text{if } [c_{activity.level}^t = MED], \\ (\mathbf{e}_{water.clean})_t - 0.95 \times (\mathbf{r}_{water})_t & \text{if } [c_{activity.level}^t = HIGH], \\ (\mathbf{e}_{water.clean})_t & \text{otherwise.} \end{cases}$$

$$(\mathbf{e}_{water.dirty})_{t+1} = \begin{cases} (\mathbf{e}_{water.dirty})_t + 0.75 \times (\mathbf{r}_{water})_t & \text{if } [c_{activity.level}^t = LOW], \\ (\mathbf{e}_{water.dirty})_t + 0.85 \times (\mathbf{r}_{water})_t & \text{if } [c_{activity.level}^t = MED], \\ (\mathbf{e}_{water.dirty})_t + 0.95 \times (\mathbf{r}_{water})_t & \text{if } [c_{activity.level}^t = HIGH], \\ (\mathbf{e}_{water.dirty})_t & \text{otherwise.} \end{cases}$$

$$(\mathbf{e}_{food})_{t+1} = \begin{cases} (\mathbf{e}_{food})_t - 0.75 \times (\mathbf{r}_{food})_t & \text{if } [c_{activity.level}^t = LOW], \\ (\mathbf{e}_{food})_t - 0.85 \times (\mathbf{r}_{food})_t & \text{if } [c_{activity.level}^t = MED], \\ (\mathbf{e}_{food})_t - 0.95 \times (\mathbf{r}_{food})_t & \text{if } [c_{activity.level}^t = HIGH], \\ (\mathbf{e}_{food})_t & \text{otherwise.} \end{cases}$$

$$(e_{science})_{t+1} = \begin{cases} (e_{science})_t + 0.75 \times R_{Number} \times R_{SciFact} \times (r_{status})_t & \text{if } [c_{activity.level}^t = LOW], \\ (e_{science})_t + 0.85 \times R_{Number} \times R_{SciFact} \times (r_{status})_t & \text{if } [c_{activity.level}^t = MED], \\ (e_{science})_t + 0.95 \times R_{Number} \times R_{SciFact} \times (r_{status})_t & \text{if } [c_{activity.level}^t = HIGH], \\ (e_{science})_t & \text{otherwise.} \end{cases}$$

Where:

$$(r_{water})_t = \min[c_{water.to.crew}^t, (e_{water.clean})_t]$$

$$(r_{food})_t = \min[R_{F.Req}, (e_{food})_t]$$

$$(r_{air.o2})_t = \min[R_{A.Req}, (e_{air.o2})_t],$$

the following are constants: $R_{F.Del}$ (crew food delay) = 120, $R_{W.Del}$ (crew water delay) = 72, $R_{A.Del}$ (crew air delay) = 3, $R_{F.Req}$ (crew food requirement) = 1/6, $R_{W.Req}$ (crew water requirement) = 1/6, $R_{A.Req}$ (crew air requirement) = 40000, R_{Number} (crew number) = 4, and $R_{SciFact}$ (crew science factor) = 1. We used the notation: $[[p]] = 1$ iff p is True, $[[p]] = 0$ otherwise.; $\min[A, B] = A$ iff $(A \leq B)$, $\min[A, B] = B$ otherwise.

Definition 8 (Crop Equations) $\delta_{C_{crops}}$:

$$(O_{status})_{t+1} = \begin{cases} 0 & \text{if } [(O_{status})_t = 0] \\ & \vee [(O_{low.energy})_t > O_{L.noE}] \\ & \vee [(O_{low.water})_t > O_{L.noW}], \\ (O_{status})_t + & \\ -0.03 \times (O_{status})_t \times [[(O_{energy})_t < 0.25]] & \\ -0.01 \times (O_{status})_t \times [[(O_{water})_t < 0.25]] & \text{otherwise.} \end{cases}$$

$$(O_{low.energy})_{t+1} = \begin{cases} (O_{low.energy})_t + 1 & \text{if } [(O_{energy})_t < 0.25], \\ 0 & \text{otherwise.} \end{cases}$$

$$(O_{low.water})_{t+1} = \begin{cases} (O_{low.water})_t + 1 & \text{if } [(O_{water})_t < 0.25], \\ 0 & \text{otherwise.} \end{cases}$$

$$(O_{age})_{t+1} = \begin{cases} 0 & \text{if } [\frac{(O_{age})_t}{24} > 88], \\ (O_{age})_t + 1 & \text{otherwise.} \end{cases}$$

$$(e_{energy})_{t+1} = \{(e_{energy})_t - (O_{energy})_t\}$$

$$(e_{water.clean})_{t+1} = \{(e_{water.clean})_t - (O_{water})_t\}$$

$$(e_{water.dirty})_{t+1} = \{(e_{water.dirty})_t + (O_{water})_t\}$$

$$(e_{food})_{t+1} = \begin{cases} (e_{food})_t + (O_{biomass})_t \times 0.05 \times (O_{status})_t & \text{if } [(O_{biomass})_t \geq 0], \\ (e_{food})_t & \text{otherwise.} \end{cases}$$

$$(e_{air.o2})_{t+1} = \{(e_{air.o2})_t + (O_{o2.released})_t \times (O_{status})_t\}$$

$$(e_{air.co2})_{t+1} = \{(e_{air.co2})_t + (O_{co2.absorved})_t \times (O_{status})_t\}$$

Where:

$$(o_{energy})_t = \min[c_{energy.to.food}^t, (e_{energy})_t],$$

$$(o_{water})_t = \min[c_{water.to.crops}^t, (e_{water.clean})_t],$$

and the following are constants: $O_{L.noW}$ (crop life with no water) = 120, and $O_{L.noE}$ (crop life with no energy) = 168.

Definition 9 (Air Revitalization System Equations) $\delta_{C_{airRevitalization}}$:

$$(a_{low.energy})_{t+1} = \begin{cases} (a_{low.energy})_t + 1 & \text{if } [(a_{energy})_t < 0.25], \\ 0 & \text{otherwise.} \end{cases}$$

$$(a_{recovery.time})_{t+1} = \begin{cases} (a_{recovery.time})_t - 1 & \text{if } [(a_{energy})_t \geq 0.25] \wedge [(a_{recovery.time})_t > 0], \\ A_{T.rec} & \text{if } [(a_{low.energy})_t \geq A_{A.del}], \\ (a_{recovery.time})_t & \text{otherwise.} \end{cases}$$

$$(e_{air.o2})_{t+1} = \begin{cases} (e_{air.o2})_t + 50 \times (a_{energy})_t & \text{if } [(a_{recovery.time})_t = 0], \\ (e_{air.o2})_t & \text{otherwise.} \end{cases}$$

$$(e_{air.co2})_{t+1} = \begin{cases} (e_{air.co2})_t - 50 \times (a_{energy})_t & \text{if } [(a_{recovery.time})_t = 0], \\ (e_{air.co2})_t & \text{otherwise.} \end{cases}$$

$$(e_{energy})_{t+1} = \{(e_{energy})_t - (a_{energy})_t\}$$

Where:

$$(a_{energy})_t = \min[c_{energy.to.air}^t, (e_{energy})_t],$$

and the following are constants: $A_{T.rec}$ (air revitalization system startup time) = 3, and $A_{A.del}$ (air revitalization systems maximum energy delay time) = 1.

Definition 10 (Water Revitalization System Equations) $\delta_{C_{waterRevitalization}}$:

$$(w_{uptime})_{t+1} = \begin{cases} 0 & \text{if } [(w_{low.energy})_t > W_{T.sdo}], \\ (w_{uptime})_t + 1 & \text{otherwise.} \end{cases}$$

$$(w_{in})_{t+1} = \{(w_{in})_t + (e_{water.dirty})_t - (w_{potable})_t\}$$

$$(w_{potable})_{t+1} = \begin{cases} 0 & \text{if } [(w_{energy})_t < 0.25], \\ W_{W.rra} \times (w_{in})_t & \text{if } [(w_{uptime})_t / W_{T.wup} \geq 1], \\ ((w_{uptime})_t / W_{T.wup}) \times W_{W.rra} \times (w_{in})_t & \text{otherwise.} \end{cases}$$

$$(w_{low.energy})_{t+1} = \begin{cases} (w_{low.energy})_t + 1 & \text{if } [(w_{energy})_t < 0.5], \\ 0 & \text{otherwise.} \end{cases}$$

$$(e_{water.dirty})_{t+1} = 0$$

$$(e_{water.clean})_{t+1} = \{(e_{water.clean})_t + (w_{potable})_t\}$$

$$(e_{energy})_{t+1} = \{(e_{energy})_t - (w_{energy})_t\}$$

Where:

$$(w_{energy})_t = \min[c_{energy.to.water}^t, (e_{energy})_t],$$

and the following constants: $W_{T.wup}$ (water revitalization system warmup time) = 10, $W_{T.sdo}$ (water revitalization system shutdown time) = 5, and $W_{w.rra}$ (water recovery rate) = 0.97.

Definition 11 (Simulation Step) Applying the transition equations δ_C to the state \vec{q}_t we obtain the next state \vec{q}_{t+1} of the simulation \mathcal{A}

$$\vec{q}_{t+1} \stackrel{def}{=} \delta_C(\vec{q}_t)$$

where:

$$\delta_C(\vec{q}_t) \stackrel{def}{=} \delta_{C_{waterRevitalization}}(\delta_{C_{airRevitalization}}(\delta_{C_{crops}}(\delta_{C_{crew}}(\vec{q}_t))))$$

Definition 12 (Simulation t Steps) The reflexive and transitive closure of δ_C is denoted by $\hat{\delta}_{C,t}$:

$$\vec{q}_t \stackrel{def}{=} \hat{\delta}_{C,t}(\vec{q}_0)$$

Definition 13 (Simulation End time) Simulation \mathcal{A} ends when the environment variable crew status, r_{status} , is equal to zero (the environment is not longer able to support human life) or when the limit time for the simulation T_{max} is reached. We denote this ending time as $t_{Final}^{\mathcal{A}}$,

$$t_{Final}^{\mathcal{A}} \stackrel{def}{=} \min_{t \leq T_{max}} \left[\vec{q}_t = \hat{\delta}_{C,t}(\vec{q}_0) \wedge (r_{status})_t = 0 \right]$$

where:

$$(r_{status})_t \in \vec{q}_t, \vec{q}_t \in \mathcal{Q}, \text{ and } \mathcal{A} = \langle \mathcal{Q}, \vec{q}_0, \mathcal{C}, \delta_C \rangle.$$

Definition 14 (Final State) The final state of simulation \mathcal{A} is denoted by $\vec{q}_{t_{Final}}^{\mathcal{A}}$ and defined as follows:

$$\vec{q}_{t_{Final}}^{\mathcal{A}} \stackrel{def}{=} \left\{ \vec{q}_t \mid \vec{q}_t = \hat{\delta}_{C,t}(\vec{q}_0) \wedge t = t_{Final}^{\mathcal{A}} \right\}$$

where:

$$\vec{q}_{t_{Final}}^{\mathcal{A}}, \vec{q}_t \in \mathcal{Q}, \text{ and } \mathcal{A} = \langle \mathcal{Q}, \vec{q}_0, \mathcal{C}, \delta_C \rangle.$$

A.2 Defining the Optimization Problem

Definition 15 (Optimization 1: maximize mission productivity) Let \mathcal{C}_{Opt1} denote the optimal control strategy for maximizing mission productivity, and be defined as follows:

$$\mathcal{C}_{Opt1} \stackrel{def}{=} \left\{ \mathcal{C}_j \mid \left(\bigvee_{\mathcal{C}_i \in \mathcal{C}} \right) \left[(e_{science})_{t_{Final}}^{\mathcal{A}_j} \geq (e_{science})_{t_{Final}}^{\mathcal{A}_i} \right] \right\}$$

where

$\mathcal{C}_{Opt1}, \mathcal{C}_i$, and $\mathcal{C}_j \in \mathbf{C}$; $\mathcal{A}_i = \langle \mathcal{Q}, \vec{q}_0, \mathcal{C}_i, \delta_{\mathcal{C}_i} \rangle$; $\mathcal{A}_j = \langle \mathcal{Q}, \vec{q}_0, \mathcal{C}_j, \delta_{\mathcal{C}_j} \rangle$; $(e_{science})_{t_{Final}}^{\mathcal{A}_i} \in \vec{q}_{t_{Final}}^{\mathcal{A}_i}$; $(e_{science})_{t_{Final}}^{\mathcal{A}_j} \in \vec{q}_{t_{Final}}^{\mathcal{A}_j}$; and \mathbf{C} is the set of all possible control strategies;

Definition 16 (Optimization 2: maximize mission duration) Let \mathcal{C}_{Opt2} denote the optimal control strategy for maximizing mission duration, and be defined as follows:

$$\mathcal{C}_{Opt2} \stackrel{def}{=} \left\{ \mathcal{C}_j \mid \left(\bigvee_{\mathcal{C}_i \in \mathbf{C}} \right) [t_{Final}^{\mathcal{A}_j} \geq t_{Final}^{\mathcal{A}_i}] \right\}$$

where

$\mathcal{C}_{Opt2}, \mathcal{C}_i$, and $\mathcal{C}_j \in \mathbf{C}$; $\mathcal{A}_i = \langle \mathcal{Q}, \vec{q}_0, \mathcal{C}_i, \delta_{\mathcal{C}_i} \rangle$; $\mathcal{A}_j = \langle \mathcal{Q}, \vec{q}_0, \mathcal{C}_j, \delta_{\mathcal{C}_j} \rangle$; and \mathbf{C} is the set of all possible control strategies;

Definition 17 (Optimization 3: maximize mission productivity and duration) Let \mathcal{C}_{Opt3} denote the optimal control strategy for maximizing mission productivity and mission duration, and be defined as follows:

$$\mathcal{C}_{Opt3} \stackrel{def}{=} \left\{ \mathcal{C}_j \mid \left(\bigvee_{\mathcal{C}_i \in \mathbf{C}} \right) \left[(e_{science})_{t_{Final}}^{\mathcal{A}_j} + 3 \times t_{Final}^{\mathcal{A}_j} \geq (e_{science})_{t_{Final}}^{\mathcal{A}_i} + 3 \times t_{Final}^{\mathcal{A}_i} \right] \right\}$$

where

$\mathcal{C}_{Opt3}, \mathcal{C}_i$, and $\mathcal{C}_j \in \mathbf{C}$; $\mathcal{A}_i = \langle \mathcal{Q}, \vec{q}_0, \mathcal{C}_i, \delta_{\mathcal{C}_i} \rangle$; $\mathcal{A}_j = \langle \mathcal{Q}, \vec{q}_0, \mathcal{C}_j, \delta_{\mathcal{C}_j} \rangle$; $(e_{science})_{t_{Final}}^{\mathcal{A}_i} \in \vec{q}_{t_{Final}}^{\mathcal{A}_i}$; $(e_{science})_{t_{Final}}^{\mathcal{A}_j} \in \vec{q}_{t_{Final}}^{\mathcal{A}_j}$; and \mathbf{C} is the set of all possible control strategies;

A.3 Defining the Fitness Functions

Definition 18 (Fitness 1: mission productivity) Let $f_1(\mathcal{C}_k)$ denote the fitness of the control strategy \mathcal{C}_k while optimizing mission productivity in simulation \mathcal{A}_k , be defined as follows:

$$f_1(\mathcal{C}_k) = \left\{ (e_{science})_{t_{Final}}^{\mathcal{A}_k} \mid (e_{science})_{t_{Final}}^{\mathcal{A}_k} \in \vec{q}_{t_{Final}}^{\mathcal{A}_k} \bigwedge \mathcal{A}_k = \langle \mathcal{Q}, \vec{q}_0, \mathcal{C}_k, \delta_{\mathcal{C}_k} \rangle \right\}$$

Definition 19 (Fitness 2: mission duration) Let $f_2(\mathcal{C}_k)$ denote the fitness of the control strategy \mathcal{C}_k while optimizing mission duration in simulation \mathcal{A}_k , be defined as follows:

$$f_2(\mathcal{C}_k) = \left\{ t_{Final}^{\mathcal{A}_k} \mid \mathcal{A}_k = \langle \mathcal{Q}, \vec{q}_0, \mathcal{C}_k, \delta_{\mathcal{C}_k} \rangle \right\}$$

Definition 20 (Fitness 3: mission productivity and duration) Let $f_3(\mathcal{C}_k)$ denote the fitness of the control strategy \mathcal{C}_k while optimizing mission productivity and mission duration in simulation \mathcal{A}_k , be defined as follows:

$$f_3(\mathcal{C}_k) = \left\{ (e_{science})_{t_{Final}}^{\mathcal{A}_k} + 3 \times t_{Final}^{\mathcal{A}_k} \mid (e_{science})_{t_{Final}}^{\mathcal{A}_k} \in \vec{q}_{t_{Final}}^{\mathcal{A}_k} \bigwedge \mathcal{A}_k = \langle \mathcal{Q}, \vec{q}_0, \mathcal{C}_k, \delta_{\mathcal{C}_k} \rangle \right\}$$

APPENDIX B
ADDITIONAL TABLES AND FIGURES

Table B.1: Results of PGA1 and GA on finding randomly generated allocation values. Average and standard deviation (std) over 100 runs.

| Best of run (BOR) fitness, hits, generation found, and genome length | | | | | | | | |
|---|-------------|--------|------------------|------------|-----------|--------|---------|--------|
| Algorithm | BOR fitness | (std) | Hit: ≥ 0.99 | ≥ 1.0 | Gen Found | (std) | Length | (std) |
| ga | 0.9857 | 0.0120 | 53 | 0 | 48.84 | 90.46 | 40.00 | 0.00 |
| pga 40 | 0.9382 | 0.0483 | 2 | 0 | 5.56 | 2.61 | 40.00 | 0.00 |
| pga 255 | 0.9869 | 0.0110 | 59 | 0 | 28.13 | 15.74 | 255.00 | 0.00 |
| pga var | 0.9941 | 0.0189 | 91 | 9 | 200.60 | 151.08 | 1641.31 | 702.53 |
| pga varpp | 0.9907 | 0.0145 | 72 | 0 | 138.64 | 143.24 | 278.72 | 137.97 |
| pga NC 40 | 0.9495 | 0.0496 | 1 | 0 | 14.14 | 28.07 | 40.00 | 0.00 |
| pga NC 255 | 0.9955 | 0.0063 | 94 | 34 | 76.62 | 90.01 | 255.00 | 0.00 |
| pga NC var | 0.9963 | 0.0135 | 95 | 30 | 200.66 | 147.86 | 1596.88 | 748.68 |
| pga NC varpp | 0.9914 | 0.0060 | 72 | 0 | 163.77 | 145.53 | 330.92 | 168.95 |

Best fitness and length averaged over last 100 generations, average run time

| Algorithm | Best fitness | (std) | Length | (std) | Time in secs. | (std) |
|--------------|--------------|--------|---------|--------|---------------|-------|
| ga | 0.9857 | 0.0120 | 40.00 | 0.00 | 13.96 | 1.24 |
| pga 40 | 0.9382 | 0.0483 | 40.00 | 0.00 | 12.08 | 0.65 |
| pga 255 | 0.9867 | 0.0108 | 255.00 | 0.00 | 15.75 | 4.38 |
| pga var | 0.9873 | 0.0341 | 1813.56 | 585.68 | 46.86 | 16.77 |
| pga varpp | 0.9808 | 0.0208 | 970.95 | 424.00 | 30.41 | 12.75 |
| pga NC 40 | 0.9487 | 0.0495 | 40.00 | 0.00 | 12.48 | 1.02 |
| pga NC 255 | 0.9942 | 0.0060 | 255.00 | 0.00 | 16.12 | 3.66 |
| pga NC var | 0.9929 | 0.0162 | 1810.30 | 606.37 | 47.53 | 13.72 |
| pga NC varpp | 0.9824 | 0.0075 | 986.95 | 440.54 | 30.78 | 8.19 |

The following is the full collection of tables and figures discussed on chapters 7, and 9. These chapters have already shown some elements of this collection. There are two sets of figures and tables each one corresponding to one of these two chapters. The first set corresponds to chapter 7 and provides all the data related to PGA1 on the resource allocation problem and PGA2 and PGA3 on the number matching and symbolic regression problem. The last set corresponds to chapter 9 and provides all the tables related to the PGA applied to the ALSS problem.

Table B.2: Results of PGA1 and GA on finding a fixed test allocation of values: 1:2:4:2:1. Average and standard deviation (std) over 100 runs.

| Best of run (BOR) fitness, hits, generation found, and genome length | | | | | | | | |
|---|-------------|--------|------------------|------------|-----------|--------|---------|--------|
| Algorithm | BOR fitness | (std) | Hit: ≥ 0.99 | ≥ 1.0 | Gen Found | (std) | Length | (std) |
| ga | 0.9911 | 0.0009 | 80 | 0 | 62.00 | 111.72 | 40.00 | 0.00 |
| pga 40 | 1.0000 | 0.0000 | 100 | 100 | 5.02 | 1.61 | 40.00 | 0.00 |
| pga 255 | 0.9922 | 0.0000 | 100 | 0 | 18.66 | 2.12 | 255.00 | 0.00 |
| pga var | 0.9997 | 0.0003 | 100 | 60 | 173.70 | 155.94 | 1460.33 | 849.38 |
| pga varpp | 0.9961 | 0.0032 | 100 | 0 | 65.68 | 98.42 | 220.54 | 154.34 |
| pga NC 40 | 1.0000 | 0.0000 | 100 | 100 | 5.82 | 2.99 | 40.00 | 0.00 |
| pga NC 255 | 1.0000 | 0.0000 | 100 | 100 | 23.63 | 6.75 | 255.00 | 0.00 |
| pga NC var | 1.0000 | 0.0001 | 100 | 99 | 96.54 | 114.98 | 1129.86 | 933.11 |
| pga NC varpp | 0.9966 | 0.0029 | 100 | 0 | 117.27 | 137.60 | 227.63 | 171.19 |

Best fitness and length averaged over last 100 generations, average run time

| Algorithm | Best fitness | (std) | Length | (std) | Time in secs. | (std) |
|--------------|--------------|--------|---------|--------|---------------|-------|
| ga | 0.9914 | 0.0012 | 40.00 | 0.00 | 5.23 | 0.45 |
| pga 40 | 1.0000 | 0.0000 | 40.00 | 0.00 | 3.34 | 0.48 |
| pga 255 | 0.9918 | 0.0001 | 255.00 | 0.00 | 8.27 | 0.45 |
| pga var | 0.9978 | 0.0004 | 1978.44 | 339.27 | 53.62 | 8.87 |
| pga varpp | 0.9849 | 0.0050 | 1006.78 | 367.67 | 24.33 | 7.54 |
| pga NC 40 | 1.0000 | 0.0000 | 40.00 | 0.00 | 3.76 | 0.43 |
| pga NC 255 | 0.9994 | 0.0008 | 255.00 | 0.00 | 8.36 | 0.48 |
| pga NC var | 0.9979 | 0.0007 | 1841.11 | 581.90 | 50.63 | 14.96 |
| pga NC varpp | 0.9841 | 0.0038 | 1030.13 | 252.74 | 28.00 | 5.54 |

Table B.3: Results of PGA2 and GA on the number matching problem. Average and standard deviation (std) over 100 runs.

| Best of run (BOR) fitness, hits, generation found, and genome length | | | | | | | | |
|---|-------------|--------|------------------|------------|-----------|--------|---------|--------|
| Algorithm | BOR fitness | (std) | Hit: ≥ 0.99 | ≥ 1.0 | Gen Found | (std) | Length | (std) |
| ga | 0.9931 | 0.0096 | 87 | 0 | 41.94 | 75.25 | 40.00 | 0.00 |
| pga 40 | 0.9498 | 0.0711 | 4 | 0 | 78.39 | 127.52 | 40.00 | 0.00 |
| pga 255 | 0.9978 | 0.0025 | 99 | 0 | 266.64 | 123.66 | 255.00 | 0.00 |
| pga var | 0.9714 | 0.0686 | 76 | 0 | 274.25 | 157.47 | 1731.65 | 660.34 |
| pga varpp | 0.9815 | 0.0409 | 63 | 0 | 163.11 | 148.97 | 368.92 | 229.04 |
| pga NC 40 | 0.9507 | 0.0688 | 5 | 0 | 68.43 | 109.18 | 40.00 | 0.00 |
| pga NC 255 | 0.9974 | 0.0035 | 99 | 0 | 254.12 | 123.72 | 255.00 | 0.00 |
| pga NC var | 0.9755 | 0.0663 | 81 | 0 | 287.63 | 143.83 | 1772.37 | 626.79 |
| pga NC varpp | 0.9851 | 0.0334 | 64 | 0 | 230.43 | 159.72 | 432.59 | 252.73 |

Best fitness and length averaged over last 100 generations, average run time

| Algorithm | Best fitness | (std) | Length | (std) | Time in secs. | (std) |
|--------------|--------------|--------|---------|--------|---------------|-------|
| ga | 0.9931 | 0.0096 | 40.00 | 0.00 | 5.22 | 0.44 |
| pga 40 | 0.9483 | 0.0710 | 40.00 | 0.00 | 3.08 | 0.27 |
| pga 255 | 0.9947 | 0.0044 | 255.00 | 0.00 | 8.79 | 0.41 |
| pga var | 0.9660 | 0.0703 | 1779.87 | 648.86 | 51.57 | 17.97 |
| pga varpp | 0.9760 | 0.0419 | 713.61 | 492.30 | 22.60 | 12.50 |
| pga NC 40 | 0.9500 | 0.0696 | 40.00 | 0.00 | 2.65 | 0.48 |
| pga NC 255 | 0.9950 | 0.0037 | 255.00 | 0.00 | 7.04 | 0.24 |
| pga NC var | 0.9702 | 0.0672 | 1816.46 | 601.64 | 46.37 | 14.68 |
| pga NC varpp | 0.9785 | 0.0349 | 711.81 | 458.03 | 19.86 | 10.40 |

Table B.4: Results of PGA3 and GA on the number matching problem. Average and standard deviation (std) over 100 runs.

| Best of run (BOR) fitness, hits, generation found, and genome length | | | | | | | | |
|---|-------------|--------|------------------|------------|-----------|--------|---------|--------|
| Algorithm | BOR fitness | (std) | Hit: ≥ 0.99 | ≥ 1.0 | Gen Found | (std) | Length | (std) |
| ga | 0.9931 | 0.0096 | 87 | 0 | 41.94 | 75.25 | 40.00 | 0.00 |
| pga 40 | 0.9183 | 0.0794 | 0 | 0 | 68.56 | 110.01 | 40.00 | 0.00 |
| pga 255 | 0.9897 | 0.0284 | 94 | 0 | 276.82 | 147.93 | 255.00 | 0.00 |
| pga var | 0.9622 | 0.0757 | 70 | 0 | 290.03 | 150.39 | 1750.88 | 659.36 |
| pga varpp | 0.9599 | 0.0672 | 3 | 0 | 140.32 | 133.89 | 683.00 | 408.64 |
| pga NC 40 | 0.9235 | 0.0729 | 0 | 0 | 92.33 | 126.98 | 40.00 | 0.00 |
| pga NC 255 | 0.9917 | 0.0161 | 95 | 0 | 291.53 | 135.13 | 255.00 | 0.00 |
| pga NC var | 0.9647 | 0.0748 | 75 | 0 | 284.40 | 157.54 | 1788.72 | 604.25 |
| pga NC varpp | 0.9588 | 0.0679 | 3 | 0 | 140.73 | 139.59 | 751.08 | 445.44 |

Best fitness and length averaged over last 100 generations, average run time

| Algorithm | Best fitness | (std) | Length | (std) | Time in secs. | (std) |
|--------------|--------------|--------|---------|--------|---------------|-------|
| ga | 0.9931 | 0.0096 | 40.00 | 0.00 | 5.22 | 0.44 |
| pga 40 | 0.9150 | 0.0789 | 40.00 | 0.00 | 3.11 | 0.31 |
| pga 255 | 0.9810 | 0.0303 | 255.00 | 0.00 | 8.21 | 0.54 |
| pga var | 0.9557 | 0.0767 | 1772.01 | 651.76 | 49.38 | 17.91 |
| pga varpp | 0.9492 | 0.0673 | 1465.12 | 540.93 | 41.12 | 13.58 |
| pga NC 40 | 0.9219 | 0.0725 | 40.00 | 0.00 | 2.70 | 0.46 |
| pga NC 255 | 0.9835 | 0.0215 | 255.00 | 0.00 | 7.10 | 0.30 |
| pga NC var | 0.9576 | 0.0759 | 1812.89 | 605.12 | 46.66 | 15.05 |
| pga NC varpp | 0.9489 | 0.0682 | 1478.74 | 571.24 | 37.25 | 12.75 |

Table B.5: Results of PGA2 and GA on the symbolic regression problem. Average and standard deviation (std) over 100 runs.

| Best of run (BOR) fitness, hits, generation found, and genome length | | | | | | | | |
|---|-------------|--------|------------------|------------|-----------|--------|---------|--------|
| Algorithm | BOR fitness | (std) | Hit: ≥ 0.99 | ≥ 1.0 | Gen Found | (std) | Length | (std) |
| ga | 0.9957 | 0.0042 | 90 | 0 | 79.02 | 117.05 | 40.00 | 0.00 |
| pga 40 | 0.9931 | 0.0080 | 83 | 0 | 80.89 | 118.92 | 40.00 | 0.00 |
| pga 255 | 0.9991 | 0.0007 | 100 | 0 | 315.01 | 125.00 | 255.00 | 0.00 |
| pga var | 0.9948 | 0.0083 | 89 | 0 | 318.42 | 148.17 | 1639.59 | 747.33 |
| pga varpp | 0.9962 | 0.0027 | 99 | 0 | 201.40 | 152.27 | 129.68 | 48.18 |
| pga NC 40 | 0.9923 | 0.0084 | 79 | 0 | 63.73 | 106.20 | 40.00 | 0.00 |
| pga NC 255 | 0.9991 | 0.0007 | 100 | 0 | 309.24 | 123.88 | 255.00 | 0.00 |
| pga NC var | 0.9953 | 0.0074 | 89 | 0 | 379.85 | 135.64 | 1748.98 | 654.69 |
| pga NC varpp | 0.9955 | 0.0034 | 94 | 0 | 228.21 | 149.07 | 145.83 | 59.39 |

Best fitness and length averaged over last 100 generations, average run time

| Algorithm | Best fitness | (std) | Length | (std) | Time in secs. | (std) |
|--------------|--------------|--------|---------|--------|---------------|-------|
| ga | 0.9957 | 0.0043 | 40.00 | 0.00 | 79.73 | 0.66 |
| pga 40 | 0.9928 | 0.0080 | 40.00 | 0.00 | 20.69 | 0.46 |
| pga 255 | 0.9982 | 0.0010 | 255.00 | 0.00 | 24.59 | 0.49 |
| pga var | 0.9937 | 0.0089 | 1717.67 | 714.50 | 58.86 | 16.07 |
| pga varpp | 0.9958 | 0.0029 | 160.93 | 115.84 | 24.58 | 2.54 |
| pga NC 40 | 0.9917 | 0.0086 | 40.00 | 0.00 | 21.55 | 0.50 |
| pga NC 255 | 0.9986 | 0.0009 | 255.00 | 0.00 | 25.80 | 0.55 |
| pga NC var | 0.9940 | 0.0086 | 1777.57 | 651.43 | 68.21 | 14.36 |
| pga NC varpp | 0.9952 | 0.0035 | 161.84 | 99.46 | 25.75 | 2.73 |

Table B.6: Results of PGA3 and GA on the symbolic regression problem. Average and standard deviation (std) over 100 runs.

| Best of run (BOR) fitness, hits, generation found, and genome length | | | | | | | | |
|---|-------------|--------|------------------|------------|-----------|--------|---------|--------|
| Algorithm | BOR fitness | (std) | Hit: ≥ 0.99 | ≥ 1.0 | Gen Found | (std) | Length | (std) |
| ga | 0.9957 | 0.0042 | 90 | 0 | 79.02 | 117.05 | 40.00 | 0.00 |
| pga 40 | 0.9882 | 0.0095 | 53 | 0 | 62.71 | 86.50 | 40.00 | 0.00 |
| pga 255 | 0.9983 | 0.0010 | 100 | 0 | 302.51 | 121.56 | 255.00 | 0.00 |
| pga var | 0.9946 | 0.0086 | 89 | 0 | 338.00 | 143.22 | 1702.14 | 705.63 |
| pga varpp | 0.9924 | 0.0081 | 89 | 0 | 246.62 | 147.62 | 205.31 | 103.75 |
| pga NC 40 | 0.9862 | 0.0103 | 46 | 0 | 53.90 | 84.72 | 40.00 | 0.00 |
| pga NC 255 | 0.9978 | 0.0027 | 98 | 0 | 329.15 | 121.50 | 255.00 | 0.00 |
| pga NC var | 0.9947 | 0.0082 | 90 | 0 | 331.08 | 141.86 | 1814.50 | 578.26 |
| pga NC varpp | 0.9927 | 0.0076 | 92 | 0 | 243.25 | 161.72 | 194.08 | 100.55 |

Best fitness and length averaged over last 100 generations, average run time

| Algorithm | Best fitness | (std) | Length | (std) | Time in secs. | (std) |
|--------------|--------------|--------|---------|--------|---------------|-------|
| ga | 0.9957 | 0.0043 | 40.00 | 0.00 | 79.73 | 0.66 |
| pga 40 | 0.9872 | 0.0099 | 40.00 | 0.00 | 21.69 | 0.51 |
| pga 255 | 0.9959 | 0.0016 | 255.00 | 0.00 | 25.58 | 0.50 |
| pga var | 0.9927 | 0.0097 | 1743.23 | 691.39 | 60.97 | 15.67 |
| pga varpp | 0.9883 | 0.0085 | 555.23 | 370.05 | 33.30 | 7.51 |
| pga NC 40 | 0.9850 | 0.0110 | 40.00 | 0.00 | 30.70 | 0.52 |
| pga NC 255 | 0.9964 | 0.0029 | 255.00 | 0.00 | 35.50 | 0.56 |
| pga NC var | 0.9926 | 0.0092 | 1854.02 | 549.69 | 70.15 | 11.85 |
| pga NC varpp | 0.9895 | 0.0084 | 480.38 | 392.39 | 41.61 | 8.06 |

Table B.7: PGA and GA on NASA ALSS simulator, optimizing mission productivity with strategy Γ_1 . Average and standard deviation (std) over 40 runs.

| Best of run (BOR) fitness, generation found, genome length | | | | | | | |
|---|-------------|-----------|------------------|--------|---------------|-------|--|
| Algorithm | BOR fitness | (std) | Generation Found | (std) | Genome Length | (std) | |
| ga 25 | 25508.7386 | 724.1122 | 124.58 | 142.51 | 25.00 | 0.00 | |
| pga 25 | 20535.9791 | 1486.5627 | 60.40 | 38.10 | 25.00 | 0.00 | |
| pga 40 | 25421.7159 | 879.9272 | 105.60 | 62.91 | 40.00 | 0.00 | |
| pga var (max 80) | 29748.5530 | 1290.3420 | 282.80 | 112.35 | 70.33 | 8.22 | |

Best fitness averaged over last 100 generations, average run time

| Algorithm | Best fitness | (std) | Time in secs. | (std) |
|------------------|--------------|-----------|---------------|--------|
| ga 25 | 25402.9143 | 827.1380 | 999.83 | 87.49 |
| pga 25 | 20506.9776 | 1569.8013 | 686.90 | 106.12 |
| pga 40 | 25421.7162 | 879.9273 | 1169.75 | 156.23 |
| pga var (max 80) | 29733.8987 | 1292.8265 | 1565.38 | 897.29 |

Table B.8: PGA and GA on NASA ALSS simulator, optimizing mission productivity with strategy Γ_3 . Average and standard deviation (std) over 40 runs.

| Best of run (BOR) fitness, generation found, genome length | | | | | | |
|---|-------------------|-----------|------------------------|--------|---------------------|-------|
| Algorithm | BOR fitness (std) | | Generation Found (std) | | Genome Length (std) | |
| ga 75 | 30731.4456 | 2084.0995 | 323.85 | 133.39 | 75.00 | 0.00 |
| pga 75 | 34144.1525 | 2841.0517 | 341.58 | 138.67 | 75.00 | 0.00 |
| pga 120 | 34931.2785 | 1275.5678 | 425.00 | 87.54 | 120.00 | 0.00 |
| pga var (max 240) | 34894.6559 | 1432.4015 | 419.35 | 71.63 | 187.05 | 31.95 |

Best fitness averaged over last 100 generations, average run time

| Algorithm | Best fitness (std) | | Time in secs. (std) | |
|-------------------|--------------------|-----------|---------------------|--------|
| ga 75 | 30643.0013 | 2110.7448 | 1385.30 | 157.51 |
| pga 75 | 34042.6220 | 2820.7889 | 1580.78 | 216.23 |
| pga 120 | 34649.3913 | 1226.5390 | 1545.05 | 120.76 |
| pga var (max 240) | 34570.0414 | 1320.3384 | 1397.20 | 114.65 |

Table B.9: PGA and GA on NASA ALSS simulator, optimizing mission productivity with strategy Γ_5 . Average and standard deviation (std) over 40 runs.

| Best of run (BOR) fitness, generation found, genome length | | | | | | |
|---|-------------------|-----------|------------------------|-------|---------------------|-------|
| Algorithm | BOR fitness (std) | | Generation Found (std) | | Genome Length (std) | |
| ga 125 | 33484.7394 | 2009.4459 | 376.53 | 99.50 | 125.00 | 0.00 |
| pga 125 | 37026.3127 | 1026.1128 | 445.13 | 78.06 | 125.00 | 0.00 |
| pga 200 | 38251.8807 | 674.3510 | 457.15 | 43.18 | 200.00 | 0.00 |
| pga var (max 400) | 35931.0829 | 1857.5803 | 463.33 | 47.22 | 295.23 | 81.18 |

Best fitness averaged over last 100 generations, average run time

| Algorithm | Best fitness (std) | | Time in secs. (std) | |
|-------------------|--------------------|-----------|---------------------|--------|
| ga 125 | 33305.0258 | 2021.9056 | 1748.33 | 201.09 |
| pga 125 | 36803.4582 | 1049.3608 | 1906.00 | 175.88 |
| pga 200 | 37788.2305 | 663.9422 | 1793.80 | 129.87 |
| pga var (max 400) | 35356.9361 | 1945.3999 | 1153.45 | 257.53 |

Table B.10: PGA and GA on NASA ALSS simulator, optimizing mission productivity with strategy Γ_7 . Average and standard deviation (std) over 40 runs.

| Best of run (BOR) fitness, generation found, genome length | | | | | | |
|---|-------------|-----------|------------------|-------|---------------|--------|
| Algorithm | BOR fitness | (std) | Generation Found | (std) | Genome Length | (std) |
| ga 175 | 34403.5451 | 1110.4432 | 428.05 | 65.50 | 175.00 | 0.00 |
| pga 175 | 38118.8467 | 1031.2142 | 449.65 | 45.47 | 175.00 | 0.00 |
| pga 280 | 37965.6329 | 1015.8243 | 481.35 | 26.08 | 280.00 | 0.00 |
| pga var (max 560) | 36842.7540 | 1577.1880 | 476.20 | 30.03 | 368.63 | 108.85 |

Best fitness averaged over last 100 generations, average run time

| Algorithm | Best fitness | (std) | Time in secs. | (std) |
|-------------------|--------------|-----------|---------------|--------|
| ga 175 | 34108.4007 | 1178.1366 | 1431.53 | 79.30 |
| pga 175 | 37838.2114 | 1254.3451 | 1595.63 | 177.61 |
| pga 280 | 37503.1610 | 1088.9034 | 1475.63 | 119.07 |
| pga var (max 560) | 36350.8311 | 1702.5978 | 1280.13 | 155.76 |

Table B.11: PGA and GA on NASA ALSS simulator, optimizing mission productivity with strategy Γ_9 . Average and standard deviation (std) over 40 runs.

| Best of run (BOR) fitness, generation found, genome length | | | | | | |
|---|-------------|-----------|------------------|-------|---------------|--------|
| Algorithm | BOR fitness | (std) | Generation Found | (std) | Genome Length | (std) |
| ga 255 | 34589.8542 | 1349.3034 | 421.88 | 74.25 | 225.00 | 0.00 |
| pga 255 | 38016.9042 | 1665.4648 | 447.55 | 76.41 | 225.00 | 0.00 |
| pga 360 | 37644.3455 | 1219.9812 | 486.70 | 15.76 | 360.00 | 0.00 |
| pga var (max 720) | 34995.9718 | 1521.0330 | 474.33 | 24.92 | 564.88 | 110.77 |

Best fitness averaged over last 100 generations, average run time

| Algorithm | Best fitness | (std) | Time in secs. | (std) |
|-------------------|--------------|-----------|---------------|--------|
| ga 255 | 34411.3304 | 1301.7954 | 1357.38 | 100.87 |
| pga 255 | 37508.3537 | 1637.3417 | 1416.58 | 132.29 |
| pga 360 | 37119.4869 | 1248.0095 | 1349.40 | 108.96 |
| pga var (max 720) | 34331.8560 | 1595.7883 | 1147.18 | 114.50 |

Table B.12: PGA and GA on NASA ALSS simulator, optimizing mission duration with strategy Γ_1 . Average and standard deviation (std) over 40 runs.

| Best of run (BOR) fitness, generation found, genome length | | | | | | | |
|---|-------------------|----------|------------------------|--------|---------------------|------|--|
| Algorithm | BOR fitness (std) | | Generation Found (std) | | Genome Length (std) | | |
| ga 25 | 8589.3250 | 296.6670 | 146.65 | 132.49 | 25.00 | 0.00 | |
| pga 25 | 6654.1250 | 648.4009 | 100.28 | 87.49 | 25.00 | 0.00 | |
| pga 40 | 9557.0000 | 0.0000 | 86.98 | 43.51 | 40.00 | 0.00 | |
| pga var (max 80) | 10030.8500 | 197.0098 | 321.85 | 131.29 | 72.70 | 5.15 | |

Best fitness averaged over last 100 generations, average run time

| Algorithm | Best fitness (std) | | Time in secs. (std) | |
|------------------|--------------------|----------|---------------------|--------|
| ga 25 | 8574.6000 | 306.9484 | 1395.95 | 66.81 |
| pga 25 | 6654.1250 | 648.4009 | 938.78 | 151.35 |
| pga 40 | 9557.0000 | 0.0000 | 2177.08 | 18.77 |
| pga var (max 80) | 10019.5000 | 188.6309 | 2141.55 | 66.57 |

Table B.13: PGA and GA on NASA ALSS simulator, optimizing mission duration with strategy Γ_3 . Average and standard deviation (std) over 40 runs.

| Best of run (BOR) fitness, generation found, genome length | | | | | | | |
|---|-------------------|----------|------------------------|--------|---------------------|-------|--|
| Algorithm | BOR fitness (std) | | Generation Found (std) | | Genome Length (std) | | |
| ga 75 | 9122.3750 | 0.7742 | 213.73 | 127.05 | 75.00 | 0.00 | |
| pga 75 | 9153.1250 | 458.5895 | 265.15 | 116.78 | 75.00 | 0.00 | |
| pga 120 | 9197.4750 | 535.7199 | 200.40 | 169.00 | 120.00 | 0.00 | |
| pga var (max 240) | 9072.1000 | 1.8369 | 85.30 | 102.33 | 191.80 | 58.27 | |

Best fitness averaged over last 100 generations, average run time

| Algorithm | Best fitness (std) | | Time in secs. (std) | |
|-------------------|--------------------|----------|---------------------|--------|
| ga 75 | 9122.3750 | 0.7742 | 1682.75 | 38.86 |
| pga 75 | 9152.8655 | 458.6336 | 1719.53 | 135.86 |
| pga 120 | 9197.2133 | 535.7805 | 1724.35 | 43.55 |
| pga var (max 240) | 9071.6250 | 1.6747 | 1768.25 | 34.89 |

Table B.14: PGA and GA on NASA ALSS simulator, optimizing mission duration with strategy Γ_5 . Average and standard deviation (std) over 40 runs.

| Best of run (BOR) fitness, generation found, genome length | | | | | | |
|---|-------------|----------|------------------|--------|---------------|-------|
| Algorithm | BOR fitness | (std) | Generation Found | (std) | Genome Length | (std) |
| ga 125 | 14757.1250 | 544.3968 | 256.63 | 153.35 | 125.00 | 0.00 |
| pga 125 | 14348.2500 | 822.3704 | 171.23 | 123.08 | 125.00 | 0.00 |
| pga 200 | 14643.9000 | 682.2123 | 226.20 | 144.76 | 200.00 | 0.00 |
| pga var (max 400) | 14993.5000 | 340.6499 | 173.38 | 107.43 | 260.33 | 95.21 |

Best fitness averaged over last 100 generations, average run time

| Algorithm | Best fitness | (std) | Time in secs. | (std) |
|-------------------|--------------|----------|---------------|--------|
| ga 125 | 14744.6583 | 555.0273 | 2932.40 | 81.57 |
| pga 125 | 14348.0198 | 822.1618 | 2875.43 | 147.45 |
| pga 200 | 14627.2498 | 704.4844 | 2891.08 | 142.04 |
| pga var (max 400) | 14988.9743 | 362.1876 | 2878.73 | 99.67 |

Table B.15: PGA and GA on NASA ALSS simulator, optimizing mission duration with strategy Γ_7 . Average and standard deviation (std) over 40 runs.

| Best of run (BOR) fitness, generation found, genome length | | | | | | |
|---|-------------|-----------|------------------|--------|---------------|-------|
| Algorithm | BOR fitness | (std) | Generation Found | (std) | Genome Length | (std) |
| ga 175 | 15344.1000 | 955.2187 | 142.90 | 114.95 | 175.00 | 0.00 |
| pga 175 | 14821.5500 | 1281.8471 | 182.63 | 140.90 | 175.00 | 0.00 |
| pga 280 | 14430.6250 | 1294.9936 | 284.65 | 148.43 | 280.00 | 0.00 |
| pga var (max 560) | 15786.0500 | 975.8598 | 174.78 | 65.15 | 409.28 | 93.03 |

Best fitness averaged over last 100 generations, average run time

| Algorithm | Best fitness | (std) | Time in secs. | (std) |
|-------------------|--------------|-----------|---------------|--------|
| ga 175 | 15343.5385 | 956.3084 | 3070.53 | 186.09 |
| pga 175 | 14803.1260 | 1298.7209 | 2982.90 | 294.28 |
| pga 280 | 14416.7418 | 1303.5396 | 2815.23 | 273.15 |
| pga var (max 560) | 15785.3905 | 977.8841 | 2965.13 | 196.32 |

Table B.16: PGA and GA on NASA ALSS simulator, optimizing mission duration with strategy Γ_9 . Average and standard deviation (std) over 40 runs.

| Best of run (BOR) fitness, generation found, genome length | | | | | | |
|---|-------------|-----------|------------------|--------|---------------|--------|
| Algorithm | BOR fitness | (std) | Generation Found | (std) | Genome Length | (std) |
| ga 255 | 14847.7250 | 1041.0159 | 210.70 | 137.31 | 225.00 | 0.00 |
| pga 255 | 15170.5250 | 1283.4939 | 206.58 | 139.78 | 225.00 | 0.00 |
| pga 360 | 16012.2000 | 592.4544 | 165.20 | 56.25 | 360.00 | 0.00 |
| pga var (max 720) | 15568.9750 | 1099.4568 | 268.98 | 116.82 | 548.68 | 137.83 |

Best fitness averaged over last 100 generations, average run time

| Algorithm | Best fitness | (std) | Time in secs. | (std) |
|-------------------|--------------|-----------|---------------|--------|
| ga 255 | 14843.4843 | 1044.3307 | 3115.23 | 221.50 |
| pga 255 | 15165.8625 | 1289.7281 | 3170.23 | 346.42 |
| pga 360 | 16010.8238 | 598.0638 | 3331.95 | 152.68 |
| pga var (max 720) | 15562.9788 | 1109.6910 | 3051.20 | 262.59 |

Table B.17: PGA and GA on NASA ALSS simulator, optimizing mission productivity and duration with strategy Γ_1 . Average and standard deviation (std) over 40 runs.

| Best of run (BOR) fitness, generation found, genome length | | | | | | |
|---|-------------------|-----------|------------------------|--------|---------------------|-------|
| Algorithm | BOR fitness (std) | | Generation Found (std) | | Genome Length (std) | |
| ga 25 | 50625.7030 | 1876.3654 | 192.63 | 186.20 | 25.00 | 0.00 |
| pga 25 | 40592.0659 | 781.0210 | 74.33 | 39.31 | 25.00 | 0.00 |
| pga 40 | 57261.7518 | 0.0013 | 104.80 | 51.43 | 40.00 | 0.00 |
| pga var (max 80) | 59781.3850 | 2548.3053 | 211.68 | 111.72 | 65.28 | 13.27 |

Best fitness averaged over last 100 generations, average run time

| Algorithm | Best fitness (std) | | Time in secs. (std) | |
|------------------|--------------------|-----------|---------------------|--------|
| ga 25 | 50227.2588 | 2076.9028 | 1337.73 | 130.40 |
| pga 25 | 40592.0660 | 781.0207 | 983.20 | 52.23 |
| pga 40 | 57261.7520 | 0.0000 | 2129.55 | 74.91 |
| pga var (max 80) | 59413.7809 | 2922.5503 | 1986.23 | 216.90 |

Table B.18: PGA and GA on NASA ALSS simulator, optimizing mission productivity and duration with strategy Γ_3 . Average and standard deviation (std) over 40 runs.

| Best of run (BOR) fitness, generation found, genome length | | | | | | |
|---|-------------------|-----------|------------------------|--------|---------------------|-------|
| Algorithm | BOR fitness (std) | | Generation Found (std) | | Genome Length (std) | |
| ga 75 | 63713.7350 | 4314.5025 | 275.90 | 120.37 | 75.00 | 0.00 |
| pga 75 | 69356.9502 | 3388.4249 | 262.53 | 139.01 | 75.00 | 0.00 |
| pga 120 | 72641.8553 | 1731.0901 | 413.85 | 95.25 | 120.00 | 0.00 |
| pga var (max 240) | 72020.5877 | 3037.7582 | 373.35 | 115.62 | 184.85 | 46.12 |

Best fitness averaged over last 100 generations, average run time

| Algorithm | Best fitness (std) | | Time in secs. (std) | |
|-------------------|--------------------|-----------|---------------------|--------|
| ga 75 | 63687.1519 | 4298.5406 | 1597.40 | 141.11 |
| pga 75 | 69342.3989 | 3430.6253 | 1714.00 | 178.93 |
| pga 120 | 72444.2433 | 1866.8026 | 1788.10 | 103.41 |
| pga var (max 240) | 71314.2104 | 3314.4401 | 1503.90 | 166.90 |

Table B.19: PGA and GA on NASA ALSS simulator, optimizing mission productivity and duration with strategy Γ_5 . Average and standard deviation (std) over 40 runs.

| Best of run (BOR) fitness, generation found, genome length | | | | | | | |
|---|-------------------|-----------|------------------------|--------|---------------------|-------|--|
| Algorithm | BOR fitness (std) | | Generation Found (std) | | Genome Length (std) | | |
| ga 125 | 70490.1496 | 2772.1931 | 372.38 | 102.88 | 125.00 | 0.00 | |
| pga 125 | 75442.8339 | 1499.8686 | 385.88 | 115.63 | 125.00 | 0.00 | |
| pga 200 | 77296.5300 | 1409.9456 | 437.98 | 59.25 | 200.00 | 0.00 | |
| pga var (max 400) | 75356.3262 | 3188.9325 | 436.58 | 70.43 | 264.53 | 90.64 | |

Best fitness averaged over last 100 generations, average run time

| Algorithm | Best fitness (std) | | Time in secs. (std) | |
|-------------------|--------------------|-----------|---------------------|--------|
| ga 125 | 70284.6355 | 2708.8702 | 1707.70 | 144.82 |
| pga 125 | 75111.9216 | 1611.4810 | 1827.13 | 71.71 |
| pga 200 | 76817.6350 | 1639.0860 | 1699.38 | 103.92 |
| pga var (max 400) | 74763.5389 | 3440.6151 | 1415.70 | 193.92 |

Table B.20: PGA and GA on NASA ALSS simulator, optimizing mission productivity and duration with strategy Γ_7 . Average and standard deviation (std) over 40 runs.

| Best of run (BOR) fitness, generation found, genome length | | | | | | | |
|---|-------------------|-----------|------------------------|--------|---------------------|--------|--|
| Algorithm | BOR fitness (std) | | Generation Found (std) | | Genome Length (std) | | |
| ga 175 | 72626.8411 | 3526.9798 | 328.03 | 110.77 | 175.00 | 0.00 | |
| pga 175 | 78331.2076 | 1798.1873 | 467.20 | 55.36 | 175.00 | 0.00 | |
| pga 280 | 77825.5538 | 1973.0107 | 472.43 | 38.23 | 280.00 | 0.00 | |
| pga var (max 560) | 75276.4758 | 2613.9920 | 479.35 | 44.15 | 407.13 | 100.06 | |

Best fitness averaged over last 100 generations, average run time

| Algorithm | Best fitness (std) | | Time in secs. (std) | |
|-------------------|--------------------|-----------|---------------------|--------|
| ga 175 | 72415.3248 | 3605.7672 | 1533.33 | 190.51 |
| pga 175 | 78019.0782 | 1840.7015 | 1592.85 | 100.70 |
| pga 280 | 77286.0961 | 2069.4718 | 1404.23 | 141.19 |
| pga var (max 560) | 74222.7373 | 2734.9842 | 1236.35 | 138.20 |

Table B.21: PGA and GA on NASA ALSS simulator, optimizing mission productivity and duration with strategy Γ_9 . Average and standard deviation (std) over 40 runs.

| Best of run (BOR) fitness, generation found, genome length | | | | | | | |
|---|-------------------|-----------|------------------------|-------|---------------------|--------|--|
| Algorithm | BOR fitness (std) | | Generation Found (std) | | Genome Length (std) | | |
| ga 255 | 72869.8208 | 1878.8131 | 404.88 | 90.70 | 225.00 | 0.00 | |
| pga 255 | 78860.5968 | 1695.8682 | 474.85 | 29.99 | 225.00 | 0.00 | |
| pga 360 | 77269.8081 | 2272.8554 | 470.58 | 25.69 | 360.00 | 0.00 | |
| pga var (max 720) | 74203.5236 | 2380.7273 | 460.00 | 34.59 | 529.63 | 119.87 | |

Best fitness averaged over last 100 generations, average run time

| Algorithm | Best fitness (std) | | Time in secs. (std) | |
|-------------------|--------------------|-----------|---------------------|--------|
| ga 255 | 72414.4872 | 1816.4348 | 1738.28 | 160.75 |
| pga 255 | 78319.7190 | 1666.1816 | 1805.83 | 84.50 |
| pga 360 | 76650.9385 | 2227.2262 | 1614.10 | 44.38 |
| pga var (max 720) | 73031.9897 | 2522.6191 | 1258.20 | 149.77 |

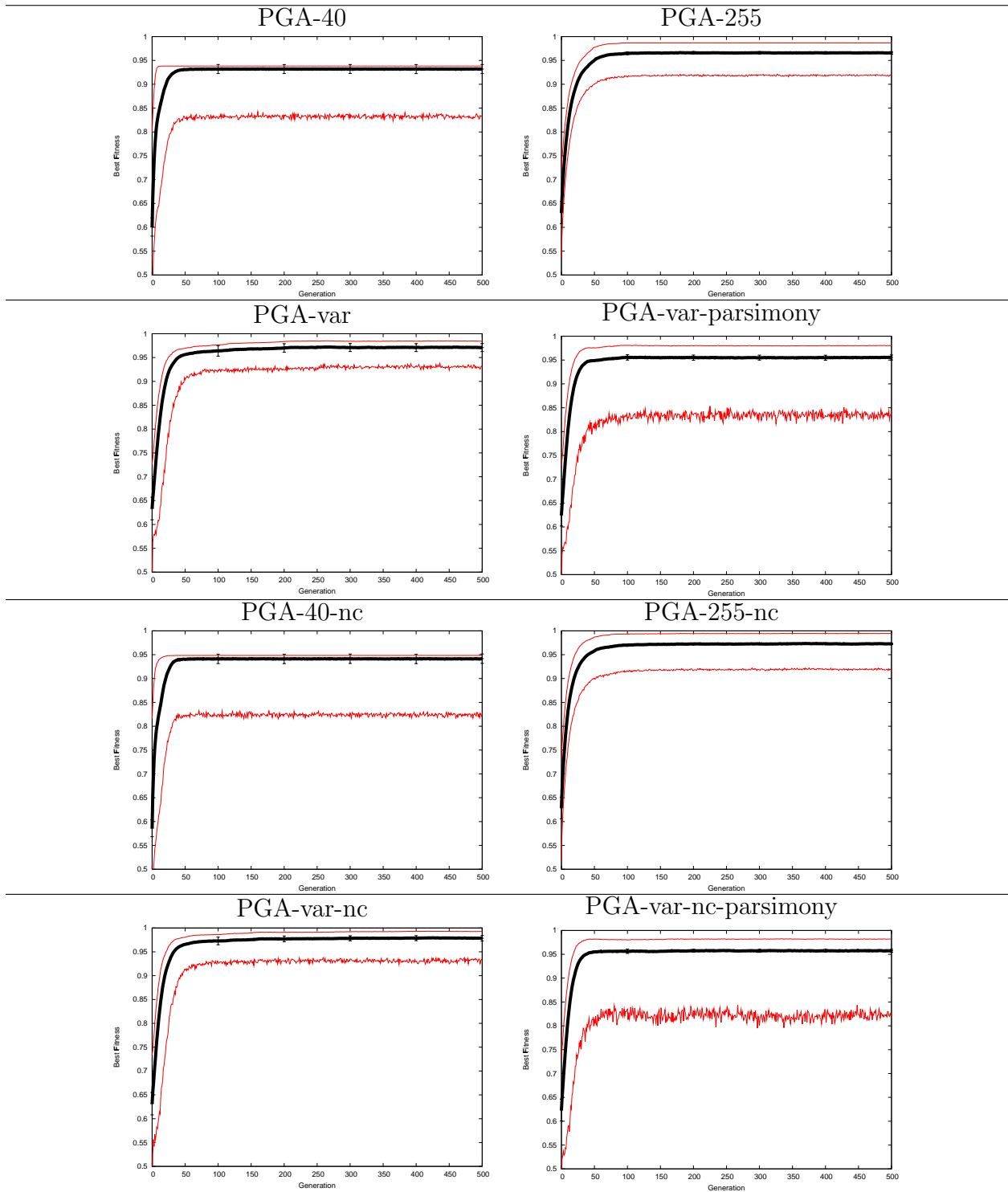


Figure B.1: Fitness data (best, average, worst) over 100 runs for the proportional GA on the resource allocation domain using randomly generated allocations

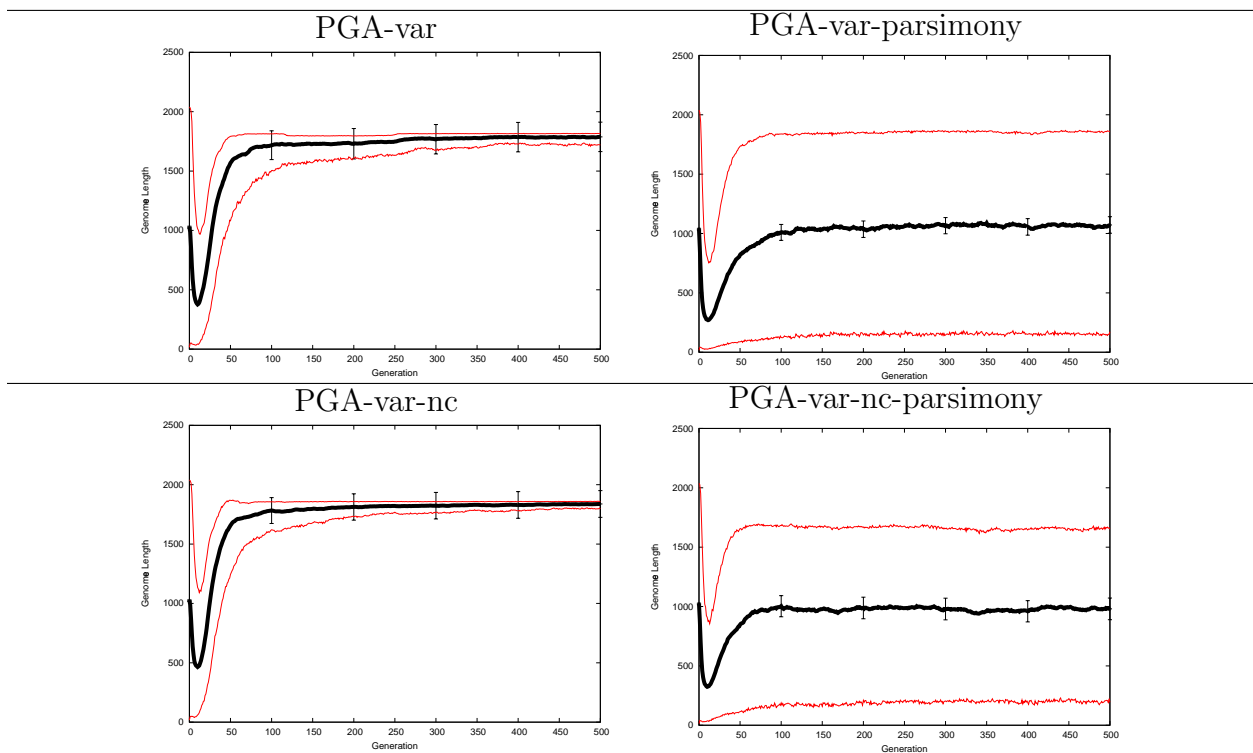


Figure B.2: Genome length data (longest, average, shortest) over 100 runs for the proportional GA on the resource allocation domain using randomly generated allocations

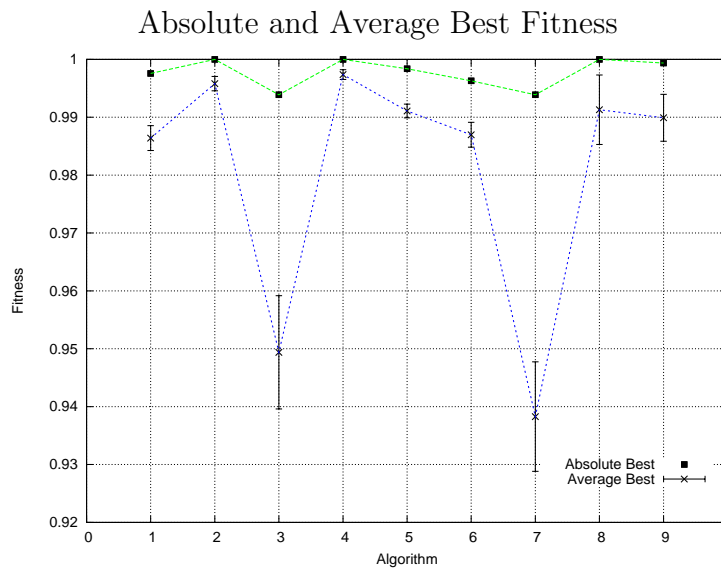
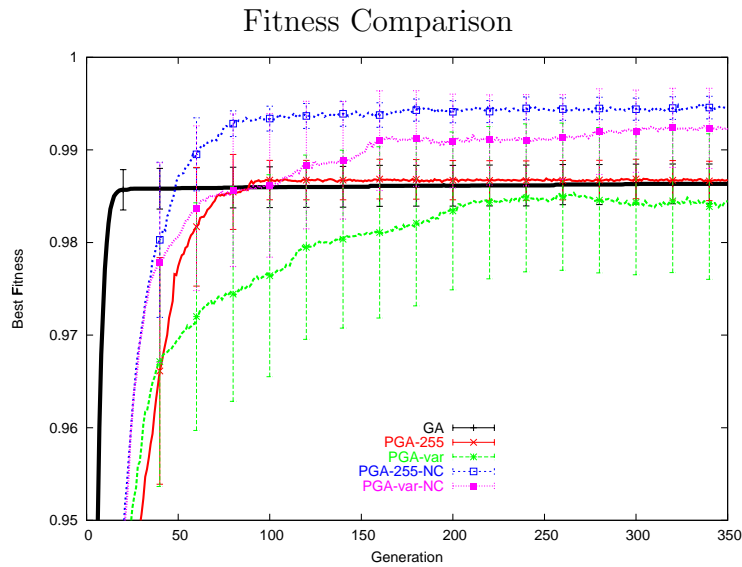


Figure B.3: Best fitness data for the resource allocation domain using randomly generated allocations: (top) close look at the best fitness per generation over 100 runs, comparing: GA, PGA-255, PGA-var, PGA-255-nc, and PGA-var-nc; (bottom) absolute and average best fitness over 100 runs for algorithms 1 to 9: GA, PGA-255-nc, PGA-40-nc, PGA-var-nc, PGA-var-nc-parsimony, PGA-255, PGA-40, PGA-var, PGA-var-parsimony

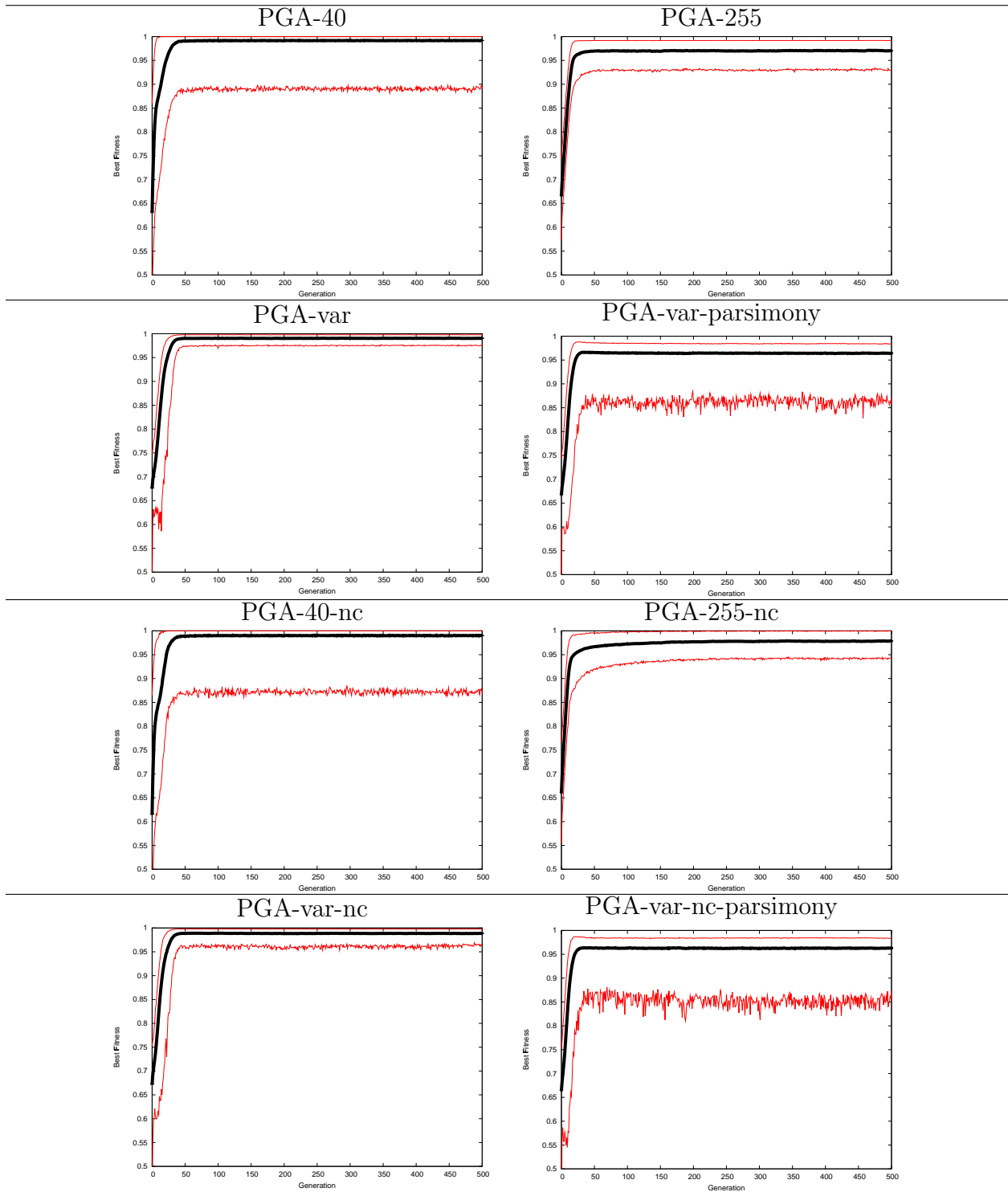


Figure B.4: Fitness data (best, average, worst) over 100 runs for the proportional GA on the resource allocation domain using fixed test allocation values: 1:2:4:2:1

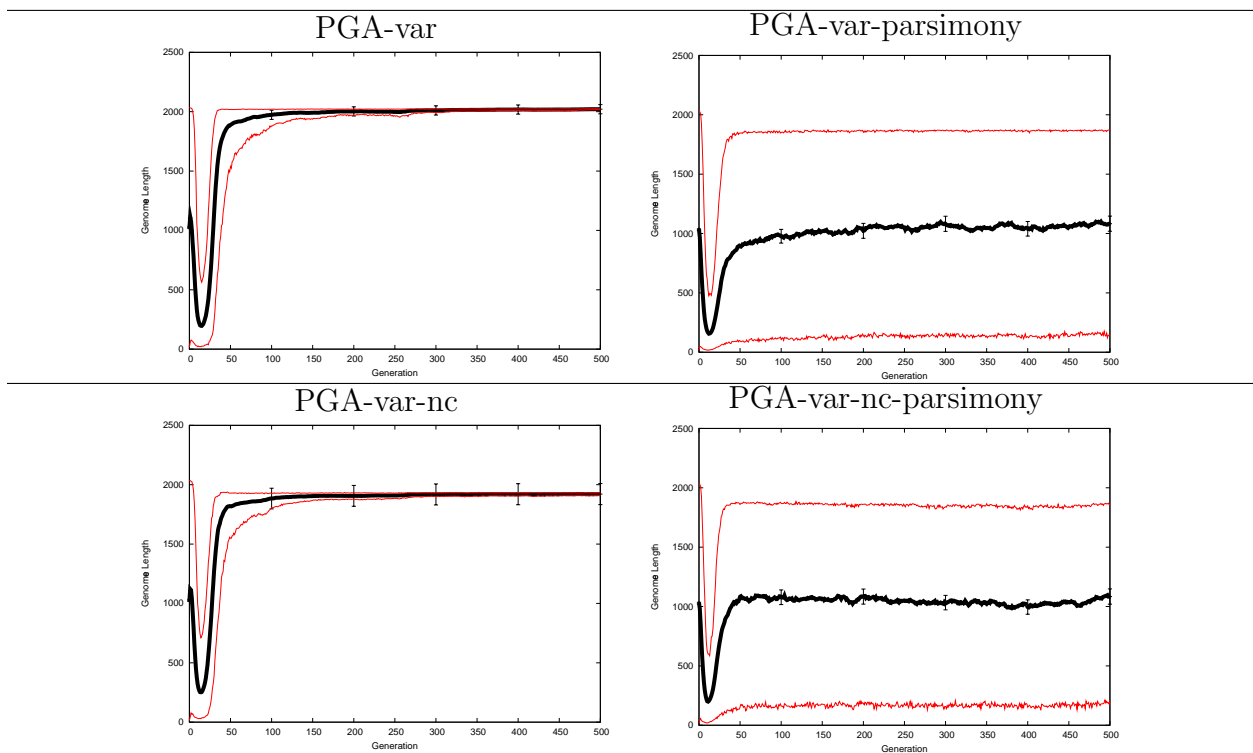


Figure B.5: Genome length data (longest, average, shortest) over 100 runs for the proportional GA on the resource allocation domain using fixed test allocation values: 1:2:4:2:1

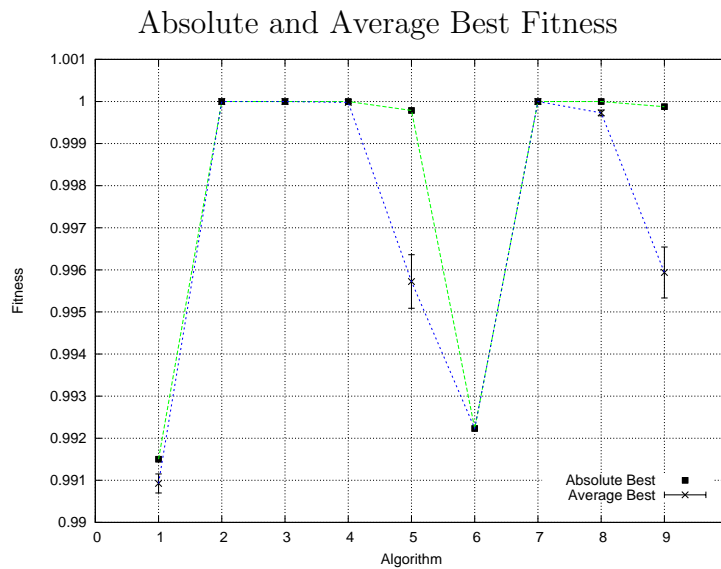
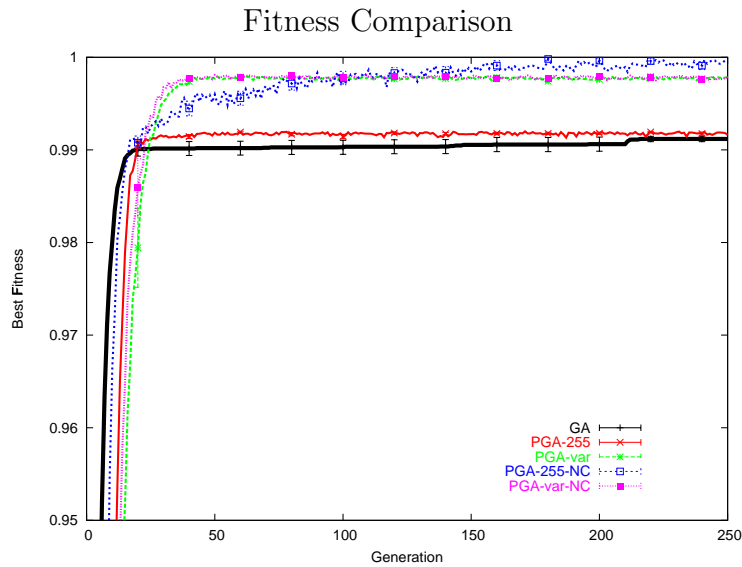


Figure B.6: Best fitness data for the resource allocation domain using fixed test allocation values: 1:2:4:2:1: (top) close look at the best fitness per generation over 100 runs, comparing: GA, PGA-255, PGA-var, PGA-255-nc, and PGA-var-nc; (bottom) absolute and average best fitness over 100 runs for algorithms 1 to 9: GA, PGA-255-nc, PGA-40-nc, PGA-var-nc, PGA-var-nc-parsimony, PGA-255, PGA-40, PGA-var, PGA-var-parsimony

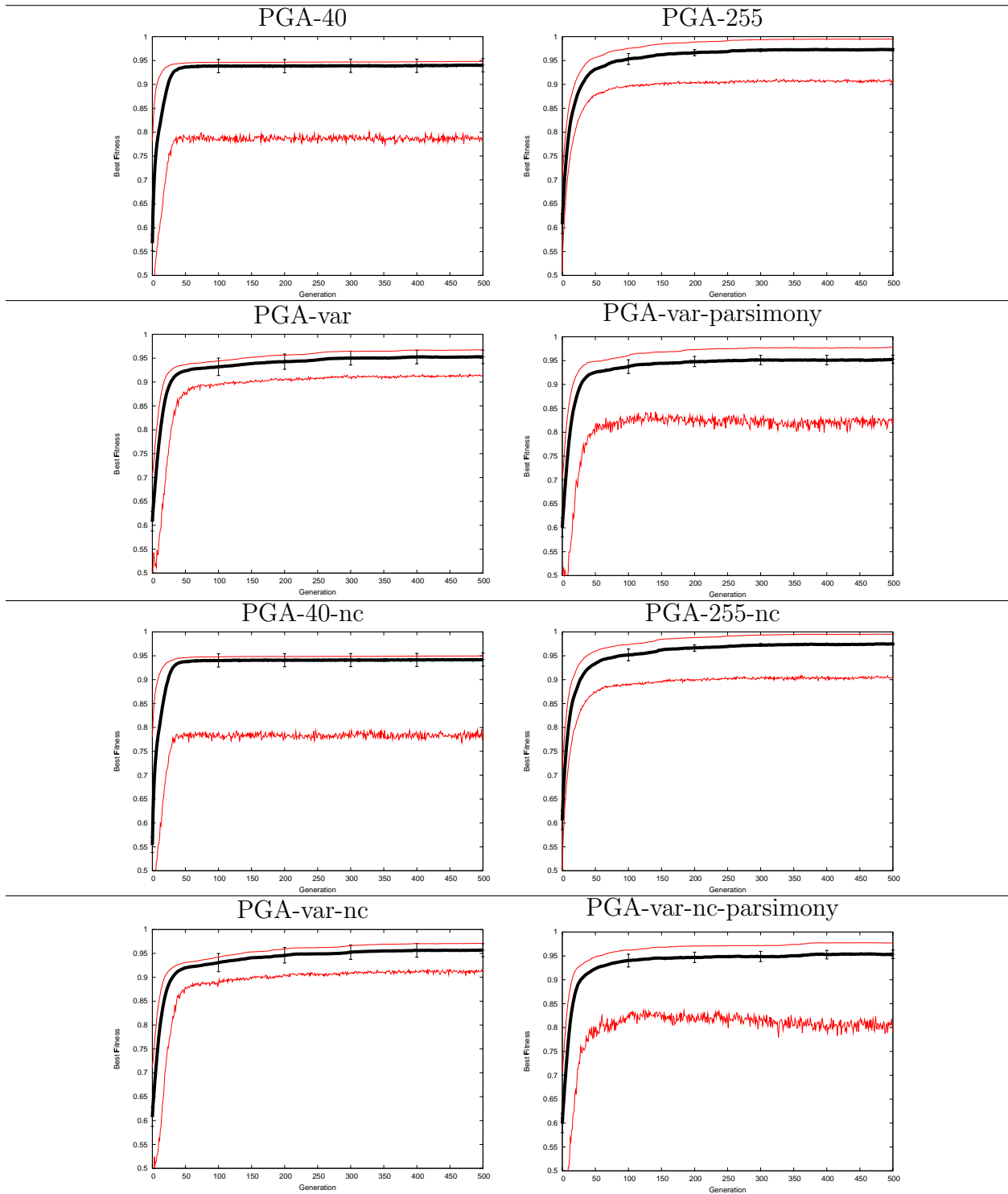


Figure B.7: Fitness data (best, average, worst) over 100 runs for the proportional GA on the number matching domain using PGA2

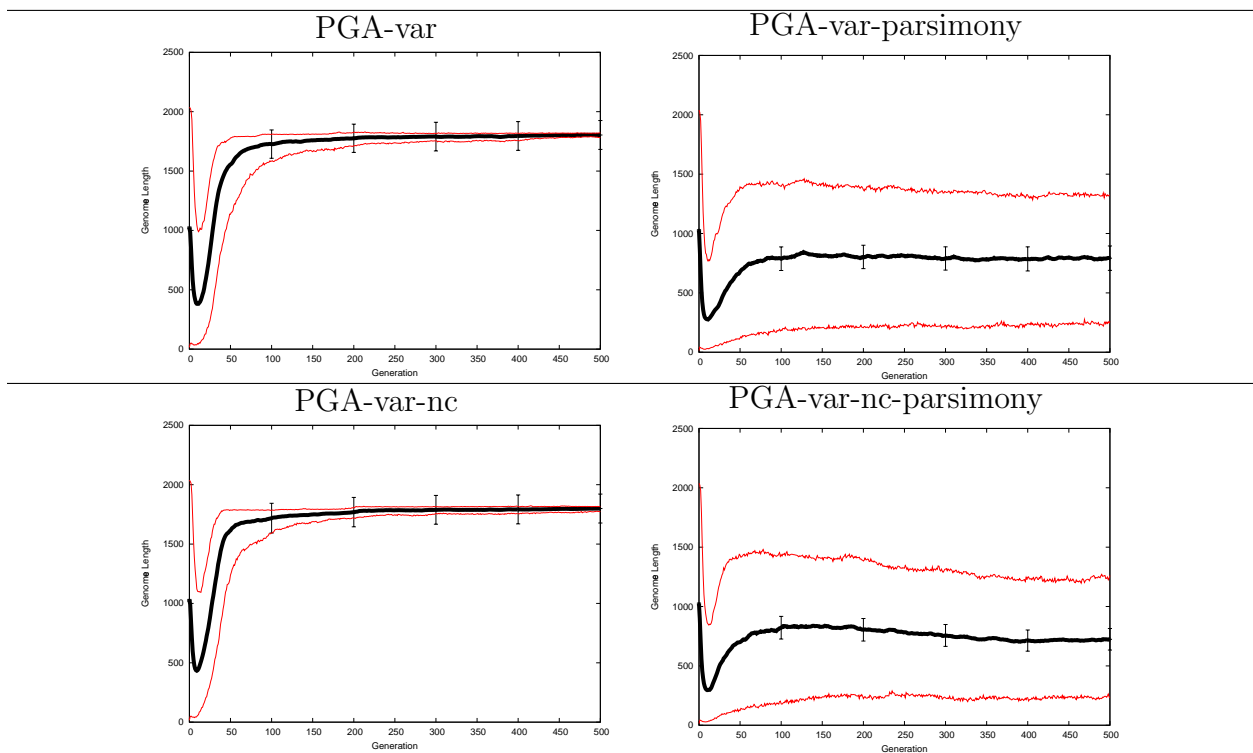


Figure B.8: Genome length data (longest, average, shortest) over 100 runs for the proportional GA on the number matching domain using PGA2

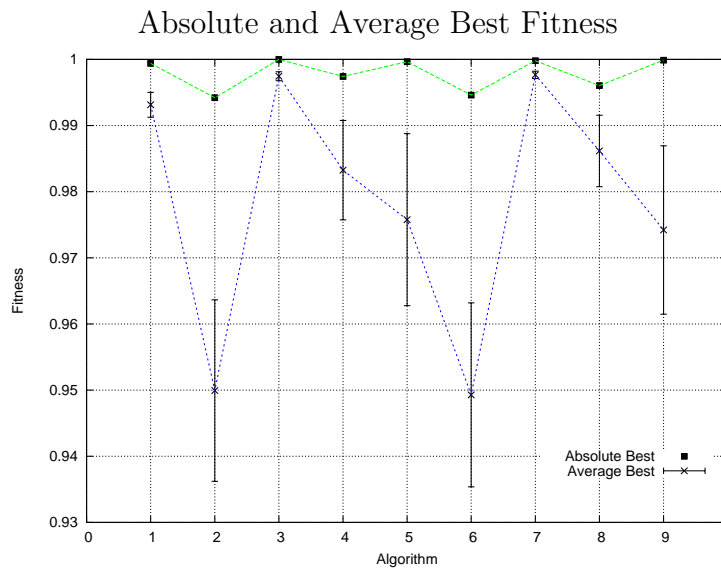
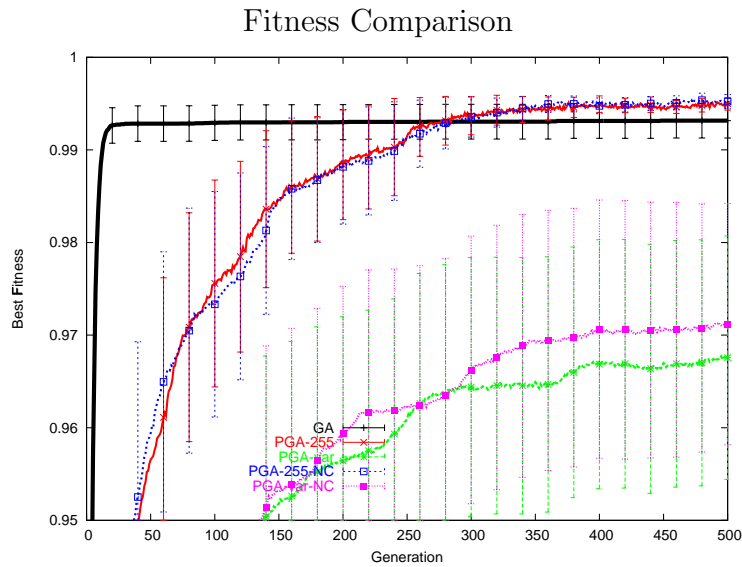


Figure B.9: Best fitness data for the number matching domain using PGA2: (top) close look at the best fitness per generation over 100 runs, comparing: GA, PGA-255, PGA-var, PGA-255-nc, and PGA-var-nc; (bottom) absolute and average best fitness over 100 runs for algorithms 1 to 9: GA, PGA-40-nc, PGA-255-nc, PGA-var-nc-parsimony, PGA-var-nc, PGA-40, PGA-255, PGA-var-parsimony, PGA-var

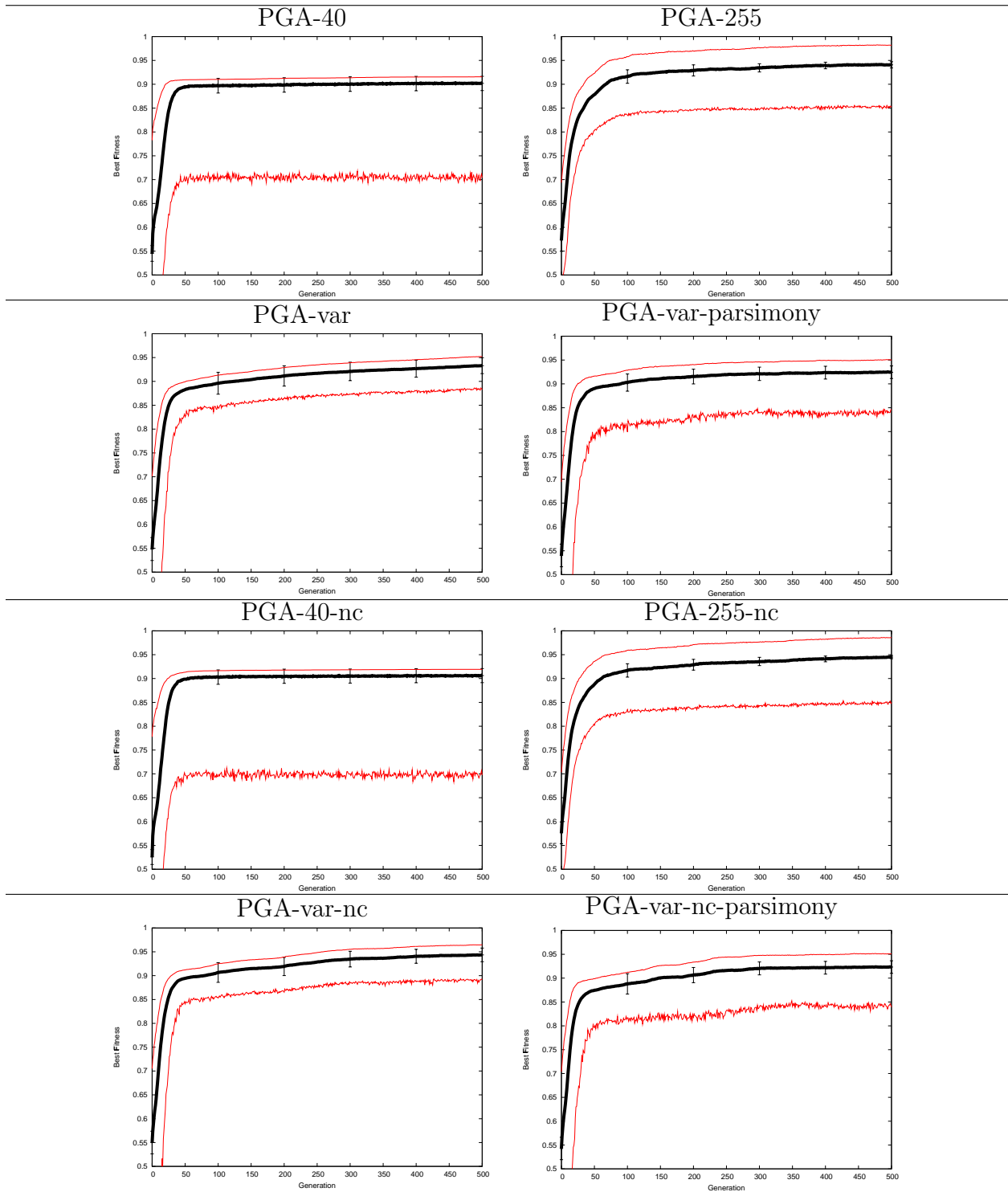


Figure B.10: Fitness data (best, average, worst) over 100 runs for the proportional GA on the number matching domain using PGA3

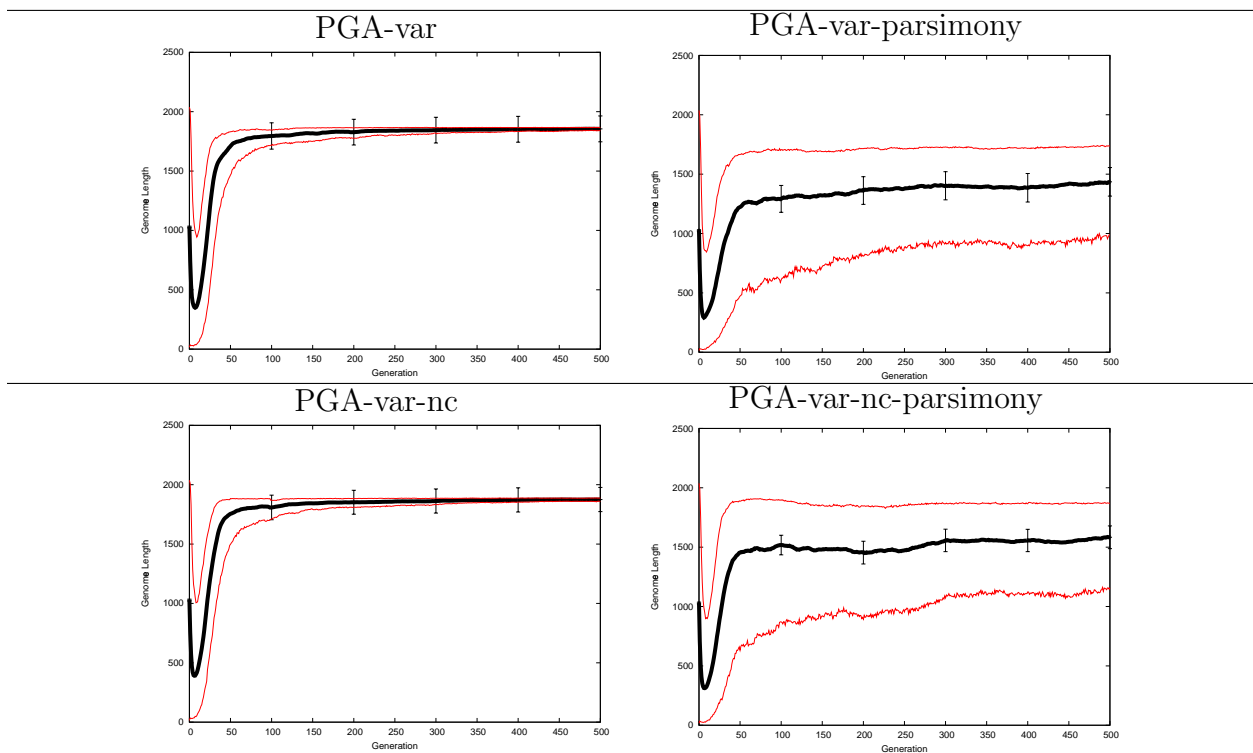


Figure B.11: Genome length data (longest, average, shortest) over 100 runs for the proportional GA on the number matching domain using PGA3

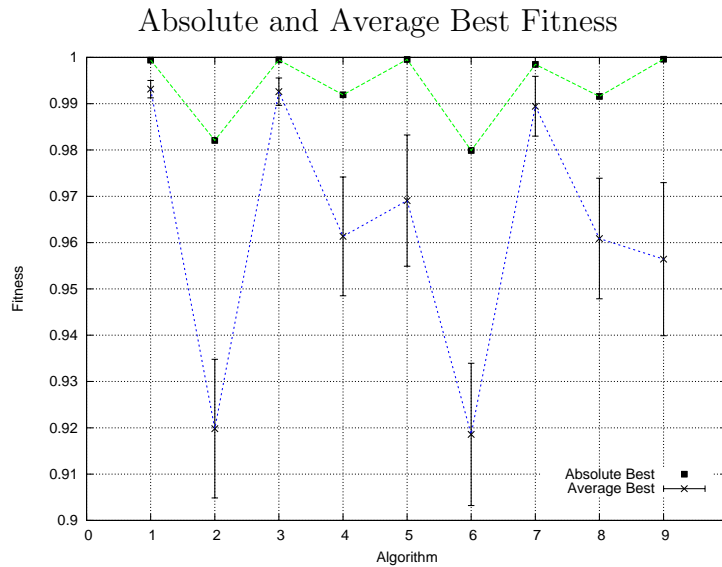
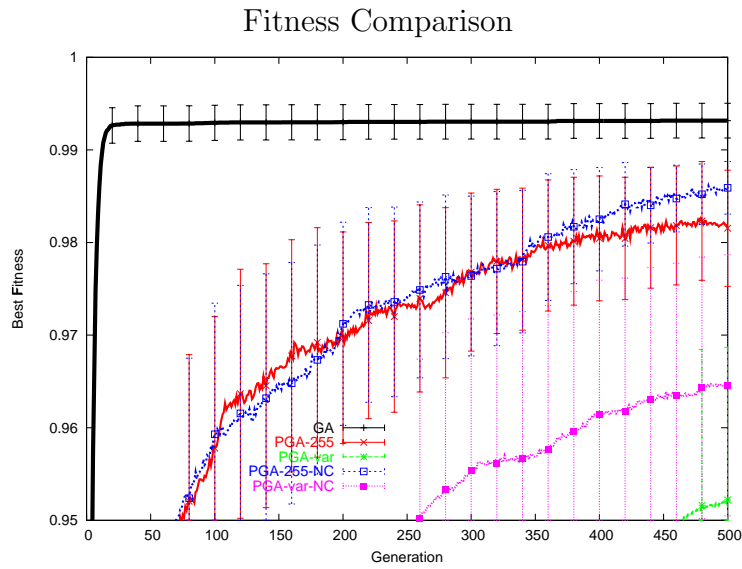


Figure B.12: Best fitness data for the number matching domain using PGA3: (top) close look at the best fitness per generation over 100 runs, comparing: GA, PGA-255, PGA-var, PGA-255-nc, and PGA-var-nc; (bottom) absolute and average best fitness over 100 runs for algorithms 1 to 9: GA, PGA-40-nc, PGA-255-nc, PGA-var-nc-parsimony, PGA-var-nc, PGA-40, PGA-255, PGA-var-parsimony, PGA-var

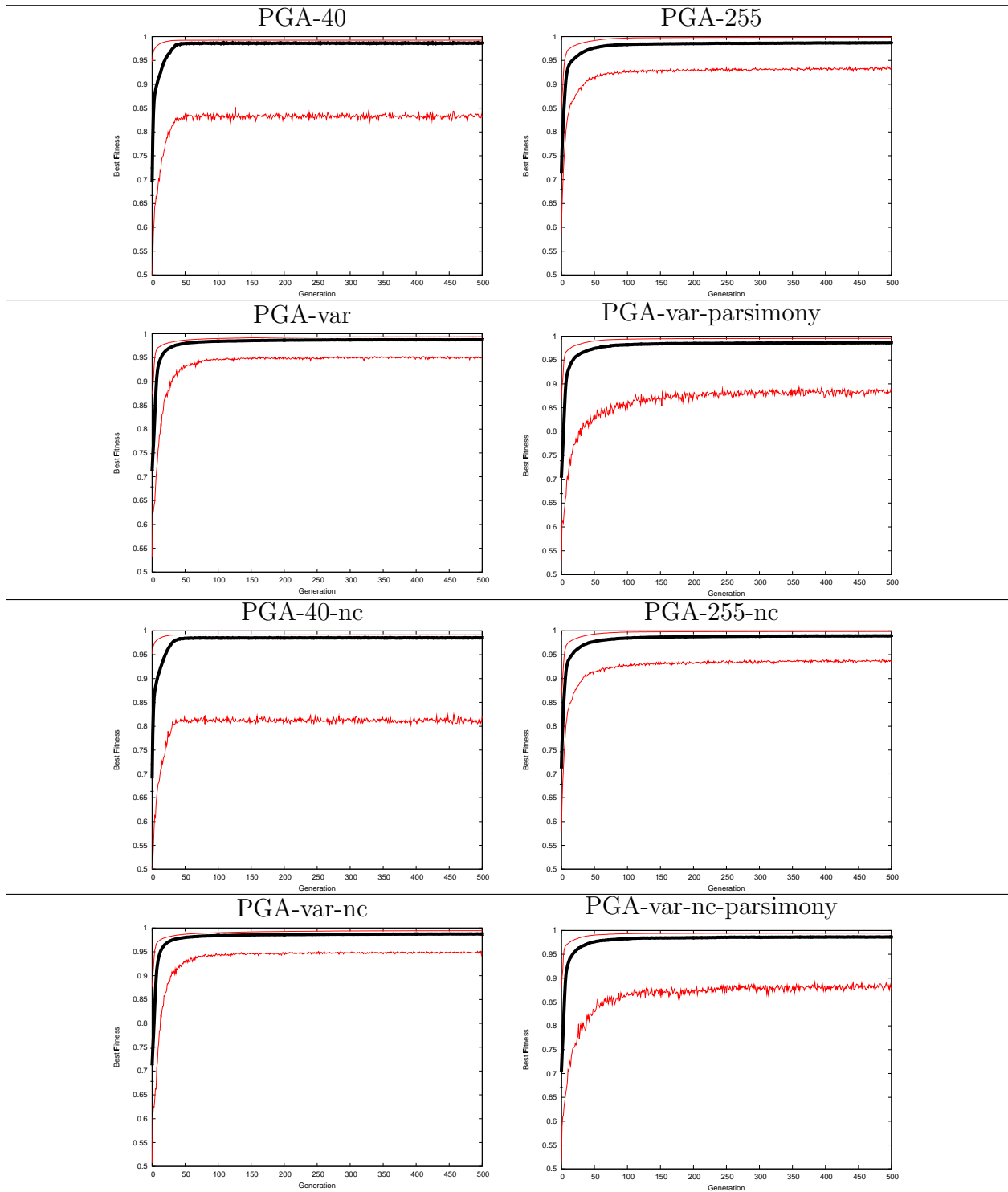


Figure B.13: Fitness data (best, average, worst) over 100 runs for the proportional GA on the symbolic regression domain using PGA2

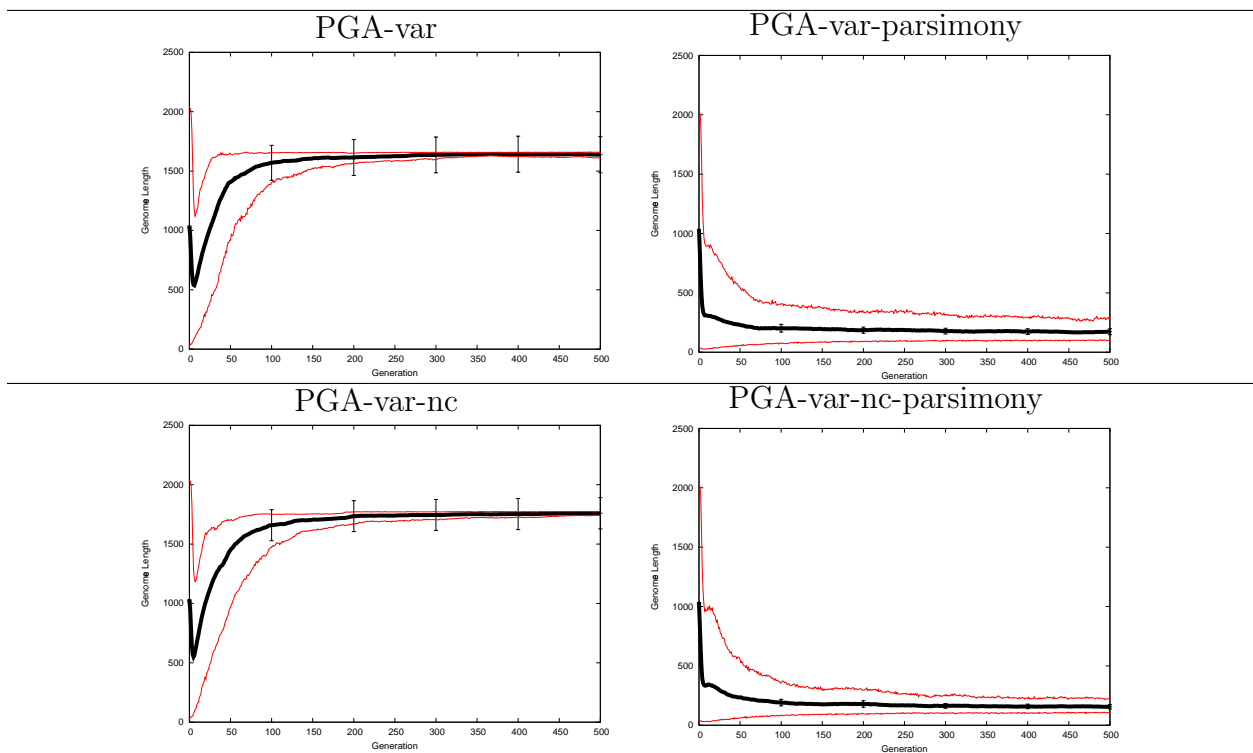


Figure B.14: Genome length data (longest, average, shortest) over 100 runs for the proportional GA on the symbolic regression domain using PGA2

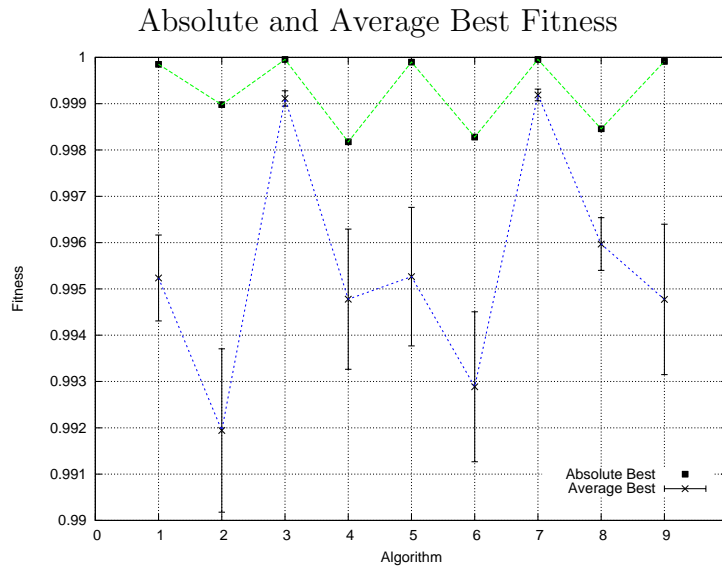
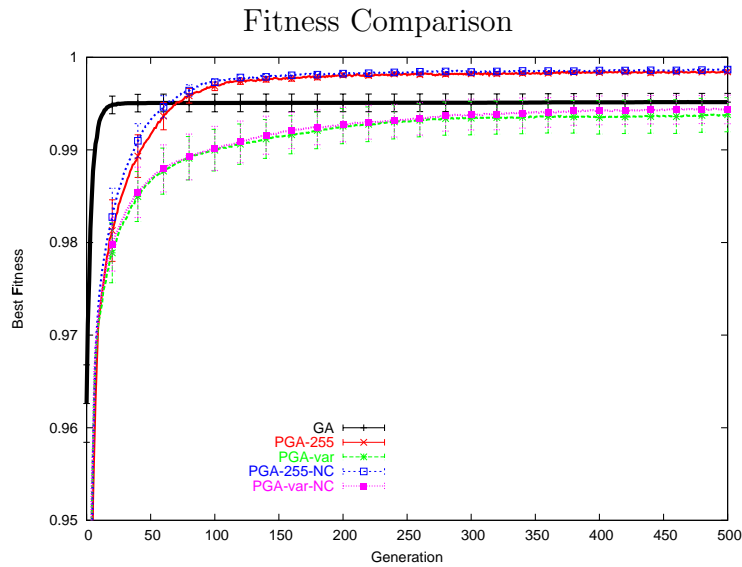


Figure B.15: Best fitness data for the symbolic regression domain using PGA2: (top) close look at the best fitness per generation over 100 runs, comparing: GA, PGA-255, PGA-var, PGA-255-nc, and PGA-var-nc; (bottom) absolute and average best fitness over 100 runs for algorithms 1 to 9: GA, PGA-40-nc, PGA-255-nc, PGA-var-nc-parsimony, PGA-var-nc, PGA-40, PGA-255, PGA-var-parsimony, PGA-var

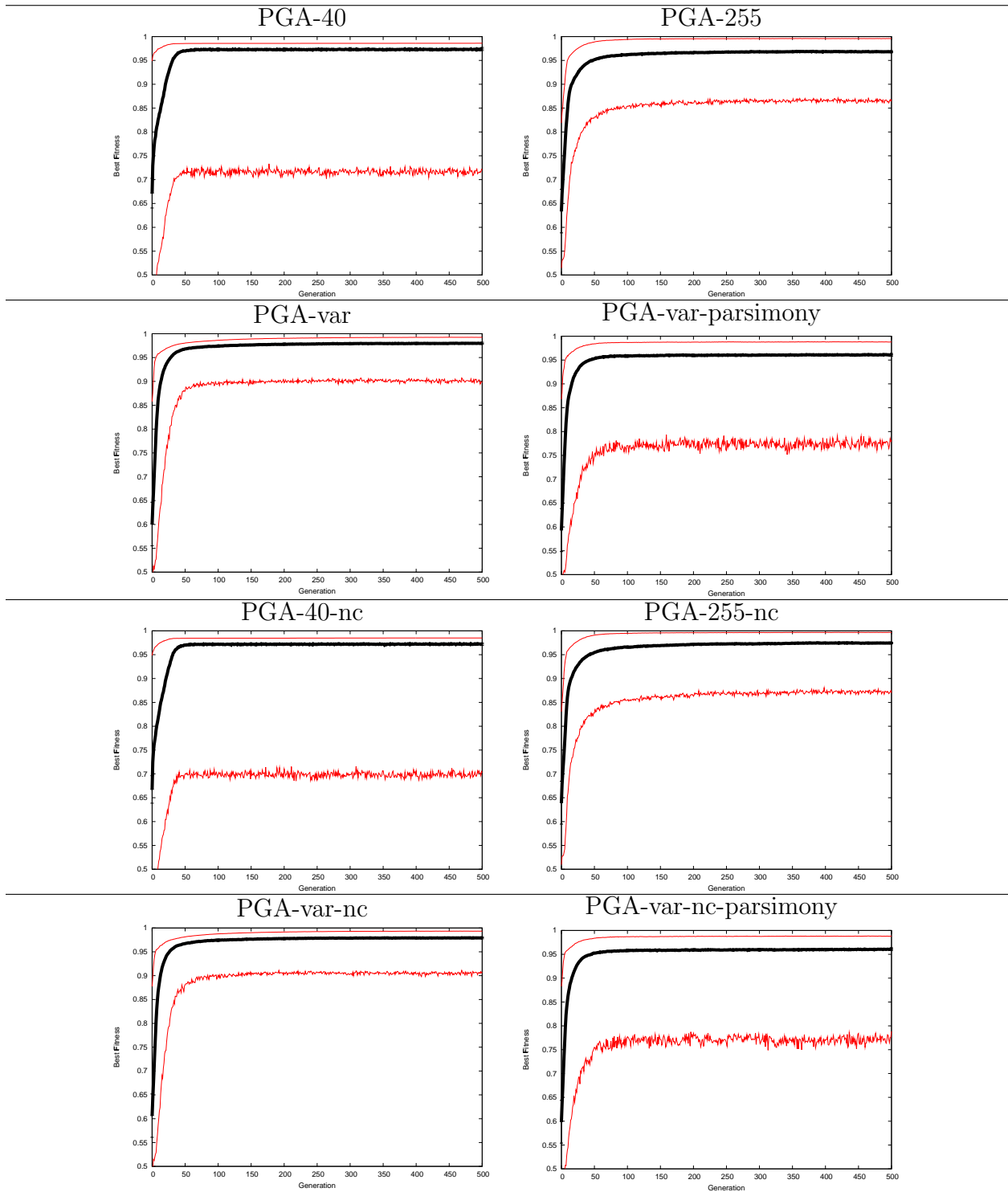


Figure B.16: Fitness data (best, average, worst) over 100 runs for the proportional GA on the symbolic regression domain using PGA3

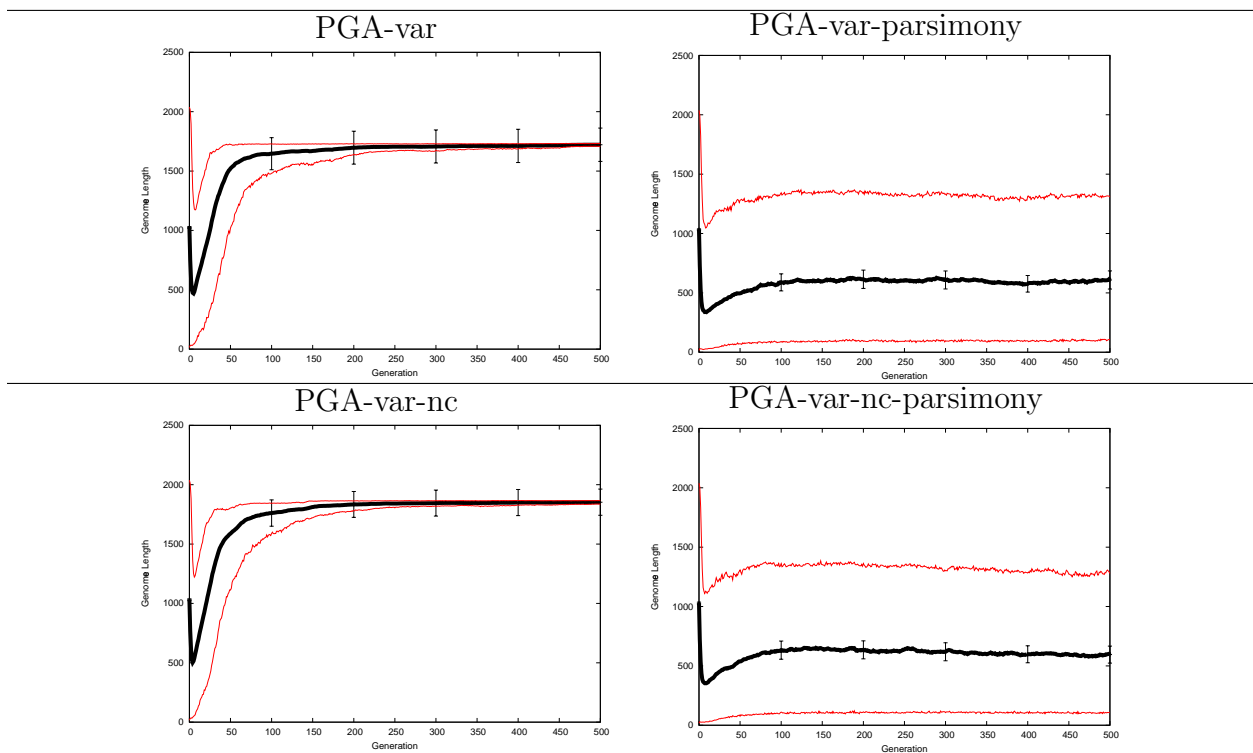


Figure B.17: Genome length data (longest, average, shortest) over 100 runs for the proportional GA on the symbolic regression domain using PGA3

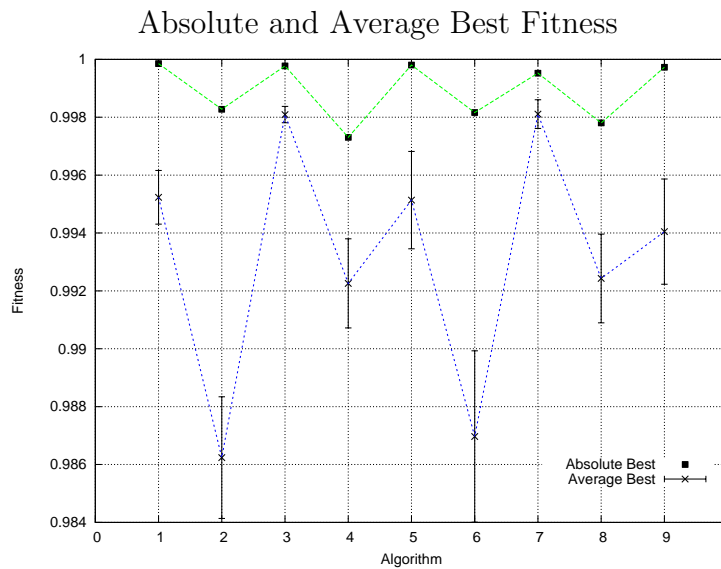
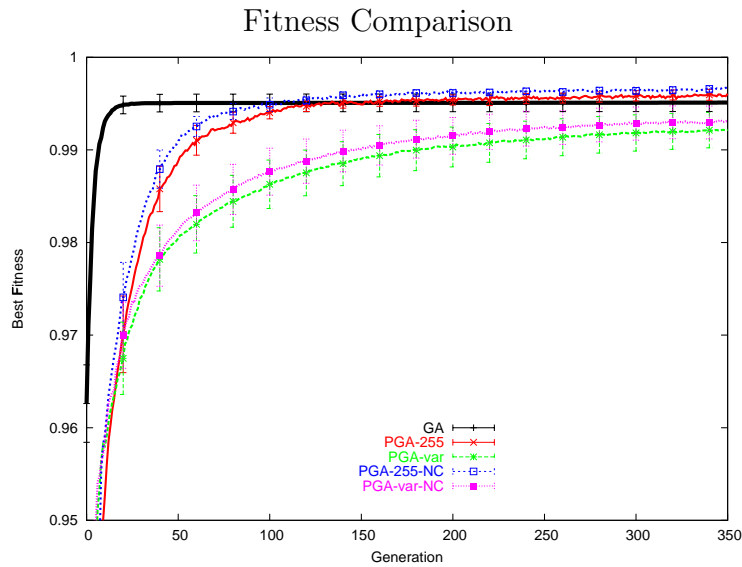


Figure B.18: Best fitness data for the symbolic regression domain using PGA3: (top) close look at the best fitness per generation over 100 runs, comparing: GA, PGA-255, PGA-var, PGA-255-nc, and PGA-var-nc; (bottom) absolute and average best fitness over 100 runs for algorithms 1 to 9: GA, PGA-40-nc, PGA-255-nc, PGA-var-nc-parsimony, PGA-var-nc, PGA-40, PGA-255, PGA-var-parsimony, PGA-var

LIST OF REFERENCES

- [ABB98] P. Abrahams, R. Balart, J.S. Byrnes, D. Cochran, M. J. Larkin, W. Moran, G. Os-theimer, and A. Pollington. “MAAP: the military aircraft allocation planner.” In *Proc. IEEE World Congress on Computational Intelligence*, pp. 336–341, 1998.
- [Ack87] D.H. Ackley. *A Connectionist Machine for Genetic Hillclimbing*. Kluwer Academic Publishers, Boston, 1987.
- [AJB00] Réka Albert, Hawoong Jeong, and Albert-László Barabási. “Error and attack tolerance in complex networks.” *Nature*, **406**:378–382, 2000.
- [AO02] John Atkins and Others. “The 22nd Amino Acid.” *Science*, **296**:1409–1410, 2002.
- [BA99] Albert-László Barabási and Réka Albert. “Emergence of Scaling in Random Networks.” *Science*, **286**:509–512, 1999.
- [Bag67] John D. Bagley. *The behavior of adaptive systems which employ genetic and correlation algorithms*. PhD thesis, University of Michigan, 1967.
- [Ban94] Wolfgang Banzhaf. “Genotype-Phenotype-Mapping and Neutral Variation – A case study in Genetic Programming.” In Yuval Davidor, Hans-Paul Schwefel, and Reinhard Männer, editors, *Parallel Problem Solving from Nature III*, pp. 322–332, Jerusalem, 9-14 1994. Springer-Verlag.
- [Bar98] Lionel Barnett. “Ruggedness and Neutrality – The NKp family of Fitness Landscapes.” In Christoph Adami, Richard K. Belew, Hiroaki Kitano, and Charles Taylor, editors, *Artificial Life VI: Proc. of the Sixth Int. Conf. on Artificial Life*, pp. 18–27, Cambridge, MA, 1998. The MIT Press.
- [BCF99] Daniel J. Barta, Juan M. Castillo, and Russ E. Fortson. “The biomass production system for the Bioregenerative Planetary Life Support Systems Test Complex: Preliminary designs and considerations.” In *Proc. 29th International Conference on Environmental Systems*, 1999. SAE Paper 1999-01-2188.
- [BDE99] Wolfgang Banzhaf, Peter Dittrich, and Burkart Eller. “Selforganization in a system of binary strings with topological interactions.” *Physica D*, **125**:85–104, 1999.

- [BDG98] Donald S. Burke, Kenneth A. De Jong, John J. Grefenstette, Connie Loggia Ramsey, and Annie S. Wu. “Putting more genetics into genetic algorithms.” *Evolutionary Computation*, **6**(4):387–410, 1998.
- [BDR03] Albert-László Barabási, Zoltán Dezsö, Erzsébet Ravasz, Soon-Hyung Yook, and Zoltán Oltvai. “Scale-free and hierararchical structures in complex networks.” To appear in: *Stiges Proceedings on Complex Networks*, 2003.
- [BEK] Axel Bernal, Uy Ear, and Nikos Kyrpides. “Genomes OnLine Database (GOLD): a monitor of genome projects world-wide.” <http://wit.integratedgenomics.com/GOLD/>.
- [Ben03] Peter J. Bentley. “Natural Design by Computer.” In *Proceedings of the 2003 AAAI Spring Symposium: Computational Synthesis: From Basic Building Blocks to High Level Functionality*, pp. 1–2. American Association for Artificial Intelligence, AAAI Press, 2003. Technical Report SS-03-02.
- [Ben04] P. J. Bentley. “Fractal Proteins.” *Genetic Programming and Evolvable Machines Journal*, **5**:71–101, 2004.
- [Beu95] P. J. Beurton et al. “Gene concepts and evolution.” *Max-Plank-Institu für Wissenschaftsgeschichte*, **18**, 1995.
- [BFG97] R. Peter Bonasso, James Firby, Erann Gat, David Kortenkamp, David P. Miller, and Marc G. Slack. “Experiences with an Architecture for Intelligent, Reactive Agents.” *Journal of Experimental & Theoretical Artificial Intelligence*, **9**(2/3):237–256, 1997.
- [BFM97] Thomas Bäck, David B. Fogel, and Zbigniew Michalewicz, editors. *Handbook of Evolutionary Computation*. IOP Publishing Ltd and Oxford University Press, 1997.
- [Bru97] Ralf Bruns. “Scheduling.” In *Handbook of Evolutionary Computation*, chapter F1.5. IOP Publishing Ltd and Oxford University Press, 1997.
- [BSd00] S. Baglioni, D. Sorbello, C. da Costa Pereira, and A. G. B. Tettamanzi. “Evolutionary Multiperiod Asset Allocation.” In *Proc. Genetic and Evolutionary Computation Conference*, pp. 597–604, 2000.
- [BYF62] Ernest Beutler, Mary Yeh, and Virgil F. Fairbanks. “The normal human female as a mosaic of X-chromosome activity: Studies using the gene for G-6-PD-Deficiency as a marker.” *Proc. Nat’l Acad. Sci.*, **48**(1):9–16, 1962.
- [CD02] Marie E. Csete and John C. Doyle. “Reverse Engineering of Biological Complexity.” *Science*, **295**:1664, 2002.

- [CLR98] David Cousins, Jackson Loomis, Fred Roeber, Pam Schoeppner, and Anne-Elise Tobin. “The embedded genetic allocator – A system to automatically optimize the use of memory resources in high performance, scalable computing systems.” In *Proc. IEEE Int’l Conference on Systems, Man, and Cybernetics*, volume 3, pp. 2166–2171, 1998.
- [CNL99] A. R. Callaghan, A. R. Nair, and K. E. Lewis. “A genetic algorithm based method for optimal resource allocation: A case study of the Buffalo Niagara International Airport expansion.” In *Proc. 3rd World Congress of Structural and Multidisciplinary Optimization*, 1999.
- [Con00] Michael Conrad. “Computation: Evolutionary, Neural, Molecular.” In *2000 IEEE Symposium on Combinations of Evolutionary Computation and Neural Networks*, pp. 1–9, 2000.
- [CPF00] Sekou S. Crawford, Christopher W. Pawlowski, and Cory K. Finn. “Power management in regenerative life support systems using market-based control.” In *Proc. ICES*, 2000.
- [CS92] Peter C-H. Cheng and Herbert A. Simon. “The right representtion for discovery: Finding the conservation of momentum.” In *Proc. 9th Int’l Workshop on Machine Learning*, pp. 62–71, 1992.
- [Cur83] Helena Curtis. *Biology*. Worth Publishers, 1983.
- [Dar59] Charles Darwin. *The Origin of Species*. J. M. Dent and Sons Ltd, 1975, 1859.
- [Dav91] Lawrence Davis. “Bit-Climbing, Representational Bias, and Test Suite Design.” In L. Booker and R. Belew, editors, *4th Int’l. Conf. on GAs*, pp. 18–23. Morgan Kaufmann, 1991.
- [DM83] Thomas G. Dietterich and Ryszard S. Michalski. “A comparative review of selected methods for learning from examples.” In *Machine Learning*, pp. 41–81. Morgan Kaufmann, 1983.
- [DM92] D. Dasgupta and D. R. MacGregor. “Nonstationary function optimization using the structured genetic algorithm.” In R. Manner and B. Manderick, editors, *Parallel Problem Solving from Nature 2*, pp. 145–154, 1992.
- [ECS89] Larry J. Eshelman, Richard A. Caruana, and J. David Schaffer. “Biases in the crossover landscape.” In J. D. Schaffer, editor, *Proc. 3rd Int’l Conference on Genetic Algorithms*, pp. 10–19, 1989.
- [ELA01] Marc Ebner, Patrick Langguth, Jurgen Albert, Mark Shackleton, and Rob Shipman. “On Neutral Networks and Evolvability.” In *Proceedings of the 2001 Congress on Evolutionary Computation*, pp. 1–8. IEEE Press, 2001.

- [EO02] Wolfgang Enard and Others. “Intra and Interspecific Variation in Primate Gene Expression Patterns.” *science*, **296**:340–343, 2002. Brains of humans and primates differ on quantity of genes expressed not in quality.
- [FDN59] R. M. Friedberg, B. Dunham, and J. H. North. “A Learning Machine: Part II.” *IBM Journal*, **3**:282–287, 1959.
- [Fin99] Cory K. Finn. “Dynamic system modeling of regenerative life support systems.” Technical Report 1999-01-2040, NASA Ames Research Center, 1999.
- [FO02] Asao Fujiyama and Others. “Construction and Analysis of a Human-Chimpanzee Comparative Clone Map.” *Science*, **295**(5552):131–134, January 2002.
- [Fog62] L. J. Fogel. “Autonomous Automata.” *Industrial Research*, **4**:14–19, 1962.
- [Fog64] L. J. Fogel. *On the Organization of Intellect*. PhD thesis, University of California at Los Angeles, 1964.
- [Fra57] A.S. Fraser. “Simulation of genetic systems by automatic digital computers.” *Australian Journal of Biological Sciences*, **10**:484–499, 1957.
- [Fra72] Daniel R. Frantz. *Non-linearities in genetic adaptive search*. PhD thesis, University of Michigan, 1972.
- [Fri58] R. M. Friedberg. “A Learning Machine: Part I.” *IBM Journal*, **2**:2–13, 1958.
- [FT01] D. H. Fleisher and K. C. Ting. “Modeling and control of plant production for advanced life support.” In *Acta Horticulturae: 4th International Symposium on Models for Plant Growth and Control in Greenhouses*, 2001. *Model-based predictive controller adjusts light intensity, air temperature, and CO2 concentration setpoints in response to system disturbances*.
- [FWM00] Robert W. Franceschini, Annie S. Wu, and Amar Mukherjee. “Computational strategies for disaggregation.” In *Proc. 9th Conf. on Computer Generated Forces and Behavioral Representation*, 2000.
- [GBB03] Max Garzon, Derrel Blain, Kiran Bobba, Andrew Neel, and Mitchael West. “Self-Assembly of DNA-like Structures in Silico.” *Genetic Programming and Evolvable Machines*, **4**(2):185–200, 2003.
- [GDK93] D. E. Goldberg, K. Deb, H. Kargupta, and G. Harik. “Rapid accurate optimization of difficult problems using fast messy genetic algorithms.” In Stephanie Forrest, editor, *Proc. 5th Int’l Conference on Genetic Algorithms*, pp. 56–64, 1993.

- [GKD89] David E. Goldberg, Bradley Korb, and Kalyanmoy Deb. “Messy Genetic Algorithms: Motivation, Analysis, and First Results.” *Complex Systems*, **3**:493–530, 1989.
- [GO02] Stephen A. Goff and Others. “A Draft Sequence of the Rice Genome (*Oryza sativa* L. ssp. *japonica*).” *Science*, **296**:92–100, 2002.
- [Gol89] David E. Goldberg. *Genetic algorithms in search, optimization, and machine learning*. Addison Wesley, 1989.
- [GRS90] J.J. Grefenstette, C. L. Ramsey, and A. C. Schultz. “Learning sequential decision rules using simulation models and competition.” *Machine Learning*, **5**(4):355–381, 1990.
- [GT99] Sara Goudarzi and K. C. Ting. “Top level modeling of crew component of ALSS.” In *Proc. 29th International Conference on Environmental Systems*, 1999.
- [GW03] Ivan I. Garibay and Annie S. Wu. “Advanced Life Support System Simulation.” Technical Report CS-TR-03-04, University of Central Florida, 2003.
- [GW04] Ivan I. Garibay and Annie S. Wu. “Emergent white noise behavior in location independent representations.” In *Proc. GECCO 2004 Workshop on Self-organization in Representations for Evolutionary Algorithms: Building complexity from simplicity*, 2004.
- [Hao02] B. Hao et al. “A New UAG-Encoded Residue in the Structure of a Methanogen Methyltransferase.” *Science*, **296**:1462–1466, 2002.
- [Har97] Georges R. Harik. *Learning gene linkage to efficiently solve problems of bounded difficulty using genetic algorithms*. PhD thesis, University of Michigan, 1997.
- [HHG00] Leland H. Hartwell, Leroy Hood, Michael L. Goldberg, Ann E. Reynolds, Lee M. Silver, and Ruth C. Veres. *Genetics: From Genes to Genomes*. McGraw-Hill, 2000.
- [HHL99] Leland H. Hartwell, John J. Hopfield, Stanislas Leibler, and Andrew W. Murray. “From molecular to modular cell biology.” *Nature*, **402**:C47–C52, December 1999.
- [HLP03] G. S. Hornby, H. Lipson, and J. B. Pollack. “Generative Representations for the Automated Design of Modular Physical Robots.” *IEEE Transactions on Robotics and Automation*, 2003. In Press.
- [Hol62] J. H. Holland. “Outline for a logical theory of adaptive systems.” *ACM*, **9**:297–314, 1962.
- [Hol75a] John H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, MI, 1975.

- [Hol75b] John H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, 1975.
- [Hol98] John H. Holland. *Emergence: From Chaos to Order*. Addison-Wesley, 1998.
- [Hor03a] Gregory S. Hornby. “Creating Complex Building Blocks through Generative Representation.” In *Proceedings of the 2003 AAAI Spring Symposium: Computational Synthesis: From Basic Building Blocks to High Level Functionality*, pp. 98–105. American Association for Artificial Intelligence, AAAI Press, 2003. Technical Report SS-03-02.
- [Hor03b] Gregory S. Hornby. “Generative Representations for Evolving Families of Designs.” In E. Cantú-Paz, J. A. Foster, K. Deb, D. Davis, R. Roy, U.-M. O’Reilly, H.-G. Beyer, R. Standish, G. Kendall, S. Wilson, M. Harman, J. Wegener, D. Dasgupta, M. A. Potter, A. C. Schultz, K. Dowsland, N. Jonoska, and J. Miller, editors, *Genetic and Evolutionary Computation – GECCO-2003*, volume 2724 of *LNCS*, pp. 1678–1689, Chicago, 12-16 July 2003. Springer-Verlag.
- [HSF96] Martijn A. Huynen, Peter Stadler, and Walter Fontana. “Smoothness within ruggedness: the role of neutrality in adaptation.” *Proc. Natl. Acad. Sci. USA*, **93**:397–401, 1996.
- [HT96] Inman Harvey and Adrian Thompson. “Through the Labyrinth Evolution Finds a Way: A Silicon Ridge.” In *First Int. Conf. on Evolvable Systems: From Biology to Hardware*, pp. 406–422. Springer-Verlag, 1996.
- [Hug99] Austing L. Hughes. *Adaptive Evolution of Genes and Genomes*. Oxford University Press, 1999. ISBN 0-19-511626-7; QH390.H84 1999 ;.
- [Huy96] Martijn A. Huynen. “Exploring phenotype space through neutral evolution.” *Journal of Molecular Evolution*, **43**:165–169, 1996.
- [JBT01] Hawoong Jeong, Albert-László Barabási, Bálint Tombor, and Zoltán N. Oltvai. “The global organization of cellular networks.” In *Computation of biochemical pathways and genetic networks workshop*, Heidelberg, 2001.
- [Jeo01] Hawoong Jeong et al. “Lethality and Centrality in protein networks.” *Nature*, **411**:41, 2001.
- [JTA00] Hawoong Jeong, Bálint Tombor, Réka Albert, Zoltán N. Oltvai, and Albert-László Barabási. “The large-scale organization of metabolic networks.” *Nature*, **407**:651–654, 2000.
- [Jul99] B. A. Julstrom. “Redundant genetic coding may not be harmful.” In W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith,

editors, *GECCO-99: Proceedings of the Genetic and Evolutionary Computation Conference*, volume 1, Orlando FL, 1999. Morgan Kaufman Publishers.

- [JW95] A. Juels and M. Wattenberg. “Hillclimbing as a Baseline Method for the Evaluation of Stochastic Optimization Algorithms.” In David S. Touretzky et al., editor, *Advances in Neural Information Processing Systems*, volume 8, pp. 430–436. MIT Press, 1995.
- [Kar96] Hillol Kargupta. “The gene expression messy genetic algorithm.” In *Proc. IEEE Int’l Conference on Evolutionary Computation*, pp. 814–819. IEEE Press, 1996.
- [Kar01] H. Kargupta. “A striking property of genetic code-like transformations.” *Complex Systems*, **11**, 2001.
- [KB99] R. E. Keller and W. Banzhaf. “Evolution of genetic code in genetic programming.” In W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith, editors, *Proc. Genetic and Evolutionary Computation Conference*, 1999.
- [KB01] Robert E. Keller and Wolfgang Banzhaf. “Evolution of Genetic Code on a Hard Problem.” In Lee Spector, Erik D. Goodman, Annie Wu, W.B. Langdon, Hans-Michael Voigt, Mitsuo Gen, Sandip Sen, Marco Dorigo, Shahram Pezeshk, Max H. Garzon, and Edmund Burke, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, pp. 50–56, San Francisco, California, USA, 7-11 July 2001. Morgan Kaufmann.
- [KB03a] David Kortenkamp and Scott Bell. “BioSim: An integrated simulaiaion of an advanced life support system for intelligent control research.” In *Proc. 7th International Symposium on Artificial Intelligence, Robotics, and Automation in Space*, 2003.
- [KB03b] Sanjeev Kumar and Peter J Bentley, editors. *On Growth, Form and Computers*. Academic Press, 2003.
- [KBS01] David Kortenkamp, R. Peter Bonasso, and Devika Subramanian. “Distributed, autonomous control of space habitats.” In *Proc. IEEE Aerospace Conf.*, 2001.
- [KC78] Motoo Kimura and J. F. Crow. “Effect of overall phenotypic selection on genetic change at individual loci.” *Proc Natl Acad Sci USA*, **75**:6168–6171, 1978.
- [KFA99] John R. Koza, Forrest H. Bennett III, David Andre, and Martin A. Keane. *Genetic Programming III: Darwinian Invention and Problem Solving*. Morgan Kaufmann Publishers, 1999. ISBN 1-55860-543-6.
- [Kim68] Motoo Kimura. “Evolutionary Rate at the Molucular Level.” *Nature*, **217**:624–626, 1968.

- [Kim77] Motoo Kimura. “Preponderance of synonymous changes as evidence for neutral theory of molecular evolution.” *Nature*, **267**:275–276, 1977.
- [Kim80] Motoo Kimura. “A simple method of estimating evolutionary rates of base substitutions through comparative studies of nucleotide sequences.” *J. Mol Evol*, **16**:111–120, 1980.
- [Kim81] Motoo Kimura. “Possibility of extensive neutral evolution under stabilizing selection with special reference to nonrandom usage of synonymous codons.” *Proc Natl Acad Sci USA*, **87**:5773–5777, 1981.
- [Kim83] Motoo Kimura. *The Neutral Theory of Molecular Evolution*. Cambridge University Press, Cambridge, 1983.
- [Kim87] Motoo Kimura. “Molecular evolutionary clock and the neutral theory.” *J Mol Evol*, **26**:24–33, 1987.
- [KKS03] John R. Koza, Martin A. Keane, Matthew J. Streeter, William Mydlowec, Jessen Yu, and Guido Lanza. *Genetic Programming IV: Routine Human-Competitive Machine Intelligence*. Kluwer Academic Publishers, 2003. ISBN 1-4020-7446-8.
- [Klo00] Joachim Klose. “Functional Proteomics.” In Sándor Suhai, editor, *Genomics and Proteomics: Functional and Computational Aspects*, pp. 107–120. Kluwer, 2000.
- [KO97] P. J. Kennedy and T. R. Osborn. “A model of gene expression and regulation in an artificial cellular organism.” *Complex Systems*, **11**, 1997.
- [KP01] H. Kargupta and B. H. Park. “Gene expression and fast construction of distributed evolutionary representation.” *Evolutionary Computation*, **9**(1):43–69, 2001.
- [KS97] Robert C. King and William D. Stansfield. *A Dictionary of Genetics*. Oxford University Press, fifth edition edition, 1997.
- [KSK03] John R. Koza, Matthew J. Streeter, and Martin A. Keane. “Automated Synthesis by Means of Genetic Programming of Human-Competitive Designs Employing Reuse, Hierarchies, Modularities, Development, and Parameterized Topologies.” In *Proceedings of the 2003 AAAI Spring Symposium: Computational Synthesis: From Basic Building Blocks to High Level Functionality*, pp. 138–145. American Association for Artificial Intelligence, AAAI Press, 2003. Technical Report SS-03-02.
- [Kum04] Sanjeev Kumar. “Multicellular Development, Self-Organization, and Differentiation.” In *Proc. GECCO 2004 Workshop on Self-organization in Representations for Evolutionary Algorithms: Building complexity from simplicity*, 2004.

- [Lan01] E. S. Lander. “Initial sequencing and analysis of the human genome.” *Nature*, **409**(6822):860–921, February 2001.
- [Lee02] Tong Ihn Lee et al. “Transcriptional Regulatory Networks in *Saccharomyces cerevisiae*.” *Science*, **298**:799–784, 2002.
- [Lew97] J. Lewis. “*A comparative study of diploid and haploid binary genetic algorithms.*”. Master’s thesis, University of Edinburgh, 1997.
- [Lie02] Daniel C. Liebler. *Introduction to Proteomics: tools for the new biology*. Humana Press, 2002.
- [LS87] Jill H. Larkin and Herbert A. Simon. “Why a diagram is (sometimes) worth ten thousand words.” *Cognitive Science*, **11**:65–99, 1987.
- [LT98] T. L. Lau and E. P. K. Tsang. “The guided genetic algorithm and its application to the generalized assignment problem.” In *10th IEEE Int’l Conference on Tools with AI*, pp. 336–343, 1998.
- [Men65] Gregor Mendel. *Experiments in plant hybridization*. ?, 1865.
- [MHF94] Melanie Mitchell, John H. Holland, and Stephanie Forrest. “When will a Genetic Algorithm Outperform Hill Climbing.” In Jack D. Cowan, Gerald Tesauro, and Joshua Alspecter, editors, *Advances in Neural Information Processing Systems*, volume 6, pp. 51–58. Morgan Kaufmann Publishers, Inc., 1994.
- [Mic98] Zbigniew Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*, chapter 10. Springer Verlag, 1998.
- [Mil02] R. Milo et al. “Network Motifs: Simple Building Blocks of Complex Networks.” *Science*, **289**:824–827, 2002.
- [Mor01] Michael Morange. *The Misunderstood Gene*. Harvard University Press, 2001.
- [MT04] J. Miller and P. Thomson. “Beyond the Complexity Ceiling: Evolution, Emergence and Regeneration.” In *Proc. GECCO 2004 Workshop on Regeneration and Learning in Developmental Systems*, 2004.
- [MW94] Keith Mathias and L. Darrell Whitley. “Transforming the search space with gray coding.” In *Proc. IEEE Int’l Conference on Evolutionary Computation*, pp. 513–518, 1994.
- [NE98] M. Newman and R. Engelhardt. “Effect of neutral selection on the evolution of molecular species.” In *Proceedings of the Royal Society of London, Series B: Biological Sciences*, volume 256, pp. 1333–1338, 1998. number 1403.

- [Nei87] M. Nei. *Molecular Evolutionary Genetics*. Columbia University Press, New York, 1987. cite found on Hughes:1999.
- [Nis97] Volker Nissen. “Management applications and other classical optimization problems.” In *Handbook of Evolutionary Computation*, chapter F1.2. IOP Publishing Ltd and Oxford University Press, 1997.
- [NW95] K. P. Ng and K. C. Wong. “A new diploid scheme and dominance change mechanism for non-stationary function optimization.” In L. J. Eshelman, editor, *Proc. 6th Int’l Conference on Genetic Algorithms*, 1995.
- [OB02] Zoltán Oltvai and Albert-László Barabási. “Life’s Complexity Pyramid.” *Science*, **298**:763–764, October 2002.
- [Ohn70] Susumu Ohno. *Evolution by gene duplication*. Springer-Verlag, 1970.
- [OR00] M. O’Neill and C. Ryan. “Grammar based function definition in grammatical evolution.” In D. Whitley, D. Goldberg, E. Cantu-Paz, L. Spector, I. Parmee, and H.-G. Beyer, editors, *Proc. Genetic and Evolutionary Computation Conference*, pp. 485–490, 2000.
- [OS94] Yuri Owechko and Soheil Shams. “Comparison of neural network and genetic algorithms for a resource allocation problem.” In *IEEE World Congress on Computational Intelligence*, volume 7, pp. 4655–4660, 1994.
- [Ott04] Tim Otter. “Toward a New Theoretical Framework for Biology.” In *Proc. GECCO 2004 Workshop on Self-organization in Representations for Evolutionary Algorithms: Building complexity from simplicity*, 2004.
- [Pal02] Timothy Palzkill. *Proteomics*. Kluwer Academic Publishers, 2002.
- [PD01] S. R. Pennington and M. J. Dunn. *Proteomics: from protein sequence to function*. BIOS Scientific Publishers, 2001.
- [PPT01] S. Papavassiliou, A. Puliafito, O. Tomarchio, and J. Ye. “Integration of mobile agents and genetic algorithms for efficient dynamics network resource allocation.” In *Proc. 6th IEEE Symposium on Computers and Communications*, pp. 456–463, 2001.
- [PPT02] S. Papavassiliou, A. Puliafito, O. Tomarchio, and J. Ye. “Mobile agent-based approach for efficient network management and resource allocation: framework and applications.” *IEEE Journal on Selected Areas in Communications*, **20**(4):858–872, 2002.
- [Pri76] D. J. DE S. Price. “A general theory of bibliometric and other cumulative advantage processes.” *Journal of the American Society for Information Science*, **27**:292–306, 1976.

- [PZS01] S. Palaniappan, S. Zein-Sabatto, and A. Sekmen. “Dynamic multiobjective optimization of war resource allocation using adaptive genetic algorithms.” In *Proc. IEEE SoutheastCon*, pp. 160–165, 2001.
- [Rad91] Nicholas J. Radcliffe. “Equivalence Class Analysis of Genetic Algorithms.” *Complex Systems*, **5**(2):183–205, 1991.
- [Rav02] E. Ravasz et al. “Hierarchical Organization of Modularity in Metabolic Networks.” *Science*, **297**:1551–1555, 2002.
- [RB99] Andreas Rechtsteiner and Mark A. Bedau. “A Generic Neutral Model for Measuring Excess Evolutionary Activity of Genotypes.” In Wolfgang Banzhaf, Jason Daida, Agoston E. Eiben, Max H. Garzon, Vasant Honavar, Mark Jakiela, and Robert E. Smith, editors, *Proc. of the Genetic and Evolutionary Computation Conf. GECCO-99*, pp. 1366–1373, San Francisco, CA, 1999. Morgan Kaufmann.
- [Rec65] I. Rechenberg. “Cybernetic Solution Path of an Experimental Problem.” Translation 1122, Royal Aircraft Establishment Library, 1965.
- [Rid00] Matt Ridley. *Genome*. Harper Collins Publishers, 2000.
- [Roc04] Luis Mateus Rocha. “Evolving Memory: Logical Tasks for Cellular Automata.” In *Proc. Ninth International Conference on the Simulation and Synthesis of Living Systems (ALIFE9)*. In Press, 2004.
- [Rot00] M. Rother et al. “Identification and Characterisation of the Selenocysteine-specific Translation Factor SelB from the Archaeon *Mathanococcus jannaschii*.” *Journal of Molecular Biology*, **299**:351–358, 2000.
- [Rys63] Herbert John Ryser. *Combinatorial Mathematics*. The mathematical association of america, 1963.
- [Sam94] Claude Sammut. “Knowledge representation.” In D. Michie, D. J. Spiegelhalter, and C. C. Taylor, editors, *Machine learning, neural and statistical classification*, pp. 228–245. Ellis Horwood, 1994.
- [SB01] Terence Soule and Amy E. Ball. “A genetic algorithm with multiple reading frames.” In L. Spector, E. D. Goodman, A. S. Wu, W. B. Langdon, H. M. Voigt, M. Gen, S. Sen, M. Dorigo, S. Pezeshk, M. H. Garzon, and E. Burke, editors, *Proc. Genetic and Evolutionary Computation Conference*, 2001.
- [Sco95] J. Scott. “A place in the world for RNA editing.” *Cell*, **81**:833–836, 1995.
- [SG92] R. E. Smith and D. E. Goldberg. “Diploidy and dominance in artificial genetic search.” *Complex Systems*, **6**(3):251–285, 1992.

- [Shi99] R. Shipman. “Genetic Redundancy: Desirable or Problematic for Evolutionary Adaptation.” In M. A. Bedau et al., editors, *Proceedings of the 4th International Conference on Artificial Neural Networks and Genetic Algorithms*, pp. 337–344. Springer-Verlag, 1999.
- [SHN99] Mahmoud R. Sherif, Ibrahim W. Habib, Mahmoud Nagshineh, and Parviz Kermani. “A generic bandwidth allocation scheme for multimedia substreams in adaptive networks using genetic algorithms.” In *Proc. Wireless Communications and Networking Conference*, volume 3, pp. 1243–1247, 1999.
- [SHN00] Mahmoud R. Sherif, Ibrahim W. Habib, Mahmoud Nagshineh, and Parviz Kermani. “Adaptive allocation of resources and call admission control for wireless ATM using genetic algorithms.” *IEEE Journal on Selected Areas Communications*, **18**(2):268–282, 2000.
- [SHO01] Tom Smith, Phil Husbands, and Michael O’Shea. “Neutral Networks in a Evolutionary Robotics Search Space.” In *Proceedings of the Congress on Evolutionary Computation*, volume 1, pp. 136–143. IEEE Press, 2001.
- [SMM02] S. S. Shen-Orr, R. Milo, S. Mangan, and U. Alon. “Network motifs in the transcriptional regulation network of *Escherichia coli*.” *Nature Genetics*, **31**:64–68, 2002.
- [SSE00a] Mark Shackleton, Rob Shipman, and Marc Ebner. “An investigation of redundant genotype-phenotype mappings and their role in evolutionary search.” In *Proc. Congress on Evolutionary Computation*, pp. 493–500, 2000.
- [SSE00b] Mark Shackleton, Rob Shipman, and Marc Ebner. “An investigation of redundant genotype-phenotype mappings and their role in evolutionary search.” In *Proc. of the 2000 Congress on Evolutionary Computation*, pp. 493–500, Piscataway, NJ, 2000. IEEE Service Center.
- [STB02] Debra Schreckenghost, Carroll Thronesbery, Peter Bonasso, David Kortenkamp, and Cheryl Martin. “Intelligent control of life support for space missions.” *IEEE Intelligent Systems*, pp. 24–31, September/October 2002.
- [Sys89] Gil Syswerda. “Uniform crossover in genetic algorithms.” In *Proc. 3rd Int’l Conference on Genetic Algorithms*, 1989.
- [TI02] Marc Toussaint and Christian Igel. “Neutrality: A Necessity for Self-Adaptation.” In *Proceedings of the Congress on Evolutionary Computation*, 2002. In Press.
- [Tri99] T. O. Tri. “Bioregenerative Planetary Life Support Systems Test Complex (BIO-Plex): Test mission objectives and facility development.”, 1999. SAE Paper 1999-01-2186, 29th Int’l Conference on Environmental Systems.

- [VM00] V. K. Vassilev and J. F. Miller. “Embedding landscape neutrality to build a bridge from the conventional to a more efficient three-bit multiplier circuit.” In D. Whitley, D. Goldberg, E. Cantu-Paz, L. Spector, I. Parmee, and H.-G. Beyer, editors, *Proc. Genetic and Evolutionary Computation Conference*, 2000.
- [VO01] J. Craig Venter and Others. “The Sequence of the Human Genome.” *Science*, **291**(5507):1304–1351, February 2001.
- [Vos93] Richard F. Voss. “1/f Noise and Fractals in DNA-base Sequences.” In Crilly, Earnshaw, and Jones, editors, *Applications of Fractals and Chaos*, pp. 7–20. Springer-Verlag, 1993.
- [Wat76] James D. Watson. *Molecular Biology of the Gene*. W. A. Benjamin, Inc., 1976.
- [WC95] Valerie C. Wasinger, Stuart J. Cordwell, et al. “Progress with gene-product mapping of the Mollicutes: *Mycoplasma genitalium*.” *Electrophoresis*, **16**:1090–1094, 1995.
- [WD99] Annie S. Wu and Kenneth A. De Jong. “An examination of building block dynamics in different representations.” In *Proc. Congress on Evolutionary Computation*, pp. 715–721, 1999.
- [WF02] Andreas Wagner and David Fell. “The Small World Inside Large Metabolic Networks.” Technical Report SFI working paper: 00-07-041, Santa Fe Institute, 2002.
- [WG02] Annie S. Wu and Ivan Garibay. “The proportional genetic algorithm: Gene expression in a genetic algorithm.” *Genetic Programming and Evolvable Hardware*, **3**(2):157–192, June 2002.
- [WGW92] James D. Watson, Michael Gilman, Jan Witkowski, and Mark Zoller. *Recombinant DNA*. Scientific American Books, 1992.
- [Wil00] M. Williams. “Making the best use of the airways: an important requirement for military communications.” *Electronics and Communication Engineering Journal*, pp. 75–83, 2000.
- [WL96] Annie S. Wu and Robert K. Lindsay. “A comparison of the fixed and floating building block representation in the genetic algorithm.” *Evolutionary Computation*, **4**(2):169–193, 1996.
- [WM95] David H. Wolpert and William G. Macready. “No Free Lunch Theorems for Search.” Technical Report SFI-TR-95-02-010, The Santa Fe Institute, Santa Fe, NM, 1995.

- [WM97] David H. Wolpert and William G. Macready. “No Free Lunch Theorems for Optimization.” *IEEE Transactions on Evolutionary Computation*, **1**(1):67–82, 1997.
- [WSA99] Annie S. Wu, Alan C. Schultz, and Arvin Agah. “Evolving control for distributed micro air vehicles.” In *Proc. IEEE Int’l Symp. Computational Intelligence in Robotics and Automation*, 1999.
- [Wuc02] Stefan Wuchty. “Interaction and domain networks of yeast.” *Proteomics*, **2**:1715–1723, 2002.
- [WW01] Karster Weicker and Nicole Weicker. “Burden and Benefits of Redundancy.” In Worthy N. Martin and William M. Spears, editors, *Foundations of Genetic Algorithms*, volume 6, pp. 313–333, 2001.
- [YA94] K. Yoshida and N. Adachi. “A diploid genetic algorithm for preserving population diversity.” In Y. Davidor, H.-P. Schwefel, and R. Manner, editors, *Parallel Problem Solving from Nature 3*, 1994.