

EXAPTATION AND FUNCTIONAL SHIFT IN EVOLUTIONARY COMPUTING

By

Lee Graham, B.C.S., M.C.S.

A thesis submitted to

The Faculty of Graduate Studies and Research

In partial fulfillment of

the requirements for the degree of

Doctor of Philosophy

Ottawa-Carleton Institute for Computer Science

School of Computer Science

Carleton University

Ottawa, Ontario

December 2008

© Copyright

2008, Lee Graham



Library and
Archives Canada

Published Heritage
Branch

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque et
Archives Canada

Direction du
Patrimoine de l'édition

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file *Votre référence*
ISBN: 978-0-494-47465-5
Our file *Notre référence*
ISBN: 978-0-494-47465-5

NOTICE:

The author has granted a non-exclusive license allowing Library and Archives Canada to reproduce, publish, archive, preserve, conserve, communicate to the public by telecommunication or on the Internet, loan, distribute and sell theses worldwide, for commercial or non-commercial purposes, in microform, paper, electronic and/or any other formats.

The author retains copyright ownership and moral rights in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

AVIS:

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque et Archives Canada de reproduire, publier, archiver, sauvegarder, conserver, transmettre au public par télécommunication ou par l'Internet, prêter, distribuer et vendre des thèses partout dans le monde, à des fins commerciales ou autres, sur support microforme, papier, électronique et/ou autres formats.

L'auteur conserve la propriété du droit d'auteur et des droits moraux qui protègent cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

In compliance with the Canadian Privacy Act some supporting forms may have been removed from this thesis.

Conformément à la loi canadienne sur la protection de la vie privée, quelques formulaires secondaires ont été enlevés de cette thèse.

While these forms may be included in the document page count, their removal does not represent any loss of content from the thesis.

Bien que ces formulaires aient inclus dans la pagination, il n'y aura aucun contenu manquant.


Canada

Abstract

The field of evolutionary computing includes a variety of simulation algorithms and problem-solving techniques, such as genetic algorithms and genetic programming, that use abstract versions of principles and mechanisms from biological evolution. There still remain many ideas, principles, and mechanisms from evolution, including molecular evolution and evolutionary developmental biology, that await adoption or further exploration in evolutionary computing. Many principles and mechanisms are already widely adopted and developed, such as mutation, recombination, selection, adaptation, and co-evolution, but plenty more that are known to contribute to the power of evolution in nature have barely begun to be explored, developed, and harnessed.

One such mechanism is that of exaptation. Exaptations occur when a trait is co-opted to serve a new role that it previously had not. Exaptation in nature is a valuable source of variation and innovation that augments the variation and innovation generated by mutation at the genetic level. When exaptations occur, complete ‘fully-evolved’ structures, behaviors, and entities are made available to serve or enhance a new function without having to be built up from scratch. Side-effects of old traits can be amplified by selection, enriching the repertoire of abilities for an evolutionary lineage of organisms.

As processing power continues to increase, evolutionary computing systems will be able to take steps that bring them closer to the levels of complexity and power found in biological evolution. Better study and development, and more widespread adoption of the mechanisms that drive natural evolution will need to take place to help realize the full potential of evolutionary computing in the coming years.

This work examines the current use of exaptation in evolutionary computing, proposes a framework of categorization for different types of exaptation and functional shift mechanisms, and develops a number of example exaptive systems to test and demonstrate the utility of exaptation in a computational context.

Acknowledgments

I would like to thank my supervisor, Franz Oppacher, for his pointers, guidance, suggestions, and support during the time in which this research and writing took place. The ample freedom he provided me, allowing me to tackle those issues and subjects that most interested me, is much appreciated.

Many thanks are also due to my thesis examination committee: Stan Matwin, Mark Lanthier, George Carmody, and Dan Ashlock. Their patience in reading and constructively criticizing this document, each from a different perspective and base of expertise and experience, was most helpful.

Many suggestions and encouragement for this research came from meetings of the Artificial Life and Evolutionary Computing (ALEC) group at Carleton University. The mixture of a relaxed informal atmosphere with engaging and thought-provoking presentations and discussions was always a pleasure. Many of the ideas and experiments in this work were shaped both directly and indirectly by discussion with others at ALEC, especially Rob Catral, Steffen Christensen, Yandu Oppacher, and Hassan Masum.

Finally, the administrative and technical staff in the School of Computer Science at Carleton University were always very helpful in terms of support and information. The efforts of Claire Ryan, Linda Pfeiffer, Sandy Herbert, Sharmila Namasivayampillai, Andrew Miles, and Gerardo Reynaga are much appreciated.

Table of Contents

CHAPTER 1 INTRODUCTION AND OUTLINE.....	15
1.1 EVOLUTIONARY COMPUTING	15
1.2 EXAPTATION AND FUNCTIONAL SHIFT	16
1.3 OBJECTIVES.....	17
1.3.1 Primary Goals.....	17
1.3.2 Contributions.....	18
1.3.3 Summary of Contributions.....	21
1.4 THESIS OUTLINE.....	22
CHAPTER 2 INTRODUCTION TO EVOLUTIONARY COMPUTING.....	25
2.1 GENETIC ALGORITHMS.....	29
2.1.1 Representation.....	29
2.1.2 Selection.....	30
2.1.3 Crossover.....	30
2.1.4 Mutation.....	32
2.2 GENETIC PROGRAMMING.....	32
2.2.1 Representation.....	32
2.2.2 Selection.....	34
2.2.3 Crossover.....	34
2.2.4 Mutation.....	34
2.2.5 Exaptive Approaches	35
CHAPTER 3 EXAPTATION IN BIOLOGICAL EVOLUTION	36
3.1 ORIGINS OF THE TERM.....	36
3.2 HISTORY OF THE IDEA	39
3.3 THE IMPORTANCE OF EXAPTATION IN BIOLOGY.....	43
3.4 DEFINITIONS.....	46
3.5 EXAMPLES OF EXAPTATION IN BIOLOGY	48
CHAPTER 4 EXAPTATION IN EVOLUTIONARY COMPUTING	50
4.1 EXAPTATION SYSTEMS WITH IMPLICIT FITNESS	50
4.1.1 Spandrels in an ALife 2D Cellular Automaton.....	50
4.1.2 Preadaptation of Neural Networks in a Changing Environment.....	51
4.2 EXAPTATION IN SYSTEMS WITH EXPLICIT FITNESS.....	51
4.2.1 Incremental Approaches in the Evolution of Neural Network Robot Control Systems.....	51
4.2.2 Symbiotic Composition for Crossing Fitness Saddles.....	54
4.2.3 Exaptation in Dynamic Landscapes.....	56
4.2.4 Gradual Functional Change for Evolving Differentiation in Gene Regulatory Networks.....	59
CHAPTER 5 THE CARRYOVER TO EVOLUTIONARY COMPUTING	61
5.1 USE OF THE TERM “PREADAPTATION” IN EC.....	63
5.2 CONFUSION OF CURRENT UTILITY WITH HISTORICAL ORIGIN.....	64
5.3 TYPES OF EXAPTATION AND FUNCTIONAL SHIFT IN EC	65
5.3.1 Deliberate vs. Emergent Exaptation.....	65
5.3.2 Strong vs. Weak Exaptations.....	66
5.3.3 Cumulative vs. Replacement Exaptations.....	67
5.3.4 Composition as a Form of Weak Exaptation	68
5.4 SHARED GENOTYPE-PHENOTYPE MAPPING IN EXAPTIVE SYSTEMS	69
CHAPTER 6 EXAPTATION EXPERIMENTS WITH A SIMPLE GENETIC ALGORITHM.....	71
6.1 GENERAL DESCRIPTION.....	71

6.1.1	<i>GA Parameters</i>	75
6.1.2	<i>Representation</i>	76
6.1.3	<i>Crossover Operator</i>	77
6.1.4	<i>Mutation Operators</i>	78
6.1.5	<i>The BG Niche Fitness Function</i>	79
6.1.5.1	<i>The Effect of Each Box Type on the Game Ball</i>	80
6.1.5.2	<i>Population Initialization</i>	81
6.2	THE SINGLE NICHE EXAPTIVE GA	83
6.2.1	<i>Interlocking, or “Irreducible”, Complexity</i>	83
6.2.2	<i>The Functional Change Mechanism</i>	84
6.2.3	<i>The Complexity Check</i>	85
6.2.4	<i>Results</i>	86
6.2.5	<i>Conclusions</i>	89
6.3	THE TWO NICHE EXAPTIVE GA	92
6.3.1	<i>Speciation Facilitated by Exaptation</i>	92
6.3.2	<i>The BG Niche Fitness Function</i>	95
6.3.3	<i>The LEA Niche Fitness Function</i>	95
6.3.4	<i>Migration</i>	97
6.3.5	<i>Results</i>	99
6.3.6	<i>Visualizing Exaptation and Secondary Adaptation in Fitness Space</i>	104
6.3.7	<i>Conclusions</i>	107
6.4	THE FOUR NICHE EXAPTIVE GA	109
6.4.1	<i>The BG Niche Fitness Function</i>	110
6.4.2	<i>The PLCR Niche Fitness Function</i>	110
6.4.3	<i>The Sieve Niche Fitness Function</i>	112
6.4.4	<i>The CRD Niche Fitness Function</i>	113
6.4.5	<i>Migration</i>	114
6.4.6	<i>Results</i>	116
6.4.6.1	<i>Appearance of Viable Individuals</i>	117
6.4.6.2	<i>Potential vs. Actual Migration Routes</i>	118
6.4.6.3	<i>Exaptation Events: Sequence and Rates</i>	118
6.4.7	<i>Conclusions</i>	121
6.5	THE COMBINATION-FITNESS EXAPTIVE GA	122
6.5.1	<i>The Fitness Function before Exaptation</i>	122
6.5.2	<i>The Fitness Function after Exaptation</i>	123
6.5.3	<i>GA Parameters</i>	124
6.5.4	<i>Fitness Trajectories</i>	124
6.5.4.1	<i>LEA Fitness Trajectories</i>	125
6.5.4.2	<i>LEA Fitness Improvement vs. Generation</i>	128
6.5.4.3	<i>Success Rates</i>	130
6.5.4.4	<i>Trajectories in 2D Fitness Space</i>	132
6.5.5	<i>The Role of Plateau- and Valley-Crossing</i>	133
6.5.5.1	<i>Results</i>	135
6.5.6	<i>Conclusions</i>	139
CHAPTER 7 EXAPTATION EXPERIMENTS WITH A 2D GP NAVIGATOR		141
7.1	VERSION 1	141
7.1.1	<i>Description</i>	141
7.1.1.1	<i>The Environment</i>	141
7.1.1.2	<i>The Learning Tasks</i>	143
7.1.1.3	<i>The Navigator</i>	149
7.1.1.4	<i>Early Termination of Evaluations</i>	152
7.1.2	<i>Results</i>	152
7.2	VERSION TWO	156
7.2.1	<i>Description</i>	156
7.2.1.1	<i>The Environment</i>	157
7.2.1.2	<i>The Learning Tasks</i>	158
7.2.1.3	<i>The Navigator</i>	166

7.2.1.4 Early Termination of Evaluations.....	166
7.2.2 Results.....	166
7.3 CONCLUSIONS	169
CHAPTER 8 EXAPTATION EXPERIMENTS WITH 3D VIRTUAL CREATURES.....	172
8.1 GENERAL DESCRIPTION.....	172
8.2 AN OVERVIEW OF RELATED PROJECTS IN EVOLUTIONARY ROBOTICS	173
8.3 THE 3DVCE GA	182
8.3.1 Evolutions, Runs, Fitness Measurements, and Evaluations.....	183
8.3.2 Fitness Schedules.....	184
8.3.3 Producing Offspring.....	185
8.4 THE PHYSICS ENGINE	186
8.5 CREATURE PHENOTYPES AND GENOTYPES.....	186
8.5.1 Morphology and Control	186
8.5.1.1 Segments.....	186
8.5.1.2 Joints.....	187
8.5.1.3 Sensors.....	188
8.5.1.4 Control.....	189
8.5.1.5 Oscillators.....	191
8.5.2 Creature Genotypes.....	191
8.5.3 Embryogenesis.....	193
8.6 FITNESS EVALUATION	195
8.6.1 Environments.....	198
8.6.2 Combining Fitness Components.....	199
8.7 VARIATION OPERATORS	199
8.7.1 Crossover Operators.....	200
8.7.1.1 Segment Specification Crossover.....	200
8.7.1.2 GP Tree Crossover in 3DVCE.....	200
8.7.2 Mutation Operators.....	202
8.7.2.1 Segment Specification Mutation.....	202
8.8 POPULATION INITIALIZATION	203
8.8.1 Creature Bushiness.....	203
8.8.2 Randomly-Generated GP Trees.....	203
8.9 EXAPTATION EXPERIMENT: TRAVEL ACROSS ROUGH TERRAIN	204
8.10 EXAPTATION EXPERIMENT: CATCHING AND TRANSPORTING A SPHERE	208
CHAPTER 9 EXAPTATION EXPERIMENTS WITH SYMBOLIC REGRESSION USING GP...211	
9.1 PIECEWISE BUILD-UP TO A TARGET FUNCTION.....	211
9.1.1 Target Functions.....	211
9.1.2 The GP System.....	212
9.1.3 Results.....	213
9.1.4 Conclusions.....	215
9.2 USING THE TARGET'S DERIVATIVE FOR PREADAPTATION.....	216
9.2.1 Target Functions.....	217
9.2.2 The GP System.....	219
9.2.3 Results.....	219
9.2.4 Conclusions.....	222
CHAPTER 10 EXAPTATION EXPERIMENTS WITH A GP MAZE RUNNER.....224	
10.1 DESCRIPTION.....	224
10.1.1 The Mazes.....	224
10.1.2 The Agent.....	225
10.1.3 Fitness Functions.....	225
10.1.3.1 The Maze Solving Fitness Function.....	226
10.1.3.2 The Preadapting Fitness Function.....	227
10.1.4 The GP System.....	227
10.2 RESULTS.....	229

10.3 CONCLUSIONS	231
CHAPTER 11 CONCLUSION	232
11.1 SUMMARY OF CONTRIBUTIONS AND EXPERIMENTS.....	232
11.2 DIFFICULTIES.....	238
11.3 FUTURE WORK.....	240
APPENDIX A. EXAMPLES OF EXAPTATION HYPOTHESES IN BIOLOGY	243
APPENDIX B. DETAILS OF THE 3DVCE SYSTEM.....	248
B.1. THE MAIN GA LOOP.....	248
<i>B.1.1. The Halting Condition.....</i>	<i>249</i>
<i>B.1.2. Elitism.....</i>	<i>250</i>
B.2. FIXED SYSTEM PARAMETERS.....	251
<i>B.2.1. Physics Engine Settings.....</i>	<i>251</i>
<i>B.2.2. Creature Control System Settings.....</i>	<i>252</i>
<i>B.2.3. Creature Morphology Settings</i>	<i>253</i>
B.3. CREATURES	257
<i>B.3.1. Phenotypes</i>	<i>258</i>
B.3.1.1. Body Structure.....	258
B.3.1.2. Embryogenesis.....	260
B.3.1.3. Creature Size.....	264
<i>B.3.2. Genotypes</i>	<i>264</i>
B.3.2.1. Sine Wave Generators	266
B.3.2.2. GP Trees.....	267
B.3.2.3. Body Segment Specifications	271
B.3.2.3.1. Branch Specifications.....	273
B.4. FITNESS FUNCTIONS	277
<i>B.4.1. Environments.....</i>	<i>277</i>
B.4.1.1. Flat Terrain Environment.....	278
B.4.1.2. Rough Terrain Environment	278
B.4.1.3. Spheres Environment.....	280
B.4.1.4. Water Environment.....	281
<i>B.4.2. Components of Fitness</i>	<i>282</i>
B.4.2.1. DIST Fitness Component.....	282
B.4.2.2. MAXH Fitness Component	283
B.4.2.3. AVGH Fitness Component.....	284
B.4.2.4. TOG Fitness Component	284
B.4.2.5. Sphere Fitness Component	284
B.4.2.6. Transport Fitness Component.....	285
<i>B.4.3. The Fitness Formula</i>	<i>286</i>
B.5. EVOLUTION CONFIGURATION	286
<i>B.5.1. GA Settings.....</i>	<i>287</i>
<i>B.5.2. Creature Morphology Settings</i>	<i>289</i>
<i>B.5.3. Creature Control System Settings.....</i>	<i>291</i>
<i>B.5.4. Fitness Function Schedules</i>	<i>291</i>
B.6. VARIATION OPERATORS	293
<i>B.6.1. Crossover</i>	<i>293</i>
B.6.1.1. Creature Crossover	293
B.6.1.2. GP Tree Crossover.....	294
B.6.1.2.1. Random Tree Crossover	296
B.6.1.2.2. Regular Tree Crossover.....	296
B.6.1.2.3. Special Tree Crossover.....	297
<i>B.6.2. Mutation</i>	<i>298</i>
B.6.2.1. Creature Mutation.....	298
B.6.2.2. Segment Copying Mutation	301
B.6.2.3. Segment Splitting Mutation.....	302
B.6.2.4. Segment Randomizing Mutation	304
B.6.2.5. 'Stub' Growing Mutation.....	305

B.6.2.6. GP Tree Mutation	306
B.6.2.6.1. Shrink-Tree Method	308
B.6.2.6.2. Grow-Tree Method	308
B.6.2.6.3. Point-Mutate-Tree Method	308
B.7. POPULATION INITIALIZATION	309
<i>B.7.1. Random Creature Generation</i>	<i>310</i>
<i>B.7.2. Random GP Tree Generation</i>	<i>310</i>
B.8. EXPERIMENTAL SETUPS	311
<i>B.8.1. Setup One: Traversal of Rough Terrain without Exaptation</i>	<i>312</i>
<i>B.8.2. Setup Two: Traversal of Rough Terrain with Exaptation</i>	<i>313</i>
<i>B.8.3. Setup Three: Sphere Transporting without Exaptation</i>	<i>315</i>
<i>B.8.4. Setup Four: Sphere Transporting with Exaptation</i>	<i>316</i>
APPENDIX C. MAZES	318
APPENDIX D. EXAPTATION CATEGORY RATIONALES	321

List of Tables

TABLE 3.1 DEFINITIONS FOR TYPES OF ADAPTATIONS, THEIR ROLES, AND PROCESSES THAT PRODUCE THEM.....	47
TABLE 6.1 PARAMETERS AND VALUES FOR THE EXAPTIVE GA.	76
TABLE 6.2 MEAN BEST FITNESS AND OTHER STATISTICS FOR THE COMBINATION-FITNESS EXAPTIVE GA.	127
TABLE 7.1 PARAMETERS AND SETTINGS FOR THE SIX EPOCHS OF THE 2D NAVIGATOR GP SYSTEM.	143
TABLE 7.2 SETTINGS FOR THE 2D NAVIGATOR GP SYSTEM.....	149
TABLE 7.3 BEST FITNESS STATISTICS FOR THE 2D NAVIGATOR GP SYSTEM.	154
TABLE 7.4 PARAMETERS AND SETTINGS FOR THE THREE EPOCHS OF THE 2D GP NAVIGATOR (VERSION TWO).	159
TABLE 7.5 SETTINGS FOR THE 2D NAVIGATOR GP SYSTEM, VERSION TWO.	165
TABLE 7.6 BEST FITNESS STATISTICS FOR THE 2D GP NAVIGATOR, VERSION TWO.	167
TABLE 9.1 GP SYSTEM SETTINGS FOR THE 'PIECEWISE' SYMBOLIC REGRESSION EXPERIMENTS.	212
TABLE 9.2 SETTINGS FOR THE 'DERIVATIVE' EXAPTIVE GP SYSTEM.....	219
TABLE 9.3 BEST FITNESS AND HITS STATISTICS FOR THE 'DERIVATIVE' EXAPTIVE GP SYSTEM.	220
TABLE 9.4 MEAN BEST FITNESS VALUES AND HIT COUNTS FOR SEVERAL HIT THRESHOLDS FOR TARGET FUNCTION #2.	221
TABLE 10.1 PARAMETERS OF THE MAZE SOLVER GP SYSTEM.	228
TABLE 10.2 BEST FITNESS AND MAZES SOLVED STATISTICS FOR THE MAZE RUNNER GP SYSTEM.	230
TABLE 11.1 TEN EC SYSTEMS INVOLVING EXAPTATION OR FUNCTIONAL CHANGE.	233
TABLE 11.2 TYPES OF EXAPTATION EXHIBITED IN BIOLOGICAL EVOLUTION, IN THE SYSTEMS DESCRIBED IN CHAPTER 4, AND IN THE SYSTEMS USED FOR EXPERIMENTS IN THE PRESENT DOCUMENT.	235

List of Figures

FIGURE 2.1 DEPICTIONS OF SINGLE-POINT, TWO-POINT, AND UNIFORM GA CROSSOVER.....	31
FIGURE 2.2 AN EXAMPLE OF A GP TREE REPRESENTING THE FORMULA $x(1+((x-y)/y))$	33
FIGURE 3.1 A BREAKDOWN OF TRAITS BASED ON CURRENT AND PAST UTILITY AND ORIGINS.	38
FIGURE 5.1 HYPOTHETICAL 2D FITNESS GRAPHS FOR CUMULATIVE AND REPLACEMENT EXAPTATION.	67
FIGURE 6.1 EXAMPLES OF EVOLVED PHENOTYPES IN THE BG NICHE UNDER BOTH VERSIONS OF ITS FITNESS FUNCTION.	78
FIGURE 6.2 AN ILLUSTRATION OF THE RULES GOVERNING EACH BOX TYPE IN THE BG NICHE.	81
FIGURE 6.3 MEAN AND FOUR QUANTILES OF BEST FITNESS VS. GENERATION IN THE SINGLE NICHE EXAPTIVE GA.	86
FIGURE 6.4 MEAN AND FOUR QUANTILES OF CHROMOSOME PENALTY P VS. GENERATION IN THE SINGLE NICHE EXAPTIVE GA.....	87
FIGURE 6.5 MEAN AND FOUR QUANTILES OF THE NUMBER OF INDIVIDUALS POSSESSING INTERLOCKING COMPLEXITY VS. GENERATION IN THE SINGLE NICHE EXAPTIVE GA.	88
FIGURE 6.6 MEAN AND FOUR QUANTILES OF THE NUMBER OF VIABLE INDIVIDUALS VS. GENERATION IN THE SINGLE NICHE EXAPTIVE GA.....	89
FIGURE 6.7 MEAN AND STANDARD DEVIATION OF INDIVIDUALS EXHIBITING INTERLOCKING COMPLEXITY AS A PERCENTAGE OF TOTAL POPULATION VS. GENERATION IN THE SINGLE NICHE EXAPTIVE GA.	89
FIGURE 6.8 AN ALTERNATIVE DIVISION INTO PARTS OF THE PHENOTYPE FROM THE RIGHT SIDE OF FIGURE 6.1.	91
FIGURE 6.9 AN EXAMPLE PHENOTYPE FROM THE LEA NICHE, JUST A FEW GENERATIONS AFTER EXAPTATION, SHOWING A FIT BG NICHE PHENOTYPE IN THE PROCESSING OF BEING CONVERTED TO A FIT LEA NICHE PHENOTYPE.....	97
FIGURE 6.10 NICHES AND MIGRATION ROUTE IN THE TWO NICHE EXAPTIVE GA.....	98
FIGURE 6.11 A CLADOGRAM DEPICTING EVOLUTION, EXAPTATION, AND SECONDARY ADAPTATION IN THE TWO NICHE EXAPTIVE GA.....	99
FIGURE 6.12 MEAN AND FIVE PERCENTILES OF BEST FITNESS VS. GENERATION IN THE TWO NICHE EXAPTIVE GA'S BG NICHE.	100
FIGURE 6.13 MEAN AND FIVE PERCENTILES OF CHROMOSOME LENGTH PENALTY P VS. GENERATION IN THE TWO NICHE EXAPTIVE GA'S BG NICHE.	102
FIGURE 6.14 MEAN AND FIVE PERCENTILES OF THE NUMBER OF VIABLE INDIVIDUALS VS. GENERATION IN THE TWO NICHE EXAPTIVE GA'S LEA NICHE.	102
FIGURE 6.15 MEAN AND FIVE PERCENTILES OF BEST FITNESS VS. GENERATION IN THE TWO NICHE EXAPTIVE GA'S LEA NICHE.	103
FIGURE 6.16 MEAN NUMBER OF MIGRANTS PER LOGGING INTERVAL VS. GENERATION, AND MIGRANTS PER RUN FREQUENCIES IN THE TWO NICHE EXAPTIVE GA.....	103
FIGURE 6.17 HYBRIDIZATION TEST OUTCOMES IN THE TWO NICHE EXAPTIVE GA.....	104
FIGURE 6.18 MEAN AND MEDIAN POPULATION TRAJECTORIES THROUGH FITNESS SPACE IN A VARIANT OF THE TWO NICHE EXAPTIVE GA.....	105
FIGURE 6.19 EXAMPLES OF EVOLVED PHENOTYPES FROM EACH OF THE FOUR NICHES (LEFT TO RIGHT: BG, PLCR, SIEVE, AND CRD) IN THE FOUR NICHE EXAPTIVE GA.	109
FIGURE 6.20 POTENTIAL AND ACTUAL MIGRATION PATHS BETWEEN THE FOUR NICHES IN THE FOUR NICHE EXAPTIVE GA. ACTUAL PATHS IN BOLD.	114
FIGURE 6.21 A CLADOGRAM DEPICTING EVOLUTION, EXAPTATION, AND SECONDARY ADAPTATION IN THE FOUR NICHE EXAPTIVE GA.	116
FIGURE 6.22 NUMBER OF VIABLE INDIVIDUALS VS. GENERATION IN ALL NICHES OF THE FOUR NICHE EXAPTIVE GA.	117
FIGURE 6.23 MEAN NUMBER OF MIGRANTS BETWEEN NICHES VS. GENERATION IN THE FOUR NICHE EXAPTIVE GA.	118
FIGURE 6.24 FREQUENCY DISTRIBUTION FOR THE NUMBER OF MIGRANTS PER RUN BETWEEN THE BG AND PLCR NICHES IN THE FOUR NICHE EXAPTIVE GA.	119
FIGURE 6.25 FREQUENCY DISTRIBUTION FOR THE NUMBER OF MIGRANTS PER RUN BETWEEN THE PLCR AND SIEVE NICHES IN THE FOUR NICHE EXAPTIVE GA.....	120

FIGURE 6.26 FREQUENCY DISTRIBUTION FOR THE NUMBER OF MIGRANTS PER RUN BETWEEN THE SIEVE AND CRD NICHES IN THE FOUR NICHE EXAPTIVE GA.	120
FIGURE 6.27 MEAN BEST LEA FITNESS TRAJECTORIES FOR THE COMBINATION-FITNESS EXAPTIVE GA.	125
FIGURE 6.28 MEAN BEST LEA FITNESS TRAJECTORIES IN THE FINAL GENERATIONS OF THE COMBINATION-FITNESS EXAPTIVE GA.	127
FIGURE 6.29 MEAN FINAL BEST FITNESS IN THE COMBINATION-FITNESS EXAPTIVE GA AS A FUNCTION OF PREADAPTATION TIME.	128
FIGURE 6.30 ABSOLUTE FITNESS IMPROVEMENTS OVER THE NON-EXAPTIVE CONFIGURATION IN THE COMBINATION-FITNESS EXAPTIVE GA.	129
FIGURE 6.31 PEAK LEA FITNESS IMPROVEMENT OVER ALL 5000 GENERATIONS FOR THE COMBINATION-FITNESS EXAPTIVE GA.	130
FIGURE 6.32 SUCCESS RATES FOR THE COMBINATION-FITNESS EXAPTIVE GA.	131
FIGURE 6.33 SUCCESS RATE AS A FUNCTION OF PREADAPTATION TIME IN THE COMBINATION-FITNESS EXAPTIVE GA.	132
FIGURE 6.34 COMBINATION-FITNESS EXAPTIVE GA TRAJECTORIES THROUGH 2D FITNESS SPACE.	133
FIGURE 6.35 PROBABILITY OF THE ELITE INDIVIDUAL HAVING HIGHER LEA FITNESS THAN ITS PARENT'S IN THE COMBINATION-FITNESS EXAPTIVE GA.	135
FIGURE 6.36 PROBABILITY OF THE ELITE INDIVIDUAL HAVING LEA FITNESS EQUAL TO ITS PARENT'S IN THE COMBINATION-FITNESS EXAPTIVE GA.	136
FIGURE 6.37 PROBABILITY OF THE ELITE INDIVIDUAL HAVING LEA FITNESS LOWER THAN ITS PARENT'S IN THE COMBINATION-FITNESS EXAPTIVE GA.	136
FIGURE 6.38 OCCURRENCE RATES FOR PARENT-CHILD RELATIONSHIPS IN E1200 BEFORE EXAPTATION.	138
FIGURE 7.1 GP NAVIGATOR ENVIRONMENT AND OBJECTS (VERSION 1).	142
FIGURE 7.2 MEAN BEST FITNESS AND LOG-FITNESS VS. GENERATION IN THE 2D NAVIGATOR GP SYSTEM.	153
FIGURE 7.3 MEAN BEST LOG-FITNESS VALUES FOR THE 2D NAVIGATOR GP SYSTEM, WITH STANDARD DEVIATIONS (LEFT SIDE) AND 95% CONFIDENCE INTERVALS (RIGHT SIDE).	154
FIGURE 7.4 BEST FITNESS OUTCOME DISTRIBUTIONS FOR THE 2D NAVIGATOR GP SYSTEM.	155
FIGURE 7.5 GP NAVIGATOR ENVIRONMENT AND OBJECTS (VERSION 2).	158
FIGURE 7.6 MEAN BEST CONVERTED FITNESS TRAJECTORIES FOR THE 2D GP NAVIGATOR, VERSION TWO.	167
FIGURE 7.7 MEAN BEST CONVERTED FITNESS VALUES FOR THE 2D GP NAVIGATOR, VERSION TWO, WITH STANDARD DEVIATIONS (LEFT SIDE) AND 95% CONFIDENCE INTERVALS (RIGHT SIDE).	168
FIGURE 7.8 BEST CONVERTED FITNESS OUTCOME DISTRIBUTIONS FOR THE 2D GP NAVIGATOR, VERSION TWO.	168
FIGURE 8.1 TWELVE RANDOMLY GENERATED 3DVCE CREATURES.	187
FIGURE 8.2 AN ILLUSTRATION OF THE 3DVCE CREATURE GENOTYPE.	192
FIGURE 8.3 TWO EXAMPLE SCREENSHOTS OF EVOLVED 3DVCE CREATURES.	194
FIGURE 8.4 BEST FITNESS VS. GENERATION, AVERAGED OVER ALL RUNS, FOR 3DVCE CONFIGURATIONS 1 AND 2.	205
FIGURE 8.5 BEST-OF-RUN FITNESS VALUES AND MEANS THEREOF FROM INDEPENDENT RUNS OF 3DVCE CONFIGURATIONS #1 AND #2.	206
FIGURE 8.6 BEST FITNESS VS. GENERATION, AVERAGED OVER ALL RUNS, FOR 3DVCE CONFIGURATIONS 3 AND 4.	209
FIGURE 8.7 BEST-OF-RUN FITNESS VALUES FROM INDEPENDENT RUNS OF 3DVCE CONFIGURATIONS 3 AND 4.	210
FIGURE 9.1 BEST FITNESS OUTCOME DISTRIBUTIONS FOR THE 'PIECEWISE' EXAPTIVE GP SYSTEM.	214
FIGURE 9.2 OUTCOME DISTRIBUTIONS FOR THE NUMBER OF HITS FROM THE BEST INDIVIDUALS IN THE 'PIECEWISE' EXAPTIVE GP SYSTEM.	214
FIGURE 9.3 MEAN BEST FITNESS AND HITS, WITH 95% CONFIDENCE INTERVALS, FOR THE 'PIECEWISE' EXAPTIVE GP SYSTEM.	215
FIGURE 9.4 TARGET FUNCTION #1. $F(x) = 28x^2(x - 0.8)(x - 0.9) - 0.4$	217
FIGURE 9.5 TARGET FUNCTION #2. $F(x) = (x - 0.05)(x - 0.7)(x - 0.8)(x - 0.9)$	217
FIGURE 9.6 TARGET FUNCTION #3. $F(x) = 600(x - 0.05)(x - 0.1)(x - 0.3)(x - 0.5)(x - 0.99)$	218
FIGURE 9.7 TARGET FUNCTION #4. $F(x) = 10(x - 0.4)(x - 0.7) - 1$	218

FIGURE 9.8 TARGET FUNCTION #5. $F(x) =$ $100000x(x + 0.3)(x - 0.05)(x - 0.1)(x - 0.2)(x - 0.5)(x - 0.8)(x - 0.9)(x - 0.95)(x - 1.3)$	218
FIGURE 9.9 MEAN BEST FITNESS VALUES AND HITS, WITH 95% CONFIDENCE INTERVALS, FOR THE 'DERIVATIVE' EXAPTIVE GP SYSTEM.	220
FIGURE 9.10 MEAN BEST FITNESS VALUES, WITH 95% CONFIDENCE INTERVALS, FOR SEVERAL HIT THRESHOLD VALUES FOR TARGET FUNCTION #2.	221
FIGURE 9.11 MEAN BEST HITS, WITH 95% CONFIDENCE INTERVALS, FOR SEVERAL HIT THRESHOLD VALUES FOR TARGET FUNCTION #2.	222
FIGURE 10.1 MEAN BEST FITNESS VS. GENERATION FOR THE MAZE RUNNER GP SYSTEM.	229
FIGURE 10.2 BEST FITNESS AND STANDARD DEVIATION VS. AMOUNT OF PREADAPTATION IN THE MAZE RUNNER GP SYSTEM.	230
FIGURE 10.3 NUMBER OF MAZES SOLVED AND STANDARD DEVIATION VS. AMOUNT OF PREADAPTATION IN THE MAZE RUNNER GP SYSTEM.	230

List of Appendices

APPENDIX A. EXAMPLES OF EXAPTATION HYPOTHESES IN BIOLOGY	243
APPENDIX B. DETAILS OF THE 3DVCE SYSTEM.....	248
APPENDIX C. MAZES	318
APPENDIX D. EXAPTATION CATEGORY RATIONALES.....	321

Chapter 1 Introduction and Outline

This thesis studies the roles, types, and uses of exaptation in evolutionary computing.

This is done through an examination of exaptation in the context of evolutionary biology and, more importantly, in the context of its current extent of use in the evolutionary computing literature. A basic framework for the categorization of exaptations, and a method for visualizing and distinguishing two types of exaptations are both proposed and applied. Most importantly, a number of different exaptive evolutionary systems are implemented to demonstrate uses and benefits of exaptation.

This chapter briefly describes evolutionary computing and exaptation, lays out the objectives and contributions of the research, and gives a short outline of the remainder of the dissertation.

1.1 Evolutionary Computing

The principle of natural selection, first understood and elucidated by Charles Darwin [Darwin, 1859] and Alfred Wallace in the middle of the 19th century, explains how a population of imperfectly-replicating entities will change over generations to become better adapted to their environment. The understanding of genetics, inheritance, and DNA, developed through the 20th century, shows the nature of a digital coding system with which the process of evolution tinkers. The theoretical framework combining the insights of Darwin with the understanding of genetics, called the modern synthesis, is the thread that ties together all aspects of the biological sciences. The pioneering work of [Fogel, Owens, & Walsh, 1966], [Holland, 1993], and others has shown how the

principles at work in the modern synthesis, when abstracted, constitute a very general algorithmic bottom-up problem solving technique that can be applied in many media to many problems under the proper conditions. The study of software and hardware systems that employ these abstracted principles is called *evolutionary computing*.

1.2 Exaptation and Functional Shift

Through the introduction of terms like “spandrel” [Gould & Lewontin, 1979] and “exaptation” [Gould & Vrba, 1982] to the field of evolutionary biology, Gould and others brought an added focus to some important mechanisms of innovation and variation in the natural world. These ideas, which dealt with organisms’ traits, behaviors, structures, and their side-effects being co-opted for new and potentially novel functions, work in harmony with the usual mechanisms of slow, gradual, and cumulative adaptation through mutation, recombination, and natural selection. Hypotheses about historical evolutionary changes involving exaptations, or structures co-opted for new functions, abound in biology and are well-established, well-tested, and accepted aspects of the evolutionary history of life on this planet.

Just as the principles from natural selection, genetics, mutation, recombination, and other aspects of evolution have been fruitfully adopted into the field of evolutionary computing, so too should the mechanism of exaptation. While some systems in evolutionary computing have made forays into this area, the potential of exaptation is still in need of a more widespread incorporation and understanding within the field.

0 explores the history and role of exaptation in evolutionary biology. Chapter 4 examines the extent to which this important mechanism has been adopted within evolutionary computing by looking at several examples of exaptive systems in the literature. A discussion of a number of aspects of exaptation as it relates to evolutionary computing appears in Chapter 5. Chapter 6 through Chapter 10 describe ten different experiments using five new implementations of exaptive GA and GP systems, each exhibiting different behaviors or exploring different aspects of the use of exaptation. Chapter 11 provides a summary and conclusion.

1.3 Objectives

1.3.1 Primary Goals

The goals of this research are centred on the exploration of exaptation mechanisms and their roles in EC. They include the task of describing the concept of exaptation in its native context of evolutionary biology, examining the uses of exaptation in EC systems described in the literature, and comparing and contrasting the two. Of key importance is the objective of developing several EC systems for testing the hypothesis that some exaptive EC systems can see benefits in terms of superior performance when compared with otherwise identical systems lacking a mechanism of exaptation. A number of exaptive EC systems are implemented for solving a variety of learning tasks, employing several types of exaptation, and demonstrating multiple ways in which exaptation can influence the evolutionary process. Of secondary importance are the objectives of fleshing out a basic framework for the categorization of different types of exaptations and applying the framework to exaptive EC systems from the literature and to exaptive

systems from this research, and the demonstration of a technique for visually representing the evolutionary dynamics of an evolving population in an exaptive EC system.

1.3.2 Contributions

The contributions of this research are as follows. First is the development of five GA, GP, and hybrid EC systems (listed below) that employ mechanisms of exaptation and functional shift, and the demonstration of various aspects of the utility of such mechanisms. The most complex of these is a hybrid GA/GP system with evolving morphology and control systems for embodied autonomous agents in a simulated physical environment. The five exaptive EC systems and their corresponding experiments and contributions are as follows.

The Simple Exaptive GA

Single Niche Exaptive GA

This version of the simple exaptive GA system demonstrates the use of a gradual change in the GA's fitness function in the evolution of systems exhibiting a property known as "interlocking complexity".

Two Niche Exaptive GA

This version of the simple exaptive GA demonstrates the use of exaptation in producing structures that meet, for a particular fitness function, minimal requirements that are not met by random initialization alone. The mechanism also facilitates a process of speciation by allowing a source population to split into two independently evolving groups. The 2D fitness graphing technique for visually representing the evolutionary dynamics of

an exaptive EC system is demonstrated using a modified version of the two niche exaptive GA.

Four Niche Exaptive GA

This version of the simple exaptive GA demonstrates the use multiple exaptations allowing a population to spread from a source niche to three others sequentially, each with a different fitness function whose minimal requirements are not satisfied by random initialization alone.

Combination-Fitness Exaptive GA

This version of the simple exaptive GA demonstrates the use of exaptation in greatly boosting the performance of an otherwise identical non-exaptive GA. The experiments also explore the role of fitness plateau crossing and fitness saddle crossing in contributing to the performance increase. The 2D fitness graphing technique for visually representing the evolutionary dynamics of an exaptive EC system is demonstrated using data collected from the combination-fitness exaptive GA.

3D Virtual Creature Evolution

Jumping and Rough Terrain Traversal

This experiment tests the hypothesis that an exaptive approach in which the morphology and control systems of virtual robots are first evolved for a jumping task and then for a rough terrain traversal task can outperform an otherwise identical system evolved only for the terrain traversal.

Catching and Transporting Spheres

This experiment tests the hypothesis that an exaptive approach in which the morphology and control system of virtual robots are first evolved for a sphere-catching task and then for a sphere-transporting task can outperform an otherwise identical system evolved only for transporting spheres.

Symbolic Regression in GP

‘Piecewise’ Symbolic Regression in GP

Experiments with this system test the hypothesis that preadapting a GP population for a symbolic regression problem by using simplified versions of the target function can boost performance as compared to an otherwise identical GP system with no such preadaptation.

‘Derivative’ Symbolic Regression in GP

Experiments with this system test the hypothesis that an approximated version of an unknown target function’s derivative can be used to preadapt a GP population for symbolic regression in such a way that performance is enhanced compared to an otherwise identical non-exaptive system.

The 2D Navigator GP System

This system demonstrates an improvement in the performance of a GP system set to evolve navigation controllers for agents performing a complex task in a 2D environment. The improvement is gained through the use of multiple epochs punctuated by exaptation, in which the learning task starts simple and is made

more complex with each new epoch. Performance is compared to an otherwise identical system that adapts only to the final learning task from the start.

The Maze Runner GP System

This system demonstrates an improvement in the performance of a GP system set to evolve a maze running algorithm for guiding an agent through a set of seven grid mazes. The improvement is made by first evolving a population using a fitness function that rewards only exploration, and then exapting and further adapting the evolved population for the task of solving the mazes. Performance is compared to an otherwise identical GP system that uses the maze solving fitness function from the start.

1.3.3 Summary of Contributions

The main contributions of this research can be summarized as follows:

- Development, experiments, and analyses of several GA systems employing exaptation mechanisms are conducted.
- Development, experiments, and analyses of several GP systems employing exaptation mechanisms are conducted.
- Development, experiments, and analyses of a hybrid GA / GP system for exaptation in evolving morphology and control systems for embodied virtual robots are conducted.
- A basic framework for the categorization of exaptations in evolutionary computing that can contribute to a better understanding of functional shifts and their place in the field, as well as highlight the differences between such mechanisms in the artificial

realm versus their counterparts in biology is proposed. This framework is applied both to the systems developed and described here, as well as to several from the EC literature.

- As a tool to assist in the analysis of exaptive evolutionary computing systems, a method for visualizing the progress and dynamics of exaptive transitions of evolving populations in two-dimensional fitness space is introduced. This technique is demonstrated using two of the implemented exaptive GAs described in this work, and is used to visualize evolutionary dynamics for two different categories of exaptation.

1.4 Thesis Outline

Chapter 2 gives a more detailed description of evolutionary computing than was presented in section 1.1, above. Two of EC's most popular and well-known techniques, namely genetic algorithms and genetic programming, are described as well.

Chapter 3 looks at exaptation in its native context of evolutionary biology. The origins, history, and importance of the idea are discussed. Relevant terminology is defined, and several interesting examples are given, with many more available in Appendix A.

Chapter 4 looks at exaptation as it currently exists in EC. Several exaptive systems described in the literature are summarized; some are artificial life systems with implicit fitness and some are problem-solving EC systems with explicit fitness.

Chapter 5 looks at some of the differences between exaptation in biology and EC. It also proposes a basic categorization system highlighting distinctions between various types of

exaptation. Such a framework allows a more detailed comparison between different exaptive systems.

Chapter 6 presents the *Simple Exaptive GA* implementation using a variety of fitness functions and multiple niches. Four versions of the GA are constructed, each using exaptation in a different manner. Experiments and results are presented for all four.

Chapter 7 details the *2D Navigator GP System*, which uses multiple epochs of preadaptation to boost the performance of a GP population learning the complex task of controlling an embodied agent navigating through a 2D environment.

Chapter 8 lays out the details of the *3D Virtual Creature Evolution* system, a hybrid GA / GP system for evolving both the morphology and control systems of virtual robots in the environment of a 3D rigid-body physics simulator. Two experiments are described, each comparing an exaptive and a non-exaptive approach for particular learning tasks. A full description of the details of the system is more than can reasonably fit in the chapter, and is instead provided in Appendix B.

Chapter 9 presents two exaptive GP systems for symbolic regression problems. The first is the '*Piecewise Symbolic Regression GP System*' and the second is the '*Derivative Symbolic Regression GP System*'. Each is constructed to test a different hypothesis about how specific mechanisms of exaptation might be beneficial.

Chapter 10 introduces the *Maze Runner GP System*. The maze runner system evolves a population of algorithms for guiding an agent through a set of seven mazes embedded in

2D grids. A standard non-exaptive approach is compared to one involving exaptation and is found to exhibit superior performance.

Chapter 11 summarizes the research in this thesis by considering the contributions, experiments, and their results. This chapter includes some discussion of the difficulties in adopting exaptive approaches in EC, and briefly describes future directions for research with the exaptive EC systems described in Chapter 6 through Chapter 10.

Chapter 2 Introduction to Evolutionary Computing

Evolutionary computing is a field that includes a large number of algorithms and subfields such as genetic algorithms (GAs) [Holland, 1993], genetic programming (GP) [Koza, 1992], evolution strategies [Rechenberg, 1993], evolutionary programming [Fogel, Owens, & Walsh, 1966], learning classifier systems [Holland & Reitman, 1978], artificial immune systems [Dasgupta, 1999], artificial life, estimation of distribution algorithms [Larraña & Lozano, 2002], evolutionary multiobjective optimization [Deb et al., 2002], evolutionary robotics [Husbands & Meyer, 1998][Gomi & Griffith, 1996], gene expression programming [Ferreira, 2001], artificial gene regulatory networks [Banzhaf, 2003], and a host of others. They are all forms of metaheuristics, most of which make use of principles in operation in biological evolution. A metaheuristic is any very general heuristic approach that is used when an efficient problem-specific search algorithm is not known.

EC algorithms are population-based search techniques. They construct an initial population of candidate solutions to a given problem. These are usually generated at random, but can be supplied by the user or generated through some other process. As a starting point for the algorithms, the initial population usually consists of very poor solutions, especially when randomly generated.

Solutions are encoded using a representation scheme that stores information in a data structure. Each encoded solution (called a genotype) is decoded into a behavior, structure,

or entity (called a phenotype). The mapping from genotype to phenotype is often fairly direct, but can be very complex and indirect in some EC systems.

At some point in the algorithms, each phenotype is passed to an objective function. This function evaluates the phenotype and assigns it a value that reflects the goodness or quality of the solution. The objective function is usually called the fitness function, and the value it assigns to a candidate solution is usually referred to as that solution's fitness value, or simply its fitness.

As an EC algorithm progresses, the particular collection of solutions that make up its population changes. This is accomplished by the reproduction of some genotypes in the population and the removal or replacement of others. This is repeated in an iterative fashion, continually replacing the population with a new one. Each new population is called a generation. The term can be used to refer to the process of producing the new population as well as to refer to the newly-produced population itself. Some EC algorithms have no discrete generations but instead continually replace individuals in the population one at a time. Such algorithms are said to use a *steady state* approach. Often, steady-state EC algorithms will hold small reproduction tournaments in which, for example, three individuals are chosen at random with the worst of the three replaced by an offspring of one or both of the other two.

The choice of which genotypes to reproduce is always biased in favour of those that encode solutions with better fitness as determined by the fitness function. The reproduction methods are usually deliberately imperfect copying procedures. They typically involve a copy operation followed by a variation operator applied to the copy.

The genotypes chosen for reproduction are sometimes called parents, and the number of parents used to generate each new genotype (or offspring) can vary from algorithm to algorithm and from offspring to offspring within an algorithm. Some variation operators are called mutation operators if they make random changes to a single genotype. Such changes are usually very small perturbations in the genotype, the success of an EC algorithm being partly influenced by the degree to which similar genotypes map to solutions with similar fitness. Variation operators that create offspring by hybridizing two or more parents are often called crossover operators.

EC algorithms involve some form of selection scheme or selection operator. This is the part of an EC algorithm that creates a bias in favour of choosing fitter parents for reproduction. A wide variety of selection schemes exists in EC. Some choose parents with probability proportional to fitness. This is called fitness-proportional selection, and tends to be useful when a large fitness gradient is present. Others simply choose a handful of candidate parents at random, taking the fittest among them as the selected parent. This is called tournament selection and is often used when little is known about the fitness landscape. Another, called rank selection, uses selection probabilities that are a function of the candidate parent's rank position when the population is sorted by fitness value. Rank selection tends to be useful in cases involving small fitness gradients, such as when refining a very fit solution at the end of a run. Other schemes simply retain the best genotypes in a generation and replace the remaining ones with offspring from these "elite" genotypes, and still others cull the least fit genotypes after a round of reproduction has taken place. What all selection schemes have in common is that fitter individuals have a greater chance of survival and parenting than their less fit competitors.

The behavior of EC algorithms is often articulated using the metaphor of a fitness landscape [Wright, 1932]. One imagines each axis in a plane as representing the range of values for particular genes, and height above the plane as representing the fitness of the genotype corresponding to the point in the plane. This surface of fitness values is called the fitness landscape. The population of genotypes in an EC system forms a cluster of points on the fitness landscape. The variation operators create new points that are close to others in the population, and the entire population gradually climbs peaks in the landscape, moving more or less upward at each step. Sometimes populations become trapped at local optima in the landscape. The tendency to sample points that are far from the current population cluster is called exploration, and the tendency to sample points very close to the best individuals in the population, in order to zero in on the local peak, is called exploitation. Programmers and users of EC algorithms attempt to find a balance between exploration and exploitation.

EC algorithms have proven themselves to be useful tools for a variety of difficult problems including circuit design and layout, spacecraft trajectories, and antenna design, among others [Cage, Kroo, & Braun, 1995], [Stoica et al., 1999], [Barnum, Bernstein, & Spector, 2000], [Koza et al., 2000 & 2003], [Lohn et al., 2004]. The two evolutionary domains, biological evolution and EC, share more in common than mere basic principles and ingredients, as is evidenced by a wide range of high-level phenomena exhibited in both domains, such as the Baldwin effect [Ackley & Littman, 1992], [Baldwin, 1896], [Hinton & Nowlan, 1987], arms races [Ficici & Pollack, 1998], mimicry [Franks & Noble, 2002], parasitism [Hillis, 1990], [Robbins, 1994], co-evolution [Ficici & Pollack, 1998], [Hillis, 1990], [Rosin & Belew, 1995], exaptation [de Oliveira, 1994], [Gould &

Vrba, 1982], and speciation [Boxhorn, 1994], [Deb & Goldberg, 1989], [Fonseca & Fleming, 1995].

The following sections look briefly at two commonly used and well-studied classes of EC algorithms: GAs and GP. Implementations of several instances of algorithms of both of these types, and corresponding experimental results, are described and discussed in Chapter 6 through Chapter 10.

2.1 Genetic Algorithms

2.1.1 Representation

The traditional GA genotype is a fixed-length array of bits. This is often called the chromosome. The chromosome is decoded by converting groups of bits (usually consecutive) into higher-level data types like integers, real numbers, or characters that each represent particular parameters to the problem one is attempting to optimize.

Many GAs deviate from this traditional representation, and instead use vectors of higher-level data types directly, eliminating the need for a binary conversion. Others allow for chromosomes to vary in length as well. Deviations of these kinds require alternative variation operators as well. It's always the case that the variation operators must be devised in concordance with the representation scheme, of course, whether the chromosomes are bit-strings, integer vectors, fixed length, variable length, or use any other encoding.

2.1.2 Selection

Some of the most common selection operators used in GAs are the three described above: fitness-proportional selection, tournament selection, and rank selection. The selection operator is used to choose a genotype from the current generation to act as a parent when producing offspring for the next generation.

Some GAs reserve a special selection method for a small number of offspring in each generation (often just one). This form of selection involves simply choosing the best individuals from the previous generation and copying them without making any changes. Such GAs are said to be using elitism. Elitism is often used to ensure that the best fitness value in each generation increases monotonically.

2.1.3 Crossover

Just as there are several common selection operators for GAs, crossover operators come in several common varieties as well. The first is the single-point crossover operator. It iterates through the chromosome vector, copying alleles (specific values for a given gene) from the first of two selected parents until a randomly chosen index is reached. This index is called the crossover point. Beyond the crossover point, alleles are copied to the offspring from the second selected parent.

The second is the two-point crossover operator. It iterates through the chromosome vector just as the single-point crossover, copying alleles, but switches parents twice instead of once. The two crossover points are chosen at random.

The third is the uniform crossover operator. Using the uniform crossover operator, each gene in the offspring has an equal and independent chance of getting its allele from either parent, as if one were tossing a coin at each index in the vector to determine which parental allele to copy.

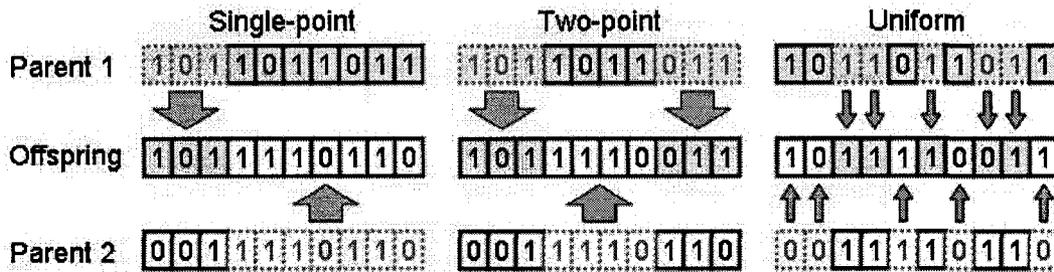


Figure 2.1 Depictions of single-point, two-point, and uniform GA crossover.

These three common crossover operators are depicted in Figure 2.1. There are many other crossover operators as well. Some involve more than two parents. Some combine parental alleles at each locus instead of choosing one or the other – taking a weighted average for example. Still others are tailored to some domain-specific aspect of the GA, such as an operator that crosses two Traveling Salesman Problem (TSP) [Applegate, 2007] tours where duplicate alleles (city indices in the TSP) may be forbidden in a chromosome.¹

The fraction of offspring produced by crossover is a parameter of the GA commonly called the crossover rate.

¹ At present, the best known algorithm for solving TSP is not an EC algorithm.

2.1.4 Mutation

In a traditional GA with bit-string chromosomes, the mutation operator iterates through the chromosomes, and, with a fixed probability, toggles the value of an allele. For chromosomes that aren't bit-strings, the operator will modify an allele in some other way, such as adding Gaussian noise to a floating-point value or incrementing and decrementing integers. This fixed probability of altering a gene is a parameter of the GA commonly called the mutation rate, and is usually kept quite small. Some GAs tune the mutation rate based on the number of genes to get a desired expected number of alterations, such as one. Others will adjust the rate throughout a run to increase or decrease population diversity.

Mutation is usually applied after a single parent is copied, and after two or more parents are combined via crossover.

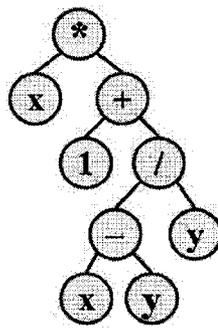
2.2 Genetic Programming

2.2.1 Representation

The primary difference between a GA and a GP system is the representation. While GAs use linear chromosomes, GP systems use ordered rooted trees, and in some cases small forests of such trees, as the data structure for the genome. A number of variants on this version of GP have been studied as well. Cartesian GP uses a directed acyclic graph representation. Some versions of GP use more GA-like linear representations. An example is a representation somewhat like machine-language instructions, called If-Skip-Action lists, or ISAc lists [Ashlock & Joenks, 1998]. Still others are based on evolved

grammars that, in turn, produce trees. This section deals only with the direct ordered rooted tree representation.

Trees in GP are composed of nodes. Each node is of a specific type, and there are usually a handful of types available, many of which are often problem-specific. In most GP systems, node types correspond to various functions that take a number of inputs. Those that take zero inputs are often called terminals, and those that take one or more inputs are often called functions. Each node in a GP tree will possess a child subtree for each of the inputs required by its type. In a symbolic regression problem, for example, if the function set were $\{+, -, *, /\}$, and the terminal set were $\{0, 1, x, y\}$ then an example GP tree might look like that shown in Figure 2.2.



$$x(1+((x-y)/y))$$

Figure 2.2 An example of a GP tree representing the formula $x(1+((x-y)/y))$.

In the symbolic regression example, above, the mapping from genotype to phenotype is extremely direct. This is not the case for all GP systems. In some schemes, such as cellular encoding [Gruau, 1994], the GP tree represents a program of instructions that, when followed, constructs the actual phenotype that will undergo fitness evaluation.

2.2.2 Selection

All of the various selection schemes and operators available in GA systems can be employed in GP systems as well, including fitness-proportional, rank, tournament, the use of elitism, and so on.

2.2.3 Crossover

Since a genotype in GP is not a linear structure, the crossover operators used in GAs are not applicable. The most common form of tree crossover used in GP converts two parent trees into a hybrid offspring tree. A subtree from within each parent tree is chosen with uniform random probability. The offspring tree is identical to one of the parents, except that its chosen subtree is replaced with the subtree chosen from the other parent.

2.2.4 Mutation

Since subtrees chosen uniformly at random in two parents can come from two different parts of the tree, even if both parents are identical, there is a high probability that the subtrees will not be homologues of one another. This makes the GP crossover operator potentially much more disruptive than its GA counterparts. For this reason, many GP systems use only this crossover operator, and dispense with any additional operators for mutation. Those that do use mutation have a variety of operators available, such as the replacement of a subtree with a randomly-generated subtree, crossover of the parent with itself, replacement of a node with another node of the same arity, mutation of node constants, and others.

2.2.5 Exaptive Approaches

The term “exaptive approach” will be used to designate any EC system that employs a mechanism of exaptation. An exaptation mechanism is one that allows either a whole phenotype, a partial phenotype, or a group of phenotypes to be co-opted to serve functions other than those that shaped them prior to the co-option event, if any. In some cases the functions may be the fitness function, in other cases the functions may be emergent ones that indirectly serve the fitness function. The chapters that follow will discuss exaptation, its importance in the field of evolutionary biology, and how that importance translates in the field of EC.

Chapter 3 Exaptation in Biological Evolution

3.1 Origins of the Term

The word exaptation is constructed from the Latin aptus, meaning fit, and ex, meaning from or out of. Like the word adaptation (the ad being Latin for towards), it can be used to refer either to a process or to its result. When used to refer to a biological trait, as opposed to a process, it was meant by Stephen Jay Gould and Elizabeth Vrba, who first introduced the term, to refer to traits that are “..fit (aptus) by reason of (ex) their form..” [Gould & Vrba, 1982]. When used to refer to a process or event, it denotes a change in the function of a trait at some point in its evolutionary history. This is sometimes referred to by other terms, such as co-option or, preadaptation, the latter having especially misleading connotations, explained below, though it was for a long time very widely used in the literature of evolutionary biology.

Referring to exaptation as a missing term in the “taxonomy of evolutionary morphology”, [Gould & Vrba, 1982] point out that the term adaptation has long been used to describe any feature or trait that enhances fitness, regardless of how it arose. They proposed the term exaptation to distinguish features that enhance fitness but were not sculpted by natural selection for their current role. For an example as old as the theory of evolution by natural selection itself, they quote [Darwin, 1859].

“The sutures in the skulls of young mammals have been advanced as a beautiful adaptation for aiding parturition, and no doubt they facilitate, or may be indispensable for this act; but as sutures occur in the skulls of young birds and reptiles, which have only to escape from a broken egg, we may infer that this

structure has arisen from the laws of growth, and has been taken advantage of in the parturition of the higher animals.”

Using Gould & Vrba’s terminology, the skull sutures would be exaptations since they currently enhance fitness by allowing skulls to flex and distort while passing through the birth canal, but arose for reasons entirely unrelated to their current role in mammalian reproduction. The definition given in their 1982 paper is “Characters, evolved for other usages (or for no function at all), and later ‘coopted’ for their current role.” [Andrews, Gangestad, & Matthews, 2002] phrase the definition as “a pre-existing trait that acquires a new beneficial effect without modification to the phenotype by selection.”

Because the ad in the term adaptation adds the connotations of selection having sculpted a trait for its current role, tying historical origin with current utility, Gould proposes a reorganization and re-labeling of traits that takes evolutionary history into account alongside current utility. This labeling system is depicted in Figure 3.1. In the new system, all traits that are currently beneficial are simply called aptations, removing the ad that carried connotations about the trait’s history, but keeping the aptus that carries appropriate connotations about current utility. Any traits that are not currently beneficial are nonaptations. Aptations are divided into two groups based on evolutionary history: adaptations, which were shaped by selection for their current beneficial roles, and exaptations which arose for reasons other than their current beneficial roles. Exaptations are divided even further into co-opted adaptations and spandrels, depending on their status prior to their co-optation event.

The term spandrel is from [Gould & Lewontin, 1979], and is borrowed from architecture where it refers to structural aspects of buildings that are not directly part of their design

but follow as necessary consequences of design decisions. Darwin's mammalian skull sutures would be an example of a spandrel. The avian wing, since it was exapted from the front limb of theropod dinosaurs [Gauthier & Gall, 2001], would qualify as an example from the other group of exaptations, co-opted adaptations.

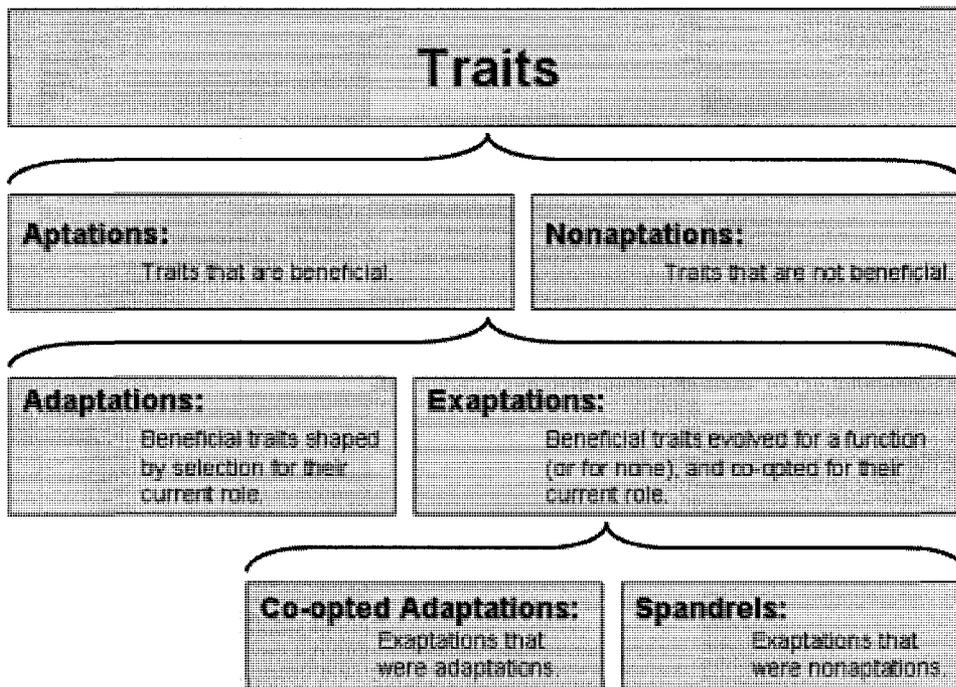


Figure 3.1 A breakdown of traits based on current and past utility and origins.

As for the term trait itself, it is defined very broadly and encompasses almost any heritable aspect of an organism. [Andrews, Gangestad, & Matthews, 2002] state that biologists use the term trait to refer to “aspects of organisms’ phenotypes”, and give the following elaboration.

“A liberal definition would allow a trait to be any aspect of the phenotype that can be discriminated on the basis of any criterion – its causes, its effects, its appearance, and so on – and would include dispositional traits (e.g., the disposition to develop callouses with friction).. ..This approach does not imply that traits are completely genetically distinct from each other, as two traits with very different effects may have common genetic underpinnings. This is not a

problem, however, because adaptationism is concerned with how traits come into being on account of the effects they have.”

Prior to Gould and Vrba’s 1982 paper suggesting the adoption of the term “exaptation”, and from time to time afterwards, another term had, in fact, been in widespread use in evolutionary biology to describe a process of adaptation of a trait (or to refer to the trait itself) that would later be co-opted for a new function. This term was preadaptation, but as Gould and Vrba point out, it carries with it completely inappropriate semantic connotations of a co-optation planned ahead of time, something blind Darwinian processes are not capable of. [Gould & Vrba, 1982] put the problem this way.

“..we traditionally apologize for ‘preadaptation’ in our textbooks, and laboriously point out to students that we do not mean to imply foreordination, and that the word is somehow wrong (though the concept is secure). [Frazzetta, 1975], for example, writes: ‘The association between the word ‘preadaptation’ and dubious teleology still lingers, and I can often produce a wave of nausea in some evolutionary biologists when I use the word unless I am quick to say what I mean by it.’”

Gould and Vrba propose the new term, at least in part, as a solution to this nagging problem with preadaptation. They suggest, instead, that the term preaptation be used to refer to traits before their co-optation event since, without the ad, the connotations of foreordained future purposes are avoided.

3.2 History of the Idea

[Gould, 2002] discusses a number of prominent figures in the history of non-adaptationist evolutionary thought. Each is an influential figure in the study of various aspects of evolution related to organisms’ traits and their properties that have explanations other than direct sculpting by adaptation through natural selection, such as side-effects,

constraints, or “laws” of growth. Gould places particular emphasis and credit, however, on one surprising historical figure, the philosopher Friedrich Nietzsche (1844-1900).

Gould credits Nietzsche with having stressed the importance of not falling for what [Gould & Vrba, 1982] imply is the major confusion that prevented evolutionary biologists from considering exaptation-based hypotheses in accounting for many aspects of biological form and function. Nietzsche’s concern was not specifically with evolutionary explanations, but was broader in scope. It is the confusion of automatically tying current utility into an explanation of historical genesis when it comes to working out the origins of things, such as traditions, behaviors, etc. Gould credits Nietzsche for having made his, Vrba’s, and Lewontin’s point long before any of the three began to focus attention on its application to biology. The problem is stated thusly, in Nietzsche’s “On the Genealogy of Morals” [Nietzsche, 2008].

“No matter how well we have understood the usefulness of some physiological organ or other (or a legal institution, a social custom, a political practice, some style in art or in religious cults), we have still not, in that process, grasped anything about its origin—no matter how uncomfortable and unpleasant this may sound in elderly ears. From time immemorial people have believed that in demonstrable purposes, the usefulness of a thing, a form, or an institution, they could understand the reasons it came into existence—the eye as something made to see, the hand as something made to grasp. So people also imagined punishment as invented to punish. But all purposes, all uses, are only signs that a will to power has become master over something with less power and has stamped on it its own meaning of some function, and the entire history of a ‘thing,’ an organ, a practice can by this process be seen as a continuing chain of signs of constantly new interpretations and adjustments, whose causes need not be connected to each other—they rather follow and take over from each other under merely contingent circumstances.

“Consequently, the ‘development’ of a thing, a practice, or an organ has nothing to do with its progress towards a single goal, even less is it the logical and shortest progress reached with the least expenditure of power and resources, but rather the sequence of more or less profound, more or less mutually independent processes

of overpowering which take place on that thing, together with the resistance which arises against that overpowering each time, the transformations of form which have been attempted for the purpose of defence and reaction, as well as the results of successful countermeasures. Form is fluid—the ‘meaning,’ however, is even more so. Even within each individual organism things are no different: with every essential growth in the totality, the ‘meaning’ of an individual organ also shifts—in certain circumstances its partial destruction, a reduction of its numbers (for example, through the destruction of intermediate structures) can be a sign of growing power and perfection.”

This is the confusion that [Gould & Vrba, 1982] highlight by giving their now famous example of an exaptation-based hypothesis to explain a certain aspect of hyena anatomy. Female hyenas develop genitalia that very closely resemble those of males in outward appearance. Gould and Vrba cite hypotheses that had been put forward, over the years, to explain the evolution of this similarity, such as a possible beneficial function played by the genitals in a hyena social interaction known as the meeting ceremony in which individual hyenas pair to inspect these parts of one another’s anatomy.

Their point is made when they propose a very reasonable possibility involving a separation of historical origin and current utility that had been overlooked until that time. They propose that high androgen levels in developing females, which are related to their morphology and dominance over the males in hyena society, may cause the development of male-like genitalia as a side-effect. This side-effect may have been exapted for its current utility in the meeting ceremony without the genitals having gone through a transition from normal female genitals to those resembling males’, driven at each step by selection for improved utility in the meeting ceremony. Gould and Vrba do not assert the correctness of this hypothetical exaptation-based scenario, but only attempt to understand why hypotheses like it may have been overlooked.

Although Nietzsche described the problem of confusing historical genesis with current utility very explicitly and within a wider context than just biology, in 1887, even Darwin, in his first edition of *On the Origin of Species* in 1859 was aware of the possible role of functional shift in tracing the origins of traits. In his sixth chapter, in which he addresses various difficulties with his theory of natural selection, he states that “In considering transitions of organs, it is so important to bear in mind the probability of conversion from one function to another..” [Darwin, 1859]. In the same chapter, Darwin mentions two properties of the way in which organs often function in bodies, and points out the inherent flexibility that these properties provide in allowing functions to change over evolutionary time:

“The same organ having performed simultaneously very different functions, and then having been specialised for one function; and two very distinct organs having performed at the same time the same function, the one having been perfected whilst aided by the other, must often have largely facilitated transitions.”

Gould demonstrates how Darwin’s understanding of the many-to-many relationships between organs and functions can facilitate evolutionary changes in function when, in [Gould, 2002] he applies this concept to the exaptations that transformed some of the bones in reptilian jaws into the inner ear bones of mammals – a transition for which a large number of finely-graded transitional fossils is known.

“Darwin presents this argument in a fascinating manner by coupling two apparently opposite facts about redundancy: that a single function can be performed by more than one organ, and that a single organ can perform more than one function. Thus, an organ need not invent an entirely new function in some mysterious manner, but may evolve by intensifying a previously minor use, or by recruiting an inherent but unexpressed potential. Meanwhile, the modified organ can abandon its previous major function because other organs can continue (or intensify) their former operation in the service of the same necessary task.

“Thus, reptilian jaw bones can become mammalian ear bones because they already played some role in sound transmission while they functioned primarily to articulate the jaw of therapsid forebears (the principle of two functions for one structure). They then become free to move into the middle ear because the transitional forms (as demonstrated empirically by such fossils as *Diarthrognathus*, and not only as a reasonable conjecture) possessed a double jaw joint (the reciprocal principle of two structures for one function) – and the bones of the old quadrate-articular joint could then become the malleus and incus of the ear because the new dentary-squamosal joint was already ‘up and running,’ thus avoiding the specter of an inconceivable intermediate with an unhinged jaw.”

3.3 The Importance of Exaptation in Biology

Aside from merely providing another mechanism or mode of operation in the explanatory toolkit of evolutionary biology alongside such things as adaptation, mimicry, arms-races, co-evolution, speciation, the Baldwin effect, genetic drift, and many others, exaptation makes possible a great deal of evolutionary flexibility, and is a vital part of the reason why evolution is such a creative and open-ended process churning out perpetual novelty and complexity. One of the ways exaptation makes more powerful the process of natural selection is by allowing populations to evolve traits whose benefits are not capable of accruing without a substantial minimum amount of structure and complexity already in place. The most famous example, in this case, is probably that of the avian wing, which would seem to require quite a substantial amount of pre-existing development before even the slightest degree of flight ability would be visible to selection.

Without exaptation, the forces of natural selection may appear impotent when it comes to constructing the minimal structural requirements that some traits require before they can provide any benefit or serve their current function. This is how it seemed to one of Darwin’s staunchest critics, the distinguished Roman Catholic professor and member of the Zoological Society of London, George Jackson Mivart (1827-1900). Darwin took

very seriously Mivart's criticisms, which Mivart assembled in his provocatively titled book *The Genesis of Species*. The book was a response to Darwin's *Origin*, attempting to disprove the new theory. The sixth and final edition of Darwin's *Origin* [Darwin, 1859] mentions Mivart by name, and covers each of his objections, including the apparent problem of "incipient stages".

"All Mr. Mivart's objections will be, or have been, considered in the present volume. The one new point which appears to have struck many readers is, 'that natural selection is incompetent to account for the incipient stages of useful structures.' This subject is intimately connected with that of the gradation of characters, often accompanied by a change of function,—for instance, the conversion of a swim-bladder into lungs,—points which were discussed in the last chapter under two headings. Nevertheless, I will here consider in some detail several of the cases advanced by Mr. Mivart, selecting those which are the most illustrative, as want of space prevents me from considering all."

Although Darwin got the exaptive relationship between swim-bladders and lungs backwards (the conversion is now thought to have been lung-to-swim-bladder) the principle of invoking changes in function is a valid one, and highlights a way in which natural processes of evolution can construct useful traits that might otherwise appear incapable of construction by gradual degrees with improved utility at each step. All adaptations have beneficial functions. All adaptations also produce side-effects, and these can be amplified, harnessed, and modified by selection should they prove beneficial. A trait exapted for one of its side-effects will already carry with it "fully" evolved structures and complexities, which must, by definition, satisfy the minimal requirements for their beneficial functions. In some cases these can be functions like those encompassed in Mivart's "incipient stages" criticism.

A second way in which exaptation can be considered to increase the power and flexibility of the evolutionary process is by exposing to the forces of selection a whole new regime of creative variations upon which they can feed – a regime potentially many times richer and more complex than that provided by genetic mutations alone. This is what [Gould, 2002] refers to as the “exaptive pool”. The pool is a second source of variation, parallel to mutation, on which selection can act, and which enhances the evolvability of the population. He describes it as follows.

“Clearly, organisms and populations maintain what we might call a ‘fund’ or ‘pool’ of potential utilities now doing something else, or at least doing no harm. I propose that we designate this ground of evolvability as ‘The Exaptive Pool’.. ..The exaptive pool represents the structural basis of evolvability, the potential vouchsafed to future episodes of selection.. ..in a world of strongly polyhedral objects always and ineluctably built with interesting corners and facets that facilitate and channel the directions of evolutionary movement.”

The importance of exaptation is seen very clearly in the fields of genetics and molecular evolution. By sequencing amino acid sequences in proteins, or DNA sequences in genes, many examples have been found where various molecules have been exapted from earlier predecessors for new and often completely novel roles. A particular group of molecules that seems to show abundant examples of past exaptations in numerous different species is the group of lens crystallins. These are a varied group of water-soluble proteins in eye lenses that are transparent to light in the visible spectrum. The table of examples of exaptations in Appendix A lists a large number of lens crystallins from many different species. In each case the molecular evidence strongly points toward these proteins have been exapted from earlier proteins that served entirely unrelated roles elsewhere in the body. Exaptations at the molecular level are particularly potent because of the phenomenon of gene duplication. When a useful gene is duplicated, one copy can

continue to perform its original function while the second, no longer constrained to remain proficient at its old job, can be co-opted and adapted for new roles. Many proteins, such as the lens crystallins, those that make up hemoglobin in blood and myoglobin in muscle tissue, and many other examples show evidence that they are the results of gene duplications followed by exaptation and divergence. This phenomenon enhances the continuous adaptation of life forms, and the enrichment of their chemical repertoires.

3.4 Definitions

Definitions from [Gould & Vrba, 1982], [Gould & Lewontin, 1979], and [Andrews, Gangestad, & Matthews, 2002] for terms for traits and their roles, dividing them into groups based on relationships between current utility and origins, are given in Table 3.1.

Term	Definition
Trait	"..any aspect of the phenotype that can be discriminated on the basis of any criterion – its causes, its effects, its appearance, and so on – and would include dispositional traits (e.g., the disposition to develop callouses with friction)..". These can be divided into aptations and nonaptations. [Andrews, Gangestad, & Matthews, 2002], [Gould & Vrba, 1982]
Adaptation	"Any feature that promotes fitness and was built by selection for its current role (<i>criterion of historical genesis</i>). The operation of an adaptation is its <i>function</i> ." [Gould & Vrba, 1982]
Exaptation	"Characters, evolved for other usages (or for no function at all), and later 'coopted' for their current role." These can be divided into co-opted adaptations, and spandrels. [Gould & Vrba, 1982], [Gould & Lewontin, 1979]
Aptation	Both adaptations and exaptations. Any trait that is currently beneficial. [Gould & Vrba, 1982]
Preaptation	Instead of <i>preadaptation</i> , to refer to exaptations before their cooptation event. [Gould & Vrba, 1982]
Nonaptation	A trait that is not an aptation. [Gould & Vrba, 1982] consider the pool of available nonaptations as "a source of raw material for further selection" and exaptation, akin to mutations.
Primary exaptation	The co-opted trait, at the moment of its co-option. [Gould & Vrba, 1982]
Secondary adaptation	The subsequent accumulating changes in a <i>primary exaptation</i> produced by selection for its new functionality after it is co-opted. [Gould & Vrba, 1982]

Spandrel	An exaptation of “..parts present for reasons of architecture, development or history.” [Gould & Lewontin, 1979]. Or, simply put, a co-opted nonadaptation.
Effect	“..the operation of a useful character not built by selection for its current role..” [Gould & Vrba, 1982]
Function	The role of a trait that was shaped by selection for that role. [Gould & Vrba, 1982].

Table 3.1 Definitions for types of adaptations, their roles, and processes that produce them.

The definitions provided by Gould, Vrba, and Lewontin are by no means etched in stone. Other authors have suggested subtle variations on some of them. In the case of effect vs. function, for example, [Andrews, Gangestad, & Matthews, 2002] make the latter a subtype of the former. They define an effect to be “..the way (or ways) in which an aspect of the phenotype interacts with the environment.”, and a function to be “The effect that causes the trait to evolve..”. For Gould, then, only primary exaptations have effects, and once secondary adaptations begin accumulating, that effect becomes a function. For [Andrews, Gangestad, & Matthews, 2002], on the other hand

“An exaptation is a pre-existing trait (i.e., one that has already evolved) that acquires a new beneficial effect without being modified by selection for this effect (i.e., it takes on a new role, but was not designed for it by selection). Because the beneficial effect did not contribute to the trait’s evolution, the effect the trait is exapted to is not a function but just an effect..”

The same statement could easily have been made by Gould, with perhaps only minor changes (“beneficial effect” would be redundant, and “just an effect” seems to betray Andrews, Gangestad, and Matthews’ preference for counting functions as types of effects whereas Gould seems to forbid the overlap). Andrews, Gangestad, and Matthews do not make any issue out of the differences, and indeed the conflict does not seem to carry any great import.

3.5 Examples of Exaptation in Biology

Although the use of the term exaptation has not received universal acceptance within evolutionary biology, the role of functional shift in the history of life is not seriously disputed. Much of the controversy over exaptation seems to revolve around whether the mechanism is non-Darwinian, the degree of its importance, and whether or not it constitutes a rival alternative to the so-called adaptationist program or whether it's merely an added emphasis on something that has been widely understood and accepted all along.

However history may ultimately come to pass its judgment on this dispute, there can be no doubt that the kinds of functional shift encompassed by the term exaptation play a key role in the modern evolutionary theory's accounting of the development of species and their traits. Three interesting examples are given below. The interested reader is directed to Appendix A for a longer list of examples ranging across the gamut from those that are highly speculative to others that are more or less undisputed.

Shell space for egg-brooding in some snails

Some species of snails brood their eggs within a space available in their shells. This space, called the *umbilicus*, seems to be a mere byproduct of the way shells develop as snails grow. Evidence strongly suggests that the snails that use the space for egg brooding evolved after those that possess it but don't use it, making this a spandrel that was exapted for brooding [Gould, 2002], [Andrews, Gangestad, & Matthews, 2000], and [Lindberg & Dobbarten, 1981].

Female hyena genitals

The genitals of female hyenas very closely resemble those of males. This seems to enhance fitness in what's called the *meeting ceremony* where two hyenas pair to inspect each other's genitals. It is hypothesized that this roll for the female genitalia is an exaptation of a non-aptation. The idea is that the resemblance to males is a side-effect of high androgen levels in females, the genesis of which is tied to the social behavior and morphology of the females; they are larger than the males and dominate them within the social framework of hyena society. [Gould & Vrba, 1982] put forth this hypothesis, though they did not assert that it is correct, only that it had been an overlooked and perfectly plausible possibility for quite a long time due at least in part to what they perceived as the constraints of a conceptual framework in evolutionary biology devoid of a labeled distinction between aptations, adaptations, and exaptations. In particular they felt the conceptual framework was most lacking in that it had no term for co-opted non-aptations. The hyena hypothesis was meant to illustrate their point.

Mammary Glands

Mammary glands are thought to have evolved from immunoprotective glands that produced antibacterial secretions on the skin [Vorbach, Capecchi, & Penninger, 2006]. This hypothesis, though still speculative, fits well with the evidence that the protein α -lactalbumin, in breast milk, is an exapted version of the antibacterial enzyme lysozyme [Dickerson & Geis, 1969], and with the fact that some remaining members of early offshoots in the mammalian line, such as platypuses, possess mammary glands but lack teats, instead simply secreting milk directly through the skin.

Chapter 4 Exaptation in Evolutionary Computing

Examples of exaptation in the EC literature are few and far between. Most involve functional changes of degree instead of the more abrupt and drastic changes that a biologist would recognize as exaptation. Some do not use the terms “exaptation”, “co-opting”, or “preadaptation”, instead referring to a particular system as using an “incremental approach” [Kodjabachian & Meyer, 1998a], or using “sequential evolution” [De Garis, 1991], or similar terms. The current chapter describes some of these systems, and the nature of the functional changes and exaptations they employ.

4.1 Exaptation Systems with Implicit Fitness

4.1.1 Spandrels in an ALife 2D Cellular Automaton

The paper “Simulation of Exaptive Behavior” [de Oliveira, 1994], describes the use of a 2D cellular automaton ALife system, called “Enact”, for experiments involving the evolution of exaptations. De Oliveira configured Enact in such a way that the exaptation of adaptations was forbidden, and only nonadaptations could be co-opted. Thus, all exaptations in Enact were spandrels. As an ALife system, Enact’s fitness was endogenous and not explicit.

The exaptations involved co-opted patterns of movement in a 2D environment that happened to confer improved relative reproductive success to ALife agents in the cellular automaton. De Oliveira’s explorations into the simulation of spandrel evolution were prompted by the statements of [Gould & Vrba, 1982] who had remarked that, were it not

for the concept of exapted nonaptations, or spandrels, their paper would not have been written. Spandrels constitute a large and important portion of Gould's so-called "exaptive pool", which acts, together with mutation, as a source of variation, and one which does not seem to have well-developed counterparts in much of EC.

4.1.2 Preadaptation of Neural Networks in a Changing Environment

Another ALife system involving exaptation was that of [Lund & Parisi, 1995]. Their system involved the evolution of neural networks in a changing environment. They examined populations that had already had an evolutionary history in which they had evolved specialist strategies for one of the food sources in the environment. These evolved specialist strategies preadapted the populations in such a way that they were able to evolve new specialist strategies when their food source was removed, and did so faster than populations that had not been preadapted (populations that began from scratch).

4.2 Exaptation in Systems with Explicit Fitness

4.2.1 Incremental Approaches in the Evolution of Neural Network Robot Control Systems

[Kodjabachian & Meyer, 1998a] describe an exaptive system using GP to evolve neural network controllers for straight-line locomotion in a simulated hexapod. Their neural network controllers have their architecture and weighting evolved using a GP encoding scheme called SGOCE, or Simple Geometry-Oriented Cellular Encoding, which is based on the cellular encoding methods of [Gruau, 1994] and [Gruau & Quatramaran, 1996]

who used GP to evolve locomotion controllers for *animats*. The execution of a GP tree in SGOCE dictates the development of a neural network embedded in a 2D plane.

There is an exaptive approach since it uses two steps with different fitness functions, in which the results from the first step are an input into the second. They refer to it as an “incremental approach”. The ideas are based on those of [Beer & Gallagher, 1992] who used an EC system to evolve control systems for a simulated insect, on the step-wise scheme used in [de Garis, 1991]’s “sequential evolution”, also a multi-step exaptive technique, which was used to evolve walking behavior, and on the work of [Harvey, Husbands, & Cliff, 1994] who showed, although without enough runs for statistical significance, improved performance using what they called an “incremental evolutionary methodology” for evolving robot control systems versus a non-incremental setup. They are also based partly on the work of [Lewis, Fagg, & Solidum, 1992], who evolved simple neural controllers for control of leg motion in a real hexapod robot. Lewis, Fagg, and Solidum’s evolved leg-motion controllers would then be combined with a second controller and together the pair would be further evolved as a whole for the new higher-level task of coordinating individual legs.

Kodjabachian and Meyer’s incremental approach was to first evolve a controller that would produce straight locomotion, simply to make the hexapod mobile. The resulting controller would then become a fixed component in the second phase. To use the terminology from biology, their adaptations from the first phase would be exapted but then forbidden from accumulating secondary adaptations in the second phase. Instead of allowing the controller from the first phase to continue adapting in the second phase, a

second controller would be evolved alongside it, but with special developmental primitives added to the SGOCE system to allow the neural network to establish outgoing connections to the neurons in the fixed controller. The pair would then be evolved using a fitness function that rewarded the ability to have locomotion switched on or off based on the properties of an incoming external signal.

With this technique they were able to successfully evolve controllers that would allow locomotion to be controlled by an incoming signal. This work demonstrated the ability of an incremental approach to produce the desired results, but not its superiority at doing so since a non-incremental approach was not performed for a side-by-side comparison.

Their follow-up work from the same year, [Kodjabachian & Meyer, 1998b], using the same incremental technique and SGOCE system, evolved controllers that would allow the hexapod to avoid obstacles while at the same time following a gradient. This was a three-step incremental approach. In the first phase a locomotion controller was evolved as before. The best controller from this phase became a fixed component in the second phase. The second phase added a new module, with extra SGOCE primitives for establishing intermodular neural connections, to produce gradient-following behavior. The best result from the second phase became, in turn, a fixed part of the third, which added another module for obstacle avoidance.

As with their earlier work, this too demonstrated only the ability of the incremental approach to evolve these complex hierarchical behaviors, but not its superiority, however likely it may seem in this instance. No results are reported for using a non-incremental approach for comparison.

4.2.2 Symbiotic Composition for Crossing Fitness Saddles

[Watson & Pollack, 2001] describe an EC system using a kind of functional change similar to that of [Kodjabachian & Meyer, 1998a & b] and [Lewis, Fagg, & Solidum, 1992] in that their evolved systems become exapted as modules in a larger whole. They refer to their composition mechanism as “symbiotic composition”, and liken it to mechanisms of composition employed in what they call “major transitions” in biological evolutionary history. As examples of major transitions they list the organizing of individual genes into gene networks, bacterial cells combining to make eukaryotic cells, eukaryotic cells aggregating to construct multicellular organisms, and large groups of multicellular organisms forming structured societies. At each major transition, evolved entities are combined into new wholes.

Watson and Pollack’s system uses a composition mechanism to allow the population to overcome “fitness saddles”, a reference to the concept of fitness landscapes [Wright, 1932]. When a population has converged to a local optimum in the fitness landscape, there are several known mechanisms that can allow it to cross the valley, or “fitness saddle”, between the current optimum and a nearby higher peak. As examples, Watson and Pollack list genetic drift and the shifting balance theory [Wright, 1932], exaptation, neutral networks [Kimura, 1983], [Schuster et al., 1994], extra-dimensional bypass [Conrad, 1990], [Cariani, 2002], ontogenic process, and landscape dynamics. They don’t count the composition mechanism as a form of exaptation. As will be argued in Chapter 5, however, it could be seen as a form of weak exaptation where functionality changes, but not drastically.

They suggest that, due to the nature of the composition mechanism, its ability to construct wholes more complex than their parts, there is a magnitude difference between the sizes of fitness saddles that can be crossed by the example mechanisms listed above versus those crossable by composition. They also describe it as a “scale-invariant” adaptation mechanism, although it is unclear whether or not they consider the other mechanisms to be scale-invariant. Entities can be composed at any level of organization. The same could be said of exaptation and functional shift in general, making it a scale-invariant mechanism as well.

In order to demonstrate the crossing of fitness saddles and the scale-invariance of a symbiotic composition mechanism, they construct a composition model called SEAM, or Symbiogenic Evolutionary Adaptation Model [Watson & Pollack, 2000]. SEAM is tailored to be a scale-invariant landscape with saddles at all scales. It uses a binary string HIFF, or Hierarchical IFF, function for strings of length 2^p where p is variable that can increase monotonically in an evolutionary lineage of descent, and acts as an index of the problem scale. The HIFF in SEAM is deliberately constructed so that composition has a much greater chance of crossing saddles at all scales than does mutation and crossover. They conducted experiments using a 64-bit HIFF in SEAM, with composition as a variation operator, and compared it to both a standard GA and an RMHC, or Random Mutation Hill Climbing approach. The SEAM system, with composition, substantially outperformed both the GA and RMHC, climbing to higher values of p while the others became trapped.

4.2.3 Exaptation in Dynamic Landscapes

Another domain for applying exaptation is examined in [Fentress, 2005]. It involves changes in function that allow island populations to pre-evolve solutions to problems before they emerge in a changing environment. Fentress tests whether or not an exaptation-based GA will consistently find better solutions than a regular GA in a particular dynamic environment. The comparisons between the exaptive approach and the regular GA are in terms of solution quality, as well as in terms of the speed with which quality solutions are found.

The dynamic environment chosen was Branke's Moving Peaks benchmark [Branke, 1999] in which a number of potentially overlapping hills are present. These hills can gradually change in height, width, and position, requiring the population not only to adapt, but to track maxima as they move, and to find new global maxima as they come and go. Fentress' exaptive GA uses a number of small "tribe" populations, each performing local search in different regions of the landscape. Tribe sizes vary dynamically depending on their fitness, and each tribe's landscape is modified to repel it from the others.

As the moving peaks landscape changes, solutions evolved in a tribe population may find themselves in a better part of the primary landscape than other tribes. When this occurs, they are exploited by the diversion of more processing power into the tribe through an increase in population size. This allows the group of tribes, as a whole, to maintain diversity, to explore many regions of the landscape by having each adapt to slightly modified versions of the primary landscape, and to exploit successes by weighting

exploitation, via population size, using a tribe's fitness. The tribes may find themselves preadapted for future changes by virtue of exploring nearby local maxima of the dynamic landscape. Or, as [Fentress, 2005] puts it,

“..a model based on biological exaptation, or preaptation, can allow the GA to gather more information about the search space it is evolving in, specifically the existence of solutions which may at a later point in time be global optima.”

Fentress' dynamic landscape exaptation GA takes advantage of the tendency of the moving peaks landscape to undergo changes that are gradual, not abrupt. Solutions found at one tribe's local maximum during evolution may find themselves at the global maximum when the landscape changes. Fentress' multi-population exaptive GA was shown to outperform both a standard GA and a hill-climber in two versions of the moving peaks problem.

A second domain for an exaptive GA in [Fentress, 2005] is the evolution of neural networks for Boolean functions. The exaptive approach involves the use of “sister” populations. These are extra island populations that evolve in modified versions of the fitness landscape. The scheme is based on the SBGA, or Shifting Balance Genetic Algorithm of [Oppacher & Wineberg, 1999 & 2000] and [Wineberg & Oppacher, 2000], a multi-population GA for changing environments based on Sewall Wright's Shifting Balance theory [Wright, 1932] in biology.

Each sister population evolves, during an initial epoch in Fentress' system, to fit an easier Boolean function than the goal function. The easier version of the function is designed in such a way that its population is expected to have a neural network structure somewhat similar to one that would solve the true goal function despite having poor fitness under

the that goal function. As an example, if the inputs were merely three bits and the output a single bit, the function

$$\begin{aligned}
 f(0,0,0) &= 1 \\
 f(0,0,1) &= 0 \\
 f(0,1,0) &= 0 \\
 f(0,1,1) &= 1 \\
 f(1,0,0) &= 1 \\
 f(1,0,1) &= 1 \\
 f(1,1,0) &= 1 \\
 f(1,1,1) &= 0
 \end{aligned}$$

could be defined using just the ordered outputs by giving the string “10011110”. This string would represent the goal function in the main population. Each sister population would evolve to fit an easier version of the function, one in which the influence of one of the three input bits is removed. In this example, the sister populations would each evolve to fit one of the following functions.

```

“1001****”
“****1110”
“10**11**”
“**01**10”
“1*0*1*1*”
“*0*1*1*0”

```

The asterisk symbols denote that the output from the neural network can be either zero or one (i.e. both are acceptable). It would seem likely that solutions to these sister functions would contain high degrees of structural similarity to solutions for the true goal function, except for neural network weights associated with one of the three input neurons.

After this first epoch is complete, the GA switches to an island model where all populations are evaluated on the goal function. This allows structures preadapted in the first epoch to be exapted for use in solving the real goal function. Fentress refers to these

preadapted structures as “transitory or partial solutions”. The exaptive approach produced results roughly comparable to an island-model-only GA, and superior to a standard GA. Not enough runs were collected to associate any statistical significance to the results.

4.2.4 Gradual Functional Change for Evolving Differentiation in Gene Regulatory Networks

[Knabe, Nehaniv, & Schilstra, 2006] developed an artificial Gene Regulatory Network (GRN) EC system in which evolution was used to produce phenotypes that would exhibit two different behaviors depending on whether or not they had received a particular input signal. A GRN is an abstract model of a biological counterpart with the same name.

These consist of a number of simulated genes and their products, as well as inputs from the environment. Genes in a GRN produce their products at rates that are influenced by the presence of other products within the virtual “cell”, whether produced by other genes, or entering from the environment. GRNs can produce complex feedback loops, and activation cascades in which genes are switched on or off in coordinated orchestration. System performance is usually measured in terms of quantifiable aspects of one or more gene products being treated as outputs.

The “cells” in the GRNs of Knabe, Nehaniv, and Schilstra consist of “proteins” and a fixed-length genome containing a fixed number of genes. Gene activity is controlled by regulatory sites. A regulatory site consists of one or more protein binding sites. Protein production for a gene is positively or negatively influenced by whether or not other proteins are being bound at the gene’s regulatory site.

Knabe, Nehaniv, and Schilstra's GRNs were evolved to exhibit a switch-like behavior, producing different output patterns depending on whether a particular input was present or absent. They found that an exaptive approach, in which the GRNs' fitness function was changed gradually, yielded superior performance on average compared to a GRN evolved for switch-like behavior all the way through a run, without any fitness function changes. They also observed that the GRNs evolved via this exaptive approach gave results that were not just superior in terms of raw fitness values, but were also more robust (i.e. less variable in performance). As Knabe, Nehaniv, and Schilstra put it:

“..the lineage's evolutionary history seems to be very important in determining the probability that a switch between two behaviors can be found. Comparing with the immediate requirement for a switch between behaviors we found that in the gradual case final GRNs usually showed better success with less variability in performance, the latter indicating an increase in evolutionary robustness.”

Chapter 5 The Carryover to Evolutionary Computing

[Banzhaf et al., 2006] suggest a set of guidelines to steer the research agenda within the EC community at large. The guidelines are intended to encourage the development of a new field they call “Computational Evolution” (CE). They contrast CE with the current palette of EC techniques, which they refer to collectively as “Artificial Evolution” (AE).

AE includes such approaches as GP, evolutionary programming, GAs, evolutionary strategies, learning classifier systems, and others. Deficiencies of current AE techniques often (but not always) include the use of relatively small genomes, whatever the corresponding data structure, simple and direct mappings from genotype to phenotype, lack of self-modification mechanisms or feedback, and evolution toward fixed foreordained objectives. Also, AE techniques tend not to be geared toward modes of evolution that are more open-ended, generating perpetual novelty, as in nature.

Banzhaf et al. propose “..a richer paradigm for algorithms inspired by evolution..”. They describe the potential utility of this richer paradigm as follows.

“This approach will produce algorithms that are based on current understanding of molecular and evolutionary biology and could solve previously unimaginable or intractable computational and biological problems”

CE is built upon the foundations of AE, but with specific concepts from evolutionary biology that are in need of being more fully incorporated. These as-yet-underdeveloped or missing facets include, but are not limited to, the following.

- A focus on physicality and embodiment instead of the kinds of symbolic genetic materials seen in AE that reflect its current focus on programmability.
- Embracing side-effects instead of eschewing them.
- Nested data and control structures interacting at all scales simultaneously.
- Feedback loops between control structures and objects, enhancing the ability to respond to change.
- Selection pressures “felt” at multiple levels, from individual genes to entire species and ecosystems.
- Mechanisms for reproductive isolation, facilitating speciation, either sympatric (without geographical separation) or allopatric (with geographical separation).
- Co-evolution of spontaneously evolved weakly-coupled modular units.
- Ecological interactions at all scales causing environment changes that encourage constant adaptations that provoke still further ecological interactions, etc.
- Multilayered and complex processes that map genotypes to phenotypes, akin to the development of multicellular organisms from a single fertilized egg.
- Exaptation mechanisms allowing the recruitment of components to purposes other than those for which they evolved.
- Metabolisms and the dynamic transformation of resources and materials within a system of limited material and energy resources.

Regarding the importance of exaptation in CE, they state the following.

“The recruitment of a component selected for one use to a new purpose (exaptation) could be an important source of innovation in natural morphogenesis and a source of new functionality.. ..Exaptation presupposes a rich set of interactions between components and functions, so that a change in one item

changes others; it might therefore happen that a side-effect is more valuable than the feature originally selected for, leading to amplified selection and eventual fixation of the new function.”

The following sections discuss several issues related to exaptation as it applies to EC (or AE, or CE). Some terminology and framework suggestions are proposed, and several aspects of exaptation as well as several types of exaptation are examined with an eye toward the integration of functional shift mechanisms into EC.

5.1 Use of the Term “Preadaptation” in EC

The misleading nature of the semantics of the term preadaptation in biology, described in 0, disappears in many cases in the carryover to EC. Except in some ALife systems in which fitness is implicit instead of being tied to a specific function, the term may finally find a legitimate home in exaptive EC systems. In such systems, if they involve exaptations and functional shifts deliberately planned ahead of time, it may make sense to speak of a population being preadapted to some fitness regime before it encounters it.

In describing several exaptive EC systems in Chapter 6, the term will be used in this way from time to time. To preadapt a population will be to prime it, by evolution, with building blocks, behaviors, solutions, strategies, or structures that will be useful in some preordained fitness regime or environment to be encountered in the population’s future. Since such systems are designed with foresight, it can be said that the purpose of earlier fitness functions is to preadapt a population for later fitness functions.

It should be noted that, even within an EC system using explicit fitness functions, it is, in principle, possible to encounter exaptations that are not planned ahead of time. In a

sufficiently rich and complex environment, a population of mobile autonomous robots whose control systems and morphologies are coevolving, to take one hypothetical example, may exhibit exaptations that are not of the preordained type described above. Even with a simple fixed fitness function, such as distance traveled in a fixed time, limbs that serve a propulsion function, by repeatedly striking and pushing off the ground, for example, could conceivably be exapted to a new function in which they do not contact the ground at all but act as body-balancing stabilizers. Other plausible examples like these can easily be imagined. Such exaptations, it may be, only arise in systems complex and expressive enough to allow the emergent evolution of modules and sub-functions that perform different tasks but which, together as a whole, serve the “true” function – the explicit fitness function.

5.2 Confusion of Current Utility with Historical Origin

Evolutionary biologists examine whatever evidence is available and attempt to make inferences about the past. One particular aspect of this inference process, stressed by [Gould & Vrba, 1982] and discussed in 0, is that the present and past roles of traits may not be the same. Conflation of the two roles is fortunately far less likely for those retracing evolution in an EC system. EC practitioners find themselves in a better position than biologists when it comes to collecting the evidence needed to trace the origins of traits. In principle, one should be able to record evolutionary trajectories, genotypes, transitional forms, environments, parent-offspring relationships, specific details from applications of the variation operators, and other statistics in as much detail as desired, within the processing and storage capacities available. This should greatly simplify the

task of understanding historical origins in simulation. In the carryover from biology to EC, the potential for confusion of current utility with historical origin is, therefore, greatly reduced if not entirely vanished.

5.3 Types of Exaptation and Functional Shift in EC

One way to better understand exaptation in EC is to consider how functional changes might be divided into categories. Different types of exaptation may require different orders of complexity or different setups in systems that give rise to them. One possible advantage of a division into types is that it may provide a more instructive way of appraising the extent to which EC systems mirror the evolutionary processes in biology. One can consider not only whether or not nature and a given simulation of interest exhibit exaptations, but also the types of exaptations exhibited, allowing a more fine-grained basis for comparisons and contrasts. Presumably, [Banzhaf et al., 2006]’s field of CE will be composed of systems that are rich, expressive, and complex enough to give rise to all of the types of exaptation found in nature, and in similar ways.

None of the categories described below should be treated as black-and-white distinctions. Any adaptive system of sufficient complexity is going to be “messy” and may allow for the occasional case that straddles the boundaries or gray zones between categories.

5.3.1 Deliberate vs. Emergent Exaptation

Exaptations can be divided into deliberate and emergent types. Deliberate exaptations occur when both the function served by a trait before exaptation and the function served after are specifically chosen ahead of time and do not arise of their own accord.

Deliberate exaptations are those that can rightfully be called preadaptations when referring to the exapted systems before the co-option event. Some, but not all exaptations in EC, are of this type.

Emergent exaptations are those for which either one or both of the functions to either side of the co-option event are not specified ahead of time by the programmer or user. All exaptations in biological evolution are of this type. The functions served are emergent properties of an evolving system and can come and go as conditions change. ALife systems with implicit fitness may exhibit emergent exaptations. EC systems with explicit fitness functions may produce emergent exaptations as well, provided they are sufficiently complex and can allow for the spontaneous development of behavior repertoires or interacting modules within a whole.

5.3.2 Strong vs. Weak Exaptations

Exaptations can be divided into strong exaptations and weak ones. This division is very subjective. A number of diametrically opposing adjective pairs could be used to describe the distinction between the two such as drastic vs. subtle, differences of type vs. differences of degree, or large vs. small changes in function. Strong exaptations are those for which the differences between the pre-co-option and post-co-option functions are substantial. Weak exaptations are those for which the differences are a matter of degree or subtlety. The weak end of the spectrum blends seamlessly into EC systems that are commonly described as possessing dynamic fitness landscapes, such as the Moving Peaks problem of [Branke, 1999], mentioned in the previous chapter. Of all the distinctions

between types of exaptation, this is the most subjective and spectrum-like, with a wide and fuzzy boundary between the two extremes.

5.3.3 Cumulative vs. Replacement Exaptations

Exaptations can be divided into cumulative and replacement types. Cumulative exaptations involve traits or systems adopting new functions while maintaining (or even continuing to improve) the old functionality. Replacement exaptations involve a complete loss or severe degradation of the old functionality as the trait is co-opted and adapted for the new one.

This particular distinction can be tracked and visualized in an EC system by plotting population progress in a two-dimensional fitness space. Idealized hypothetical examples of such graphs are shown in Figure 5.1. Actual examples from implementations of two exaptive GAs called the two niche exaptive GA and the combination-fitness exaptive GA are presented for both cumulative and replacement exaptation in the next chapter.

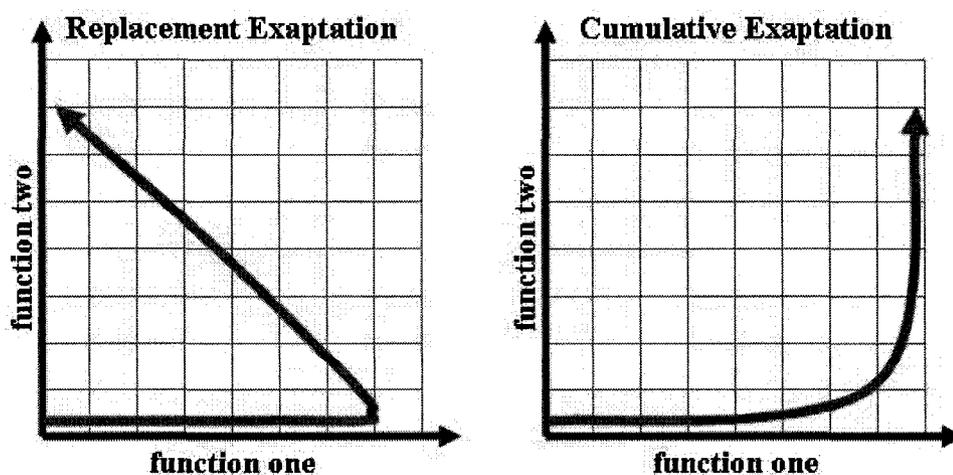


Figure 5.1 Hypothetical 2D fitness graphs for cumulative and replacement exaptation.

Examples from biology come in both types. The exaptation of penguin wings in the transition from a flying bird to a flightless waterfowl is one example of a replacement exaptation in nature. The exaptation of the elephant's trunk from an organ used for smelling to an appendage for grasping and manipulation of objects is an example of cumulative exaptation.

The picture is made more complex by the fact that, in biology the functional transitions for replacement exaptations may span many generations instead of being instantaneous. This may even be the norm, but it need not necessarily be the case in EC where functional changes, especially deliberate ones, can indeed be instantaneous. Also, since there is a many-to-many relationship between traits and functions, some functions may degrade after an exaptation while others are maintained, even within the same trait, meaning that the terms "cumulative" and "replacement" may only apply relative to certain aspects of any particular exaptation.

5.3.4 Composition as a Form of Weak Exaptation

Exaptations can be divided into compositional and non-compositional types.

Compositional exaptations are those in which the transition involves the exapted system being combined with other systems, or absorbed by another, resulting in it becoming a part of a whole from which it had previously been separate. Non-compositional exaptations involve some functional change but without the construction of new higher-level entities.

An example of a compositional exaptation in nature is Lynn Margulis' theory of the endosymbiotic origin of organelles like mitochondria and chloroplasts [Margulis, 1981], which may have originally been primitive bacteria with an existence separate from the cells of which they are now a part. Examples from EC include the symbiotic composition GA of [Watson & Pollack, 2001], which combined individuals to make larger entities as a means of crossing saddles in the fitness landscape, and the incremental approaches for evolving neural network control modules of [Kodjabachian & Meyer, 1998a & b], which constructed controllers for high-level behaviors by composing the results of early epochs into larger systems evolved in later epochs.

Compositional exaptations are likely to be weak exaptations in most cases. As a system is combined with others, its functionality may not change drastically. Nevertheless, in sufficiently versatile and complex simulations, the incorporation into a new higher-level whole will carry with it new constraints and pressures on the exapted trait. Some aspects of the old functionality may become obsolete; others may be altered, amplified, or diverted. This will coax secondary adaptations that reflect the function's changes.

5.4 Shared Genotype-Phenotype Mapping in Exaptive Systems

When devising an exaptive GA or other EC system, one issue that will constrain the particular details of the algorithm design arises from the genetic representation and the mapping from genotype to phenotype. Although it need not be etched in stone, it seems reasonable to assume that such systems are most likely going to keep a fixed representation scheme and a fixed genotype-phenotype mapping through all exaptation transitions, whether deliberate or emergent. If some of the recommendations of [Banzhaf

et al., 2006] are adopted, the representation and genotype-phenotype mapping can itself be subject to evolutionary forces. In such cases, the underlying rules governing the representation and mapping will be the same before an exaptation transition as after. The main gist is that the exaptation transition will be confined to the level of the fitness evaluation, whether implicit or explicit. Barring exceptional clever ‘tricks’ by ingenious programmers, the moment of exaptation in a run or simulation will not necessitate a restructuring or porting of one representation-and-mapping scheme into another.

For exaptive systems in which the transitions are deliberate, it will fall to the programmer to devise a single encoding and mapping scheme that is both rich enough and flexible enough to encompass a wide range of phenotypes, solutions, or strategies for each fitness task to which a phenotype may be applied at any point in a run. This kind of enhanced genetic and phenotypic richness can be seen as a desirable property for an EC system in any case, and thus, the more widespread the efforts to incorporate exaptation into EC, the more the field will automatically be pushed in this positive direction as well.

Chapter 6 Exaptation Experiments with a Simple Genetic Algorithm

6.1 General Description

This chapter presents several variations on a simple exaptive GA possessing, in some instances, multiple niches and a notion of viability wherein only viable individuals can partake in reproduction. In some variations, a population of individuals in a niche can spread to populate other niches even though the destination niches may have a different fitness function. The instances of a population spreading from one niche to another through the migration of individuals across inter-niche bridges constitute examples of exaptation in the GA. Other variations on this GA, such as the one described in section 6.5, involve no inter-niche bridges but instead involve scheduled fitness function changes within a niche. In most variations, structures evolved in one niche meet the minimum requirements in other niches, and can then be co-opted. Most involve niches that are initialized with the same random population initialization method, in which only a single niche will end up with viable solutions to its fitness function, leaving the problems in other niches unsolved until appropriate exaptation events take place to seed their populations.

The viability aspect of the GA is much like the notion of infeasible solutions, which often arises in some GA problems. When this issue arises in a GA, or in EC in general, it is usually the case that a solution categorized as infeasible is one for which the genotype is a perfectly legitimate point in the genetic space being searched but the phenotype it encodes is either nonsensical or is somehow a non-solution to the problem. Examples

include graph cycles that repeat or omit cities in the Traveling Salesman Problem, and knapsack problems in which the knapsack is filled beyond its capacity. Such infeasible individuals are often either a) eliminated by changing the genotype-phenotype mapping, b) repaired, c) penalized, or d) simply discarded when encountered.

The notion of viability in the GA described here imposes harsh reproduction restrictions on inviable solutions. They attain zero fitness, which is the lowest possible fitness in all of the GA's niches, and are strictly forbidden from being chosen as parents when producing offspring. The term viable is preferred over infeasible to denote such individuals for two reasons. First, because a large fraction of the search space is simply declared inviable by fiat in the fitness function, but does not reflect any logical contradictions or inconsistencies arising from the nature of the problem or the genotype-phenotype mapping as is usually the case with individuals labeled "infeasible" in other GAs. The individuals are not infeasible in the sense of being impossible or in the sense of being complete non-solutions. They are merely treated as if they were. Second, the manner in which the individuals are treated in the GA is more like that of inviable offspring in biology, such as mutants that cannot even complete embryogenesis, or sterile hybrids such as mules and hinnies (the offspring resulting from crosses between horses and donkeys). Such individuals in this GA have a fitness of zero and cannot reproduce.

Phenotypes in the simple exaptive GA are 2D structures embedded in a grid called the "game board". The phenotypes are composed of variable numbers of items, called "boxes", which are placed in cells of the grid. Not all fitness functions used in the GA interpret the box arrangement structure in the same way. The use of the term "game

board” for the grid stems from one such interpretation for these structures in which each box in the phenotype plays a role in a simple game that plays out in the grid. Section 6.1.5, below, describes the particular fitness function that follows this interpretation, as well as the rules of the game and the roles for each type of box.

There are several fitness functions applied to populations using identical genetic representations. Each treats the 2D pattern generated by the chromosomes in a different way, or measures a different aspect of the structure. The fitness functions are as follows.

- **Ball Game (BG):** This fitness function treats the pattern of boxes as a strategy for game play in which a ball containing associated points travels through the board, interacting with the boxes. The game score attained by an individual is used in the calculation of its fitness.
- **Largest Enclosed Area (LEA):** This fitness function looks for the largest region of empty cells that is completely surrounded by boxes. Fitness is based on the size of this area and the existence of an unbroken chain of boxes connecting the top and bottom of the grid.
- **Product of Largest Coloured Regions (PLCR):** This fitness function divides the pattern of boxes into separate regions, each containing boxes of the same type (colour). The areas of the largest regions of each colour are used to determine fitness.
- **Sieve:** The Sieve fitness function evaluates the pattern of boxes as though it were a sieve. Fit sieves have many holes (empty cell regions), but none that are too large.
- **Coloured Ring Distances (CRD):** This fitness function scans the grid for instances of particular box configurations called *coloured rings*. All instances of such rings are

compared to others in a pairwise fashion. The number of distinct Manhattan distances between rings is used to determine fitness.

This chapter examines several experiments involving different versions of the GA, each looking at exaptation in a different context. These contexts are as follows.

- **Interlocking Complexity:** A gradual change of function (exaptation) involving the BG fitness function is used in the evolution of box configurations that exhibit *interlocking complexity*, a term used to refer to systems containing multiple parts and for which all parts are vital to the function served by the system. This is done using the *Single-Niche Exaptive GA*.
- **Speciation and Exaptation:** The *Two-Niche Exaptive GA* involves two populations or “niches”, each with a different fitness function. A one-way migration bridge between the two niches allows structures evolved in one niche to become exapted to serve the fitness function in the second, once minimal structural requirements have been evolved. The two populations, once successfully seeded, continue along separate evolutionary paths, and diverge to the point of speciation. The *Four-Niche Exaptive GA* takes this process further, and allows a single population to spread across multiple migration bridges to seed and adapt to four different fitness functions in a predictable sequence.
- **Augmented Fitness and Visualization of Exaptive Dynamics:** The *Combination-Fitness Exaptive GA* shows a population evolving to suit the LEA fitness function. Its performance is shown to increase dramatically when a preadaptation epoch is used involving an augmented version of the fitness function that incorporates the BG fitness

function. The modified fitness function allows the population to make faster progress in maximizing the LEA fitness function.

The next sections detail the workings of the GA, its genotype-phenotype mapping, its variation operators and so on. Subsequent sections describe four versions of the GA. The first is the single niche exaptive GA, which uses the mechanism of gradual functional change to demonstrate the evolution of so-called “interlocking” or “irreducible” complexity (systems of multiple parts where each part is vital in maintaining performance). The second is the two niche exaptive GA, which adds a second niche with a different fitness function. It uses exaptation to facilitate speciation between the two niches. The third is the four niche exaptive GA, which uses several stages of exaptation and secondary adaptation to allow the GA to step from one region of phenotype space to another. The fourth is the combination-fitness exaptive GA, which uses the product of two fitness functions during a preadaptation epoch in order to boost the performance of the GA with respect to only one of the fitness functions. All fitness functions used in the simple exaptive GA are inventions of the present author, constructed specifically to conduct the various experiments described below, and are not functions used previously in the EC literature except in publications relating specifically to this work.

6.1.1 GA Parameters

The same GA parameters and operators are applied to both niches in the two niche exaptive GA and are as follows. The population size is 50 with single individual elitism. Selection is done by tournaments of size two, and only viable individuals are eligible for selection. On average, ten percent of offspring are produced by uniform crossover, with

the rest produced by mutating a copy of a single selected parent. Chromosome length is variable and can be modified by mutation and crossover operators. Since the experiments described here are concerned with the high-level speciation-like behavior of the GA, among other behaviors, and with the role exaptation plays in those behaviors, and not with the efficient generation of highly fit solutions, as would be the case with most GAs, no effort has been made to tune these parameters for optimal performance.

The settings described above are shown below in Table 6.1. Unless otherwise stated, these apply to all four versions of the exaptive GA described in the sections below, and to all niches within each version.

Parameter	Value
Population size	50
Tournament size	2
Crossover rate (remainder produced by mutation, with crossover-produced individuals not subsequently mutated)	10%
Crossover operator	uniform
Chromosome length	1 to 400

Table 6.1 Parameters and values for the exaptive GA.

6.1.2 Representation

All niche fitness functions assign a non-negative integer value to individuals' phenotypes. The phenotypes are two-dimensional structures constructed within a grid of 30 rows and 14 columns, called the game board. Each grid cell in the game board is blank by default until the individual's genes specify otherwise. Nonblank grid cells can have one of four types of objects placed in them, called boxes, designated by the letters S, A, L, and R. The meaning and role of each box type depends on the fitness function being applied. Each of the two fitness functions is explained below.

Individuals' chromosomes are composed of three strands of equal length. This length can vary between individuals but not between strands in the same individual. Strand one is a sequence of game board row indices with values ranging from 0 to 29 since the game board has 30 rows. Strand two is a sequence of game board column indices with values ranging from 0 to 13 since the game board has 14 columns. The third strand is a sequence of letters (either S, A, L, or R) representing box types. The phenotype is decoded from the genotype by simply reading the three strands in lockstep from left to right and placing boxes of the specified types in the specified cells on the game board. If the same cell is specified multiple times only the final box assignment will dictate the type of box that ends up in the cell. This decoding creates a two-dimensional structure on the game board which is then examined for fitness evaluation. Chromosome length is variable from individual to individual but is constrained to have a minimum length of one and a maximum length of 400.

6.1.3 Crossover Operator

The crossover operator employed is a simple uniform crossover for variable length chromosomes. Each row-column-box assignment in the chromosome has a 50% chance of coming from parent one and a 50% chance of coming from parent two. In the case of unequal parental chromosome lengths, row-column-box assignments in the longer parent that have no counterparts in the shorter parent are appended to the offspring's chromosome 50% of the time, at random.

6.1.4 Mutation Operators

When offspring are produced by mutation, one of three mutation operators is chosen with uniform probability. The first is a point-mutation operator which chooses an index and a strand from the chromosome uniformly at random and replaces it with a different value from the range of legal values for the strand. The second is a deletion mutation operator which chooses a chromosome index at random and deletes the gene from all three strands thereby shortening chromosome length by one. The third is an insertion mutation operator which chooses a chromosome insertion point at random and inserts a new value into each of the three strands thereby increasing the chromosome length by one. If a mutation operator would result in a violation of the chromosome length constraints, one of the remaining two operators is chosen uniformly at random instead.

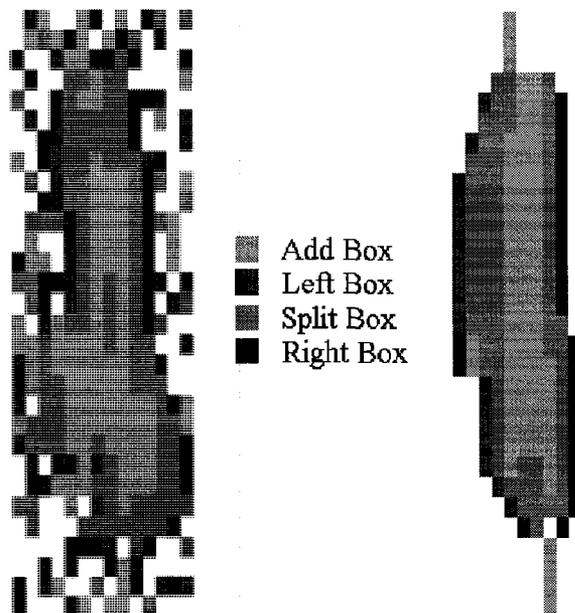


Figure 6.1 Examples of evolved phenotypes in the BG niche under both versions of its fitness function.

6.1.5 The BG Niche Fitness Function

Fitness in the BG niche is calculated by interpreting the two-dimensional phenotype on the game board in the following manner. A game ball falls through the board, entering from “above” (into row 0 as if from outside the board) at column index 5. This column will be referred to as the source column. The ball carries with it a point value which is initially set to 15 points. As the ball falls downward through the game board from row 0 to row 29, it is manipulated by any boxes it encounters. The four different types of boxes can modify the number of points assigned to the game ball, its direction of travel through the board, and even the number of balls in play. The manner in which each type of box affects game balls that pass through it is explained below. Any game balls emerging from column 8 at the bottom of the game board contribute to a total tally of points collected during the game. This column will be referred to as the sink column. The goal of the game, then, is to steer and multiply the game ball(s) on the game board in such a way as to funnel as many high-point-value balls out of the bottom of the source column as possible. This point tally contributes to the BG niche fitness value according to the fitness functions

$$F_1 = \max\{0, p - (cP)\} \quad (6.1.5-1)$$

and

$$F_2 = \max\{0, p\} \quad (6.1.5-2)$$

where F_1 and F_2 are different versions of the fitness function, p is the number of points collected, c is the chromosome length, and P is a penalty value which increases during a run in order to keep the game difficult as the population improves its performance. The value of P in the niche begins at zero and is increased at the end of every generation by

the minimum integer amount required to keep the population's maximum fitness value at or below a fixed threshold of 1000. Examples of highly evolved individuals in this niche, for each of these versions of the fitness function, are shown in Figure 6.1.

6.1.5.1 The Effect of Each Box Type on the Game Ball

The four types of boxes specified by the third strand of an individual's chromosome, designated by the letters S, A, L and R, affect the game ball in the following ways. The Split Box (or S-Box) splits the game ball into two balls, each with a point value one less than the original. The two reduced-point-value balls exit the Split Box in different directions. The second type of box, called the Add Box (or A-Box) allows the game ball to pass through and exit traveling in the same direction in which it entered. At the same time the point value assigned to the ball is increased by one point. The third and fourth box types, called Left Boxes and Right Boxes (or L-Boxes and R-Boxes) modify only the ball's direction of travel. The Left Box will cause the ball to make a left turn, as seen from the ball's point of view. The Right Box will cause the ball to make a right turn. In the cases in which such turns would cause the ball to exit the boxes traveling upwards, the ball exits traveling downward instead. The upward direction is forbidden in order to ensure that the time required for game play is finite. When a game ball enters a blank cell on the board, it always exits traveling downward. The rules for the S-, A-, L-, and R-Boxes are depicted in Figure 6.2. Incoming arrows indicate the direction from which a ball enters a box. Outgoing arrows show where balls exit the boxes. In the case of the S-Box, one ball enters but two exit. The arrow labels show any changes in the point values

(p) associated with the balls as they pass through the box. Balls that would exit the board at its extreme left and right edges always exit downward instead.

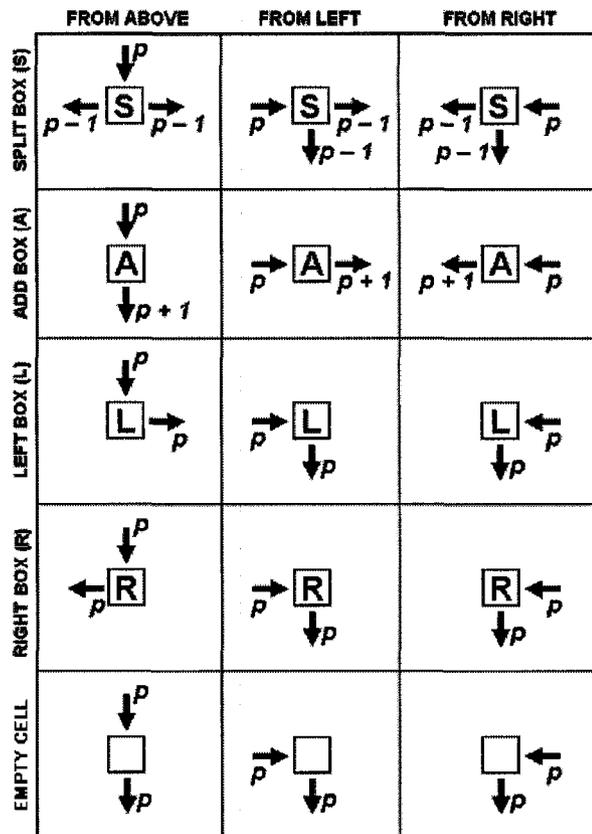


Figure 6.2 An illustration of the rules governing each box type in the BG niche.

6.1.5.2 Population Initialization

Since the population sizes for niches in the exaptive GA are quite small at 50 individuals per niche, population initialization has been biased in two ways in order to start each run off with a better sample of random individuals. Early in development of the GA it was found that a straightforward and unrestricted random population initialization technique often produced dead-end populations containing no viable individuals. The modifications to population initialization described below provide the benefits that would normally accrue from a much larger initial population size while avoiding the associated

computational costs. The modifications are tailored neither to enhance nor to hinder any of the exaptation-related phenomena under study in the GA, but instead are implemented to avoid seeding the GA with inviable individuals.

The first bias is a row and column restriction on the placement of boxes for randomly generated individuals. Randomly generated individuals' chromosomes restrict box placement to rows 2 through 15, inclusive, and to columns 3 through 10, inclusive. This boosts the chances of random individuals collecting points in the ball game by a) placing the boxes where they are more likely to intersect the path of the game ball and b) where they are more likely to be placed next to one another, increasing the likelihood neighbour interactions. Once a GA run begins, mutation can immediately violate this restriction. Since without these restrictions it was found that many randomly-generated individuals are inviable, and since inviable individuals are prevented from reproducing, this bias during initialization only ensures that nearly all runs will have at least one viable individual in the first generation (in the BG niche) instead of terminating immediately for lack of individuals from which to produce successive generations.

The second bias in population initialization is that every individual in the first generation is the best of ten randomly generated individuals. This was done for the same reason as was mentioned above, and applies to all niches whether they use the BG fitness function or not.

6.2 The Single Niche Exaptive GA

The single niche exaptive GA implements a gradual functional change within a single population. In [Graham, Oppacher, and Christensen, 2007] it was used to demonstrate that Darwinian evolution can produce systems composed of multiple parts contributing to a specified function such that all of its components are critical. Such systems have been dubbed “irreducibly complex” [Behe, 2006], with the implication that such systems cannot be produced by evolution. In fact, such systems were predicted by biologists to arise by evolution as early as 1918, and were dubbed “interlocking complexity” by [Muller, 1939]. While there is ample evidence that evolution can and does evolve such systems, let us adopt the term nonetheless as it describes the current state, and not the origin, of a system. The single niche exaptive GA employs a changing fitness function that increases parsimony pressure as the population evolves in order to produce systems exhibiting this type of complexity. This is done by using the fitness function from formula (6.1.5-1), above, which includes a penalty proportional to chromosome length.

6.2.1 Interlocking, or “Irreducible”, Complexity

The modern creationists’ concept of irreducible complexity is well known, especially in North America². The pseudoscientific baggage attached to this concept has been more than adequately dealt with in other places. A good example is the TalkOrigins archive [TalkOrigins, 2007] which, though not peer-reviewed, contains abundant information

² The concept is sound, but its modern incarnation is notorious for claims that such systems cannot evolve and are thus the products of intelligent agency. Such claims are fallacious; they are straw man arguments in their implicit impoverished portrayal of the available mechanisms of evolutionary change, arguments from incredulity, and are built upon a false evolution-or-design dichotomy. This otherwise legitimate concept is thus often tarnished by its abuse in the service of political and religious campaigns.

geared toward the layman as well as primary literature references countering a wide variety of pseudoscientific evolution-denial claims. Such topics have also been dealt with in books [Dawkins, 1986], [Miller, 1999], in courts of law [Jones III, 2005] as well as in numerous other venues. Recently [Adami, 2006] discussed research into the evolution of two hormone receptors from a common ancestor, highlighting how a plausible and reasonable evolutionary pathway was discovered that led to what one might call an “irreducibly” complex system. The pseudoscientific claims surrounding this concept will not be addressed here any further than to point the interested reader to the above references.

From an evolutionary perspective, systems possessing “irreducible complexity” (henceforth, “interlocking complexity” following [Muller, 1939]) pose an interesting challenge since they often involve non-stationary fitness that differs from the usual finely graded progression from random beginnings to a local or global optimum in a static fitness landscape.

6.2.2 The Functional Change Mechanism

The TalkOrigins Archive [Isaak, 2005] lists several evolutionary mechanisms that can contribute to the production of interlocking complexity. These include the deletion of parts, addition of multiple parts, change of function, addition of a second function to a part, gradual modification of parts, and loss of previously existing scaffolding. Although the archive deals primarily with biological evolution, the carry-over to EC is often meaningful.

Other researchers in EC have described the digital evolution of interlocking complexity in an Artificial Life context [Lenski et al., 2003] via mechanisms different from those described here. The GA detailed in the following section produces such systems with the use of a parsimony pressure penalty value which amounts to a mechanism of gradual functional change. This constitutes an EC example of one of the mechanisms listed in [Isaak, 2005], namely the change of function mechanism.

6.2.3 The Complexity Check

Each individual in a GA run is checked to determine if it possesses interlocking complexity, as described above. In accordance with the fitness function, we apply the current magnitude of chromosome length penalty, P (see formula (6.1.5-1)), being sustained by the population to determine viability. An individual will be counted as possessing interlocking complexity if the following three conditions are met.

- The individual must be performing the function determined for it – attaining non-zero fitness.
- The number of parts (boxes) in the phenotype must be five or more. This eliminates the very simplest and least complex game solutions from consideration.
- The removal of any single box from the phenotype must result in fitness dropping to zero.

The last condition is checked by evaluating the fitness of each knock-out mutant in sequence, terminating once all have been tried, or once a viable knock-out is found.

In order to observe the behavior of the GA with regard to the evolution of these complex systems, the program records the number of viable individuals in the population at each generation, and the number of individuals possessing interlocking complexity as well.

The latter are also viable by definition.

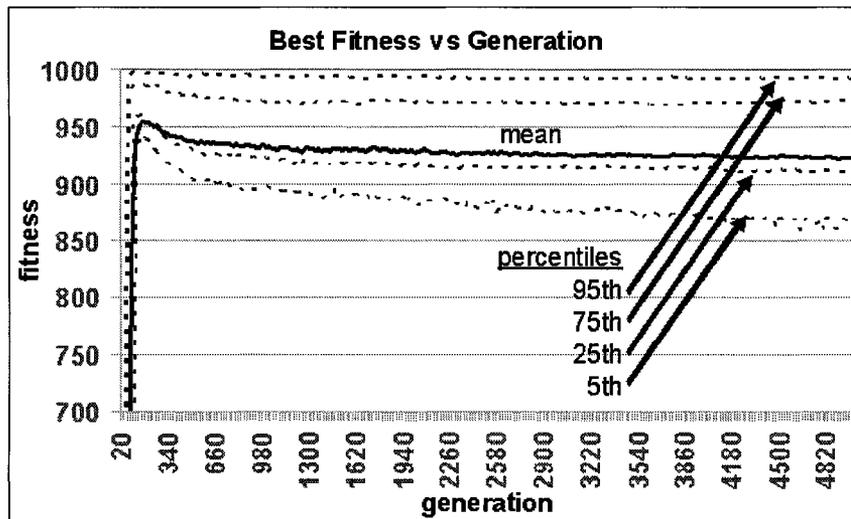


Figure 6.3 Mean and four quantiles of best fitness vs. generation in the single niche exaptive GA.

6.2.4 Results

Data were collected from 600 independent runs of the single niche exaptive GA. Each run was tracked for 5,000 generations. A number of population and fitness statistics were tallied and logged every 20th generation. These are plotted in Figure 6.3, Figure 6.4, Figure 6.5, Figure 6.6, and Figure 6.7.

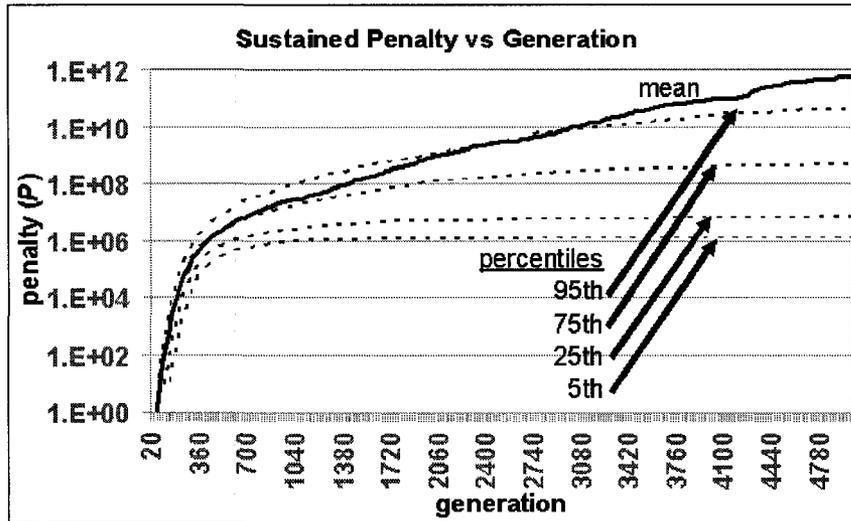


Figure 6.4 Mean and four quantiles of chromosome penalty P vs. generation in the single niche exaptive GA.

Due to the manner in which the fitness function changes over time, the raw fitness values appear to rapidly converge and level off. This is shown in Figure 6.3. Beyond the point of fitness convergence, the population has not ceased to evolve, nor does it undergo simple genetic drift. This can be seen in the graph of P , the chromosome length penalty that governs game difficulty, shown in Figure 6.4. P continues to rise beyond the point of fitness convergence, through the 5,000th generation at least. The population is remaining proficient at playing the game while the game itself becomes ever more difficult.

Figure 6.5 shows the number of individuals per run that satisfy the complexity test at each generation. The graph shows that around the 200th or 300th generation the GA begins to produce such individuals, but not many. On average, there are about four such individuals in a population of at most 50 individuals. The actual number of viable individuals drops to a small number and remains there, as shown in Figure 6.6. Figure 6.7 combines the results of Figure 6.5 and Figure 6.6 to show the percentage of viable

individuals that exhibit interlocking complex as a function of generation number. Figure 6.5, Figure 6.6, and Figure 6.7 show that these individuals quickly come to dominate the population as opposed to being a mere minority of individuals that appear from time to time, unable to compete with the rest. In fact, they constitute the bulk of the GA's evolved game solutions.

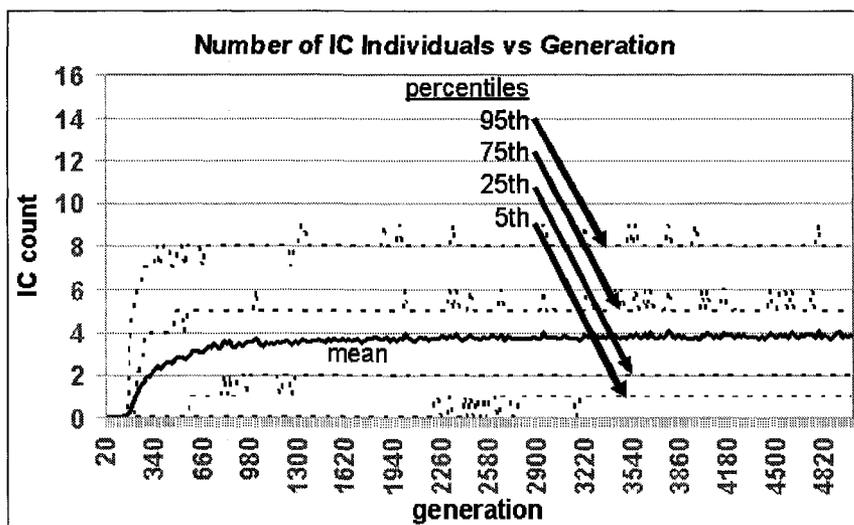


Figure 6.5 Mean and four quantiles of the number of individuals possessing interlocking complexity vs. generation in the single niche exaptive GA.

The game solution strategies hit upon by the GA are almost always variations on the general pattern shown on the right side of Figure 6.1. Leading and trailing segments of A-boxes connect the top of the source column and the bottom of the sink column to the cluster of boxes at the center of the board. That cluster consists of a mix of S- and A-box regions that together are braced on either side by a layer of L- and R-boxes to feed game balls back into the cluster. This general strategy seems very strong given the magnitude of the chromosome length penalty being sustained.

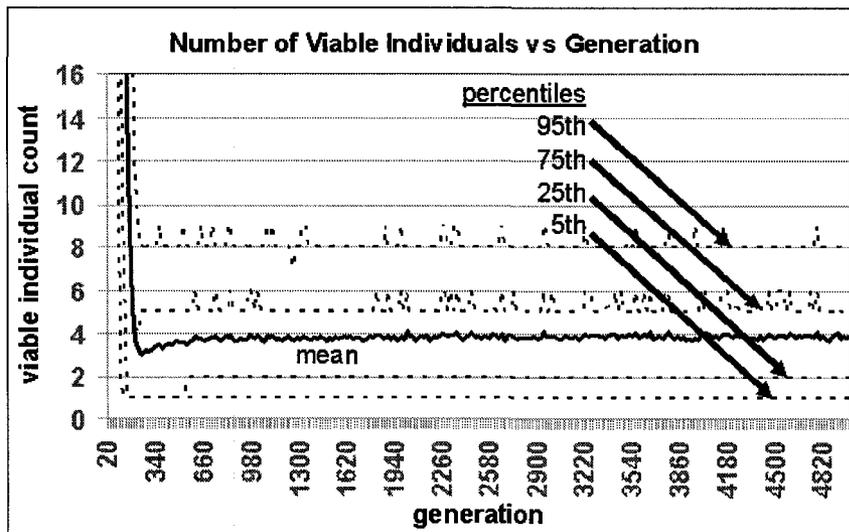


Figure 6.6 Mean and four quantiles of the number of viable individuals vs. generation in the single niche exaptive GA.

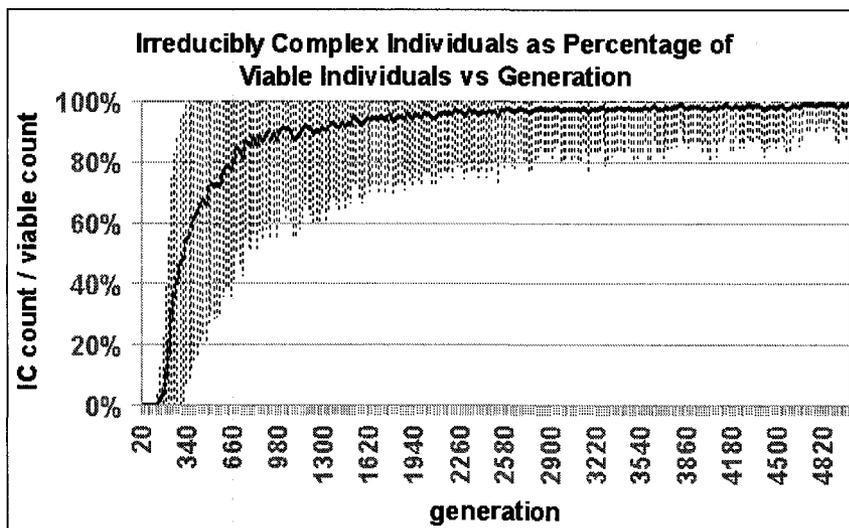


Figure 6.7 Mean and standard deviation of individuals exhibiting interlocking complexity as a percentage of total population vs. generation in the single niche exaptive GA.

6.2.5 Conclusions

The evolution of such complex systems presents an interesting problem, but one which is demonstrably not insurmountable. There is no shortage of known evolutionary mechanisms that can produce them, such as those given in [Isaak, 2005]. The single niche

exaptive GA generates such systems in a simulated medium as opposed to a biological one. It does so ultimately through a change in the system's effective function, as the environment (the ball game) becomes harsher over time. The change of function in this example is not sudden and drastic, as might be expected in cases involving exaptation as it is most commonly understood [Gould & Vrba, 1982], but is instead a gradual change in difficulty.

What the single-niche exaptive GA may tell us about the use of exaptation in GAs is that a process of gradual functional shift is one possible approach for the evolution of interlocking complexity, but also for some systems involving a large threshold-like minimum requirement (like that imposed by the chromosome length penalty). It may be an effective strategy to evolve such systems via gradual adjustment of the threshold as the population adapts.

By their nature, such systems are quite brittle and sensitive to most mutations. The result of this can be seen clearly in the low viable individual counts shown in Figure 6.6. Had there been a large number of duplicate individuals in the population, Figure 6.6 might have looked quite different, but the manner in which offspring are produced in this particular GA severely hinders but does not strictly prohibit duplicates from arising.

In an unchanging environment, one might expect systems that exhibit interlocking complexity to remain stable due to their fragility under variation operators. An interesting conclusion that might be drawn from this simple example is that such systems need not remain stable, even in dynamic environments, despite their fragility under variation operators. In the single niche exaptive GA, the dynamic environment takes the form of a

gradually changing fitness function. A biological analogy might be the hypothetical evolution of ingredients in a predator's venom against the co-evolving immune system of its prey.

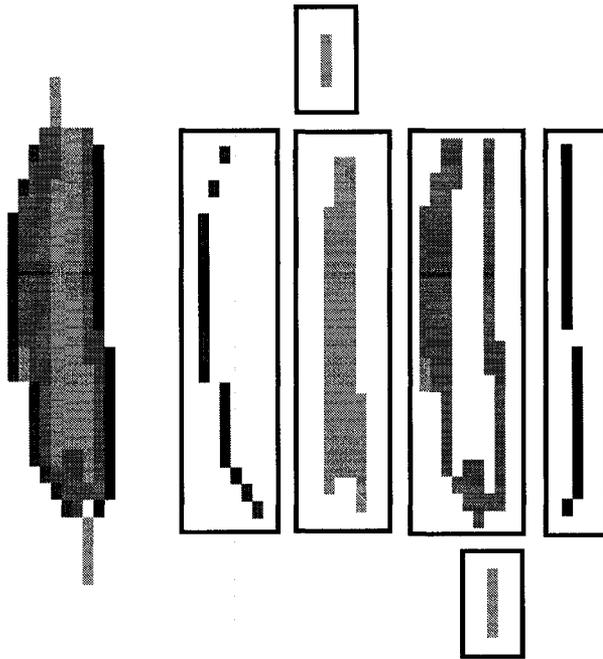


Figure 6.8 An alternative division into parts of the phenotype from the right side of Figure 6.1.

There is an interesting problem that arises when considering interlocking complexity: the problem of defining one's terms. In particular, the definitions of *system*, *function*, and *part* are quite flexible and subject to liberal semantic manipulation. By concretely operationalizing these terms in the single niche exaptive GA, this issue is at least partly avoided. Here, a *system* is defined to be the phenotype of an individual, its *function* to be the attainment of non-zero fitness, and its *parts* to be game boxes in their locations on the game board. Under these unambiguous definitions, the GA presented here produces systems of interlocking complexity consisting of upwards of 100 parts. There are, however, alternative definitions of *part* that might be used. For example, the phenotype from the right side of Figure 6.1 is shown again in Figure 6.8, disassembled into six

constituent parts using what is perhaps a more intuitive notion of *parts*. Under this scheme as well, the systems produced by this GA exhibit interlocking complexity.

6.3 The Two Niche Exaptive GA

The two niche exaptive GA [Graham & Oppacher, 2007a] adds a second niche to the exaptive GA and uses exaptation to facilitate speciation. A variation on this GA was also used to demonstrate both speciation driven by selection and speciation driven by genetic drift [Graham & Oppacher, 2007b]. Both versions relied on a form of exaptation, though only the first version is described here. This section examines the data collected from thousands of runs of the GA and provides a brief analysis. These data show that once a viable population of individuals begins to evolve in the second niche the two populations rapidly diverge to the point of effective speciation – hybrid offspring produced by the crossover operator are inviable under the fitness functions of both niches.

6.3.1 Speciation Facilitated by Exaptation

The fitness function of the second niche is deliberately set up in a way that renders random population initialization unable to produce viable individuals. Structures with the ability to satisfy the minimum requirements of the second niche must be evolved in the first niche and then be exapted to use in the second. This exaptation takes place when a migrant individual from the BG niche attempts to jump to the second niche. If it's found to be viable, the migrant seeds a new population in the second niche. The two niche populations then diverge genetically to such a degree that offspring produced by the crossover operator are inviable in both niches. This simple adaptive evolutionary

behavior in the genetic algorithm can be likened to the concept of speciation in biology. In this case the speciation is initiated by an exaptation event, and driven on by the following three factors:

- Continued adaptation and structural changes in the first niche population beyond the time of the first successful migration
- Secondary adaptations in the second niche, once seeded, which refine its population's exapted structure to better suit the new fitness function
- A genetic representation that is particularly flexible in terms of chromosome reorganization

The first two factors, simply put, mean that the two populations follow different evolutionary trajectories, because of different selection pressures, despite starting at the same point. The third simply increases the ease and speed with which this happens.

This exaptation and speciation model GA, though it does not utilize a real-world problem, is simple enough to be easily understood and provides a proof-of-concept example of how the deliberate use of functional change and exaptation can facilitate a process of speciation in evolutionary computing. In this model the type of speciation involved is somewhat more in line with biological definitions of species and speciation, due to its distinction between viable and inviable individuals.

The importance of facilitating speciation in EC is stressed by [Banzhaf et al., 2006] when they discuss a number of facets and directions for future research agendas that could transform current EC techniques (which they broadly call Artificial Evolution, or AE)

into a richer, more diverse, and more powerful field they call Computational Evolution, or CE.

“CE allows reproductive barriers to arise and can change their permeability without constraint, as the barriers themselves could evolve, allowing speciation to emerge as a property of the system. This contrasts with ‘speciation’ in AE, which exists only to increase population diversity.”

The niche population divergence in the two niche exaptive GA is closer to this kind of speciation. It’s a form of reproduction barrier, and not a means of boosting diversity within a population focused on a single fitness problem, as is usually the case for speciation techniques currently used in EC (or AE). It also demonstrates the role that exaptation can play in facilitating such speciation events.

The concept of a species and the processes of speciation are central ideas in the field of evolutionary biology, and are vital to our understanding of the abundant diversity of life forms on Earth. In the biological sciences speciation is said to occur when two populations of organisms, having originated from a common ancestral stock, diverge from one another either genetically or behaviorally to such a degree that members of one population either cannot or do not interbreed with members of the other population.³

Once speciation has occurred the two populations are said to be reproductively isolated, and continue to diverge from one another via genetic drift, selection, or both, increasing the diversity of life. The process of speciation is thought to be triggered either by the geographical isolation of a group of organisms from the original stock, in which case the phenomenon is called allopatric speciation, or by behavioral changes that prevent interbreeding despite both lineages living in the same location. This second alternative is

³ This is just one of several definitions of the term *species* used in biology.

referred to as sympatric speciation. Numerous instances of speciation have been observed both in the lab and in the wild, and are documented extensively in the biological literature [Boxhorn, 1994].

Just as the biological phenomenon of speciation is a source of diversity and innovation in the biosphere, it may be that a comparable notion of speciation, which is mostly lacking in extant EC systems (where individual species are merely subpopulations exploring different regions of a common fitness landscape), will yield benefits. As one population speciates, the system applies previously learned or evolved knowledge and structures to newer and possibly more difficult problems, or at least problems not capable of being tackled by earlier populations. An ability to harness behavior of this sort could conceivably prove to be a useful addition to the EC design and optimization toolkit.

6.3.2 The BG Niche Fitness Function

Like the single niche exaptive GA, the two niche version also uses the version of the BG niche fitness function that penalizes for chromosome length (see formula (6.1.5-1)). All parameters such as mutation and crossover rates, population and tournament sizes, etc, are also the same.

6.3.3 The LEA Niche Fitness Function

The fitness function in the second niche is called LEA for Largest Enclosed Area. It makes no distinction between the four different types of boxes placed on the game board. To attain nonzero fitness in the LEA niche, the two-dimensional structure created by an individual's phenotype must create an unbroken path of boxes from the top of the board

(row 0) to the bottom (row 29). Such a path can be established between successive neighbors in the Moore Neighborhood. The Moore neighborhood includes all of the eight cells surrounding a square on the game board's grid. Provided that such a path exists, fitness is equal to the largest contiguous region of blank cells which is completely enclosed by a circuit of boxes. Like the top-to-bottom path, this circuit follows the Moore neighborhood at each step. If an individual's phenotype does not possess a path connecting the top and bottom of the board, or does not possess a circuit enclosing an empty region consisting of at least two blank cells, the individual is assigned zero fitness and is considered inviable. An example of a highly evolved individual from the final generation of a run is shown on the left side of Figure 6.11.

The LEA fitness function operates in such a way that randomly generated individuals in the first generation are inviable with extremely high probability. The LEA niche only becomes populated when exaptation occurs in the GA and structures evolved in the BG niche become viable in the new niche. All randomly generated individuals, whether in the BG niche or the LEA niche, place their boxes only on a restricted region of the board as explained in the population initialization section, above. This prevents a top-to-bottom path from ever appearing in the first generation and, on its own, can render all first-generation individuals inviable in the LEA niche. Nevertheless, even without this restriction, viable individuals are almost never produced at random for the second niche. To test the claim that this restriction does not materially change this property of the niche with regards to the inviability of random individuals, 8,000,000,000 random individuals were generated without the box placement restriction and not one was found to be viable in the LEA niche.

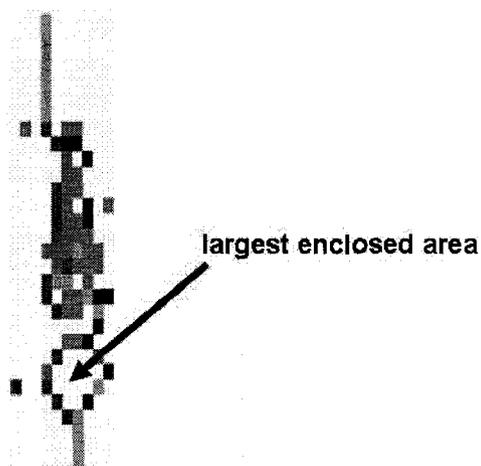


Figure 6.9 An example phenotype from the LEA niche, just a few generations after exaptation, showing a fit BG niche phenotype in the processing of being converted to a fit LEA niche phenotype.

6.3.4 Migration

Since the LEA niche remains devoid of viable individuals, and therefore effectively empty, after population initialization, some means of allowing the niche to become populated at a later time is required. This is accomplished by choosing a random viable individual from the BG niche at the end of every generation and evaluating that individual's fitness in the LEA niche. If the individual, called a pioneer, is found to be not only viable in the new niche, but also fitter than the least fit viable individual already present in the niche, then a migration event takes place, and a copy of the pioneer is placed in the new niche, replacing the least fit individual already present. In this manner, structures evolved in the BG niche can be exapted to allow individuals to survive in a new niche with new selection pressures. Just as in the case of the snail shell umbilicus (see section 3.5) being exapted for egg brooding, particular side-effect aspects of the structures evolved in the BG niche are seized upon and adapted to a new function. In this case, as with the snail umbilicus, the structure is, in fact, a gap that develops in the "true" structure adapted in the BG niche for reasons unrelated to the LEA niche fitness function.

Figure 6.9 shows just such a gap in the early stages of its accumulation of secondary adaptations, shortly after exaptation.

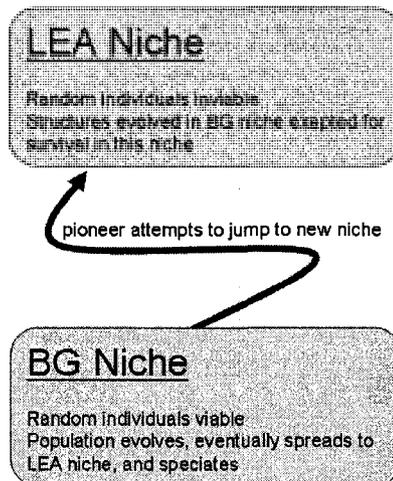


Figure 6.10 Niches and migration route in the two niche exaptive GA.

After the exaptation event(s), the newly-founded population in the second niche can undergo further adaptation in an entirely new direction. The two populations then diverge genetically to the point where hybrid offspring, produced by the crossover of a parent from each niche, are inviable in both niches. When this occurs, the initial population, which started in the BG niche, has speciated into two populations which are unable to interbreed. This GA setup is depicted in Figure 6.10.

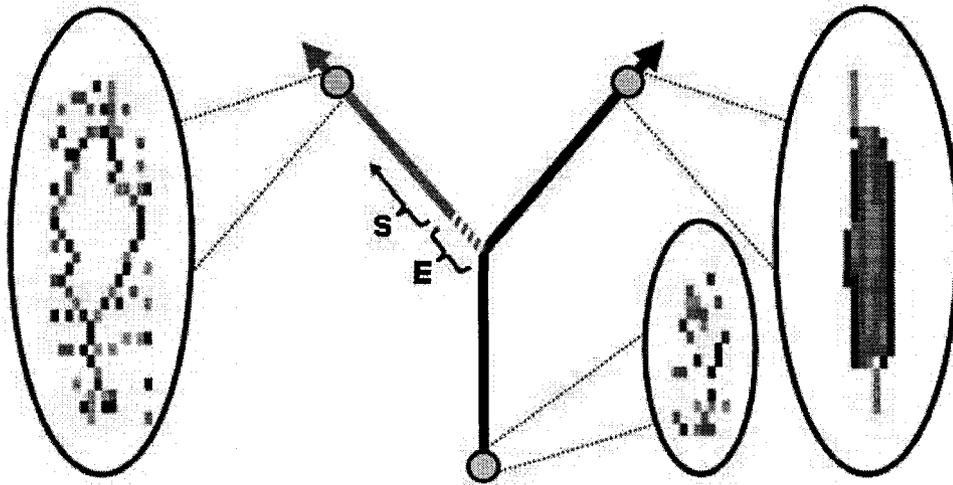


Figure 6.11 A cladogram depicting evolution, exaptation, and secondary adaptation in the two niche exaptive GA.

Figure 6.11 shows a cladogram depicting the evolution of the populations in the two niche exaptive GA. An initial population of random individuals starts the cladogram at bottom. It evolves in the BG until a structure appears which can be exapted for use by a migrant in the LEA niche. This is the point of exaptation, marked by the letter E in the figure. Following exaptation, the population seeded by the migrant(s) accumulates secondary adaptations. This part of the cladogram is marked by the letter S in the figure. Meanwhile, the original population continues to adapt to the BG niche fitness function.

6.3.5 Results

7500 runs of the two niche exaptive GA were performed with the parameters and configuration described in the previous sections. Each run lasted for a duration of 5000 generations. At the end of every twentieth generation the following statistics were logged.

- The fitness of the best individual in each niche
- The number of viable individuals in the LEA niche (between 0 and 50 individuals)
- The number of successful pioneer jumps during the 20 generation interval (between 0 and 20)
- The outcome of a crossover operator hybridization test between the best individual from the first niche and the best individual from the second niche. This test has three possible outcomes, described below.

In the case of the hybridization tests, the first possible outcome occurs when at least one of the two niches contains no viable individuals to take part in the crossover and no hybrid offspring can be generated. The second possible outcome occurs when the hybrid offspring is viable (has nonzero fitness) in at least one of the two niches. The third possible outcome occurs when the hybrid is inviable in both niches.

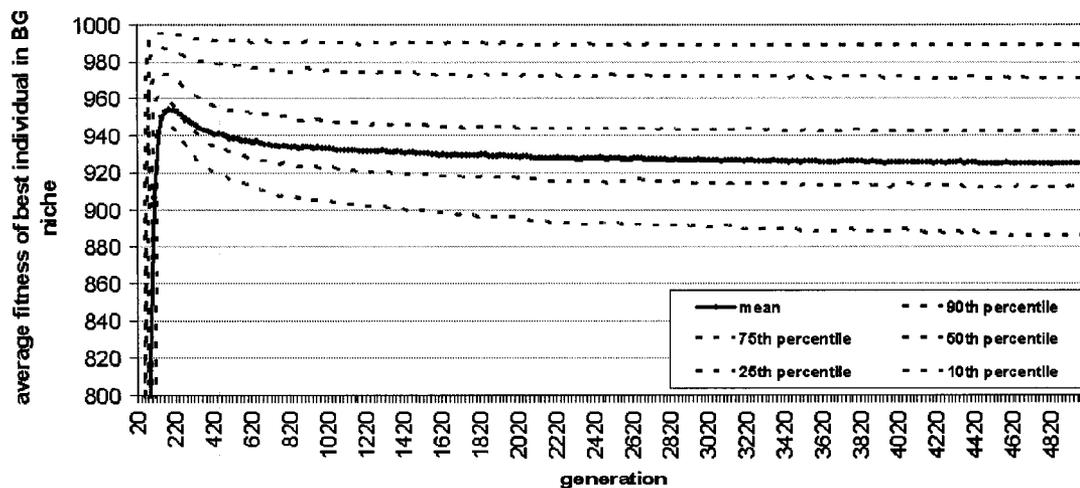


Figure 6.12 Mean and five percentiles of best fitness vs. generation in the two niche exaptive GA's BG niche.

Figure 6.12 shows the change over time in the average fitness value of the best individual in the BG niche during a run. The fitness threshold and chromosome length penalty create a fitness graph with negative slope in some regions, which is unusual for a GA, especially one employing elitism. The graph is somewhat deceptive, however, since the number of points collected by individuals in the board game continues to increase exponentially throughout the run. This is shown by the chromosome length penalty graph in Figure 6.13. This is analogous to van Valen's Red Queen Effect [van Valen, 1973] in biology, with the adjustments in chromosome length penalty playing the role of a second party in an arms race.

Figure 6.14 and Figure 6.15 show, for the LEA niche, the number of viable individuals and the average best fitness, respectively. Both of these graphs attest to the fact that the niche population is an offshoot of the BG niche population. The initial period, hundreds of generations long on average, devoid of viable individuals at the beginning of the runs indicates that the final population in the niche cannot have evolved from the initial random stock since inviable individuals cannot reproduce. This is also guaranteed by population initialization. It is evident in the graph despite the fact that Figure 6.14 averages over multiple runs in which pioneer jumps take place at different times or in some cases not at all. This has the effect of exaggerating variance during the period in which migration takes place, and yet there is still a period of calm in the graph which is common to all the runs regardless of when the first pioneer migrates. It is during this initial period of calm that a co-optable structure is being evolved in the BG niche, which can then be used to seed the LEA niche.

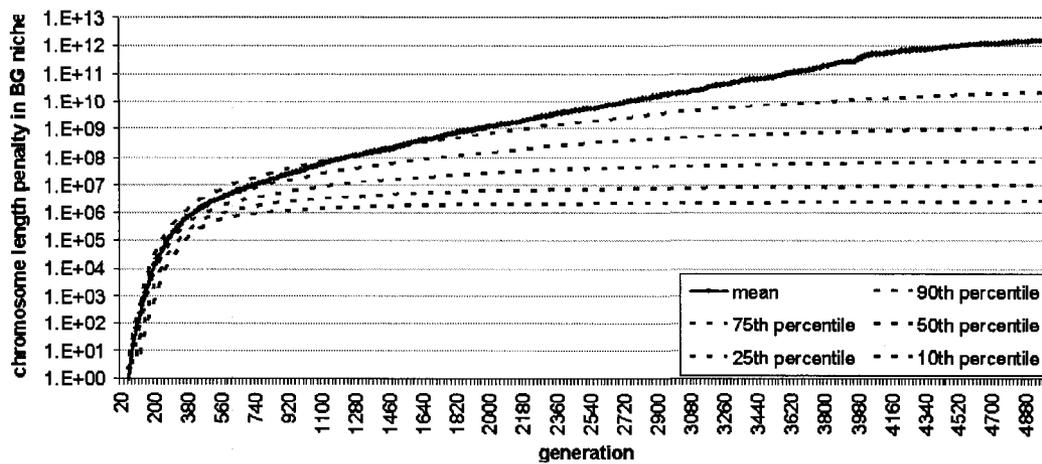


Figure 6.13 Mean and five percentiles of chromosome length penalty P vs. generation in the two niche exaptive GA's BG niche.

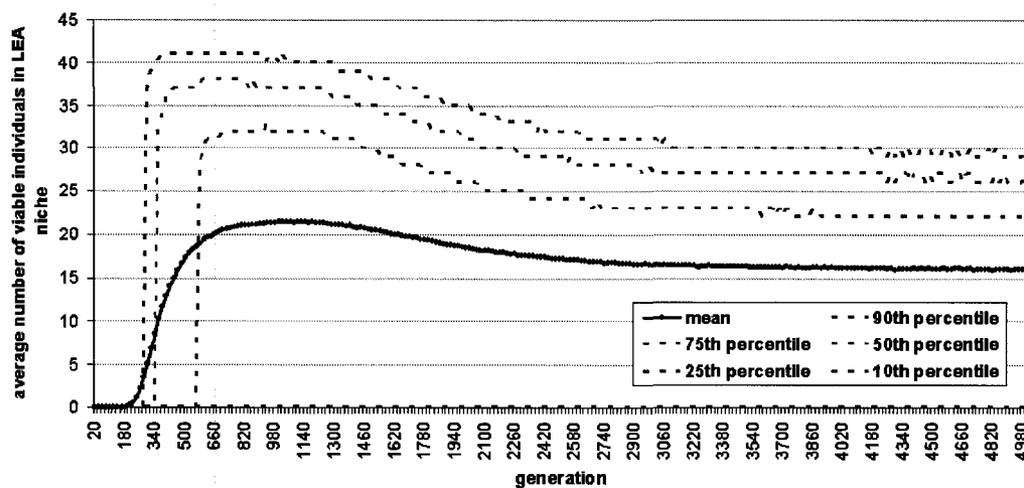


Figure 6.14 Mean and five percentiles of the number of viable individuals vs. generation in the two niche exaptive GA's LEA niche.

Figure 6.16 shows the behavior of the GA with respect to migration from the first niche to the second. It can be seen in the graph that migrations are made during the first half of the run, after the initial period of calm in which a co-optable structure is being evolved. It can also be seen that the chance of another successful migration drops rapidly to a value near zero. The histogram inset in Figure 6.16 shows that, of those runs in which migrations occurred at all, often only a single individual seeded the new population, with

subsequent potential migrants having been unable to compete against the new population's rapid adaptation to the demands of the new fitness function.

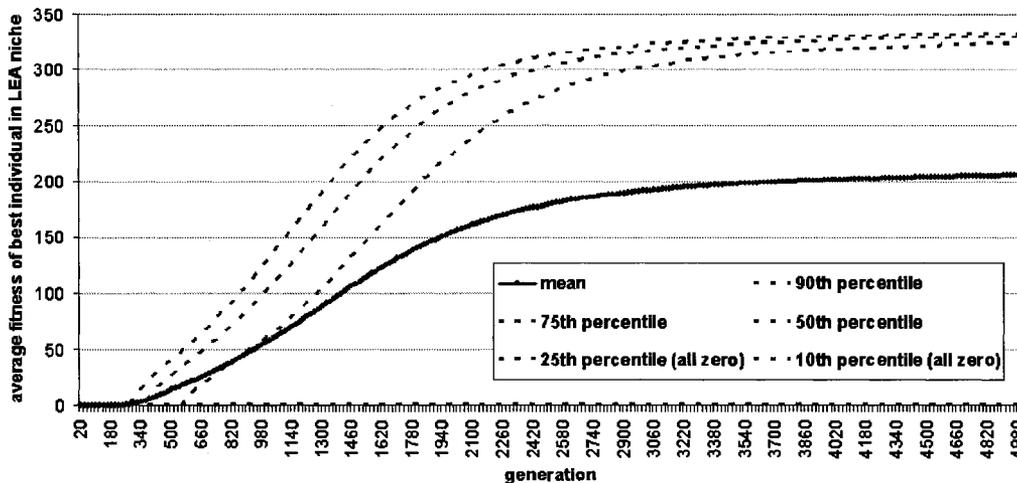


Figure 6.15 Mean and five percentiles of best fitness vs. generation in the two niche exaptive GA's LEA niche.

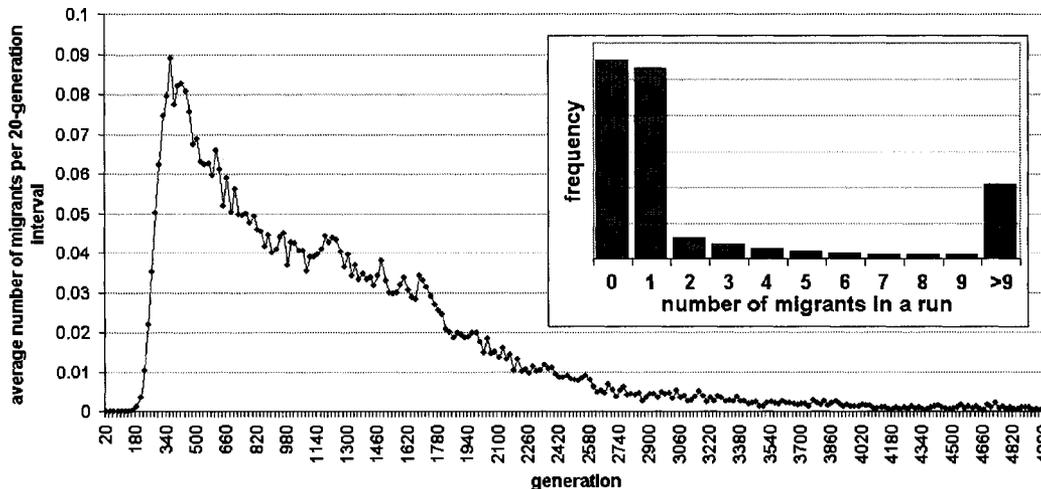


Figure 6.16 Mean number of migrants per logging interval vs. generation, and migrants per run frequencies in the two niche exaptive GA.

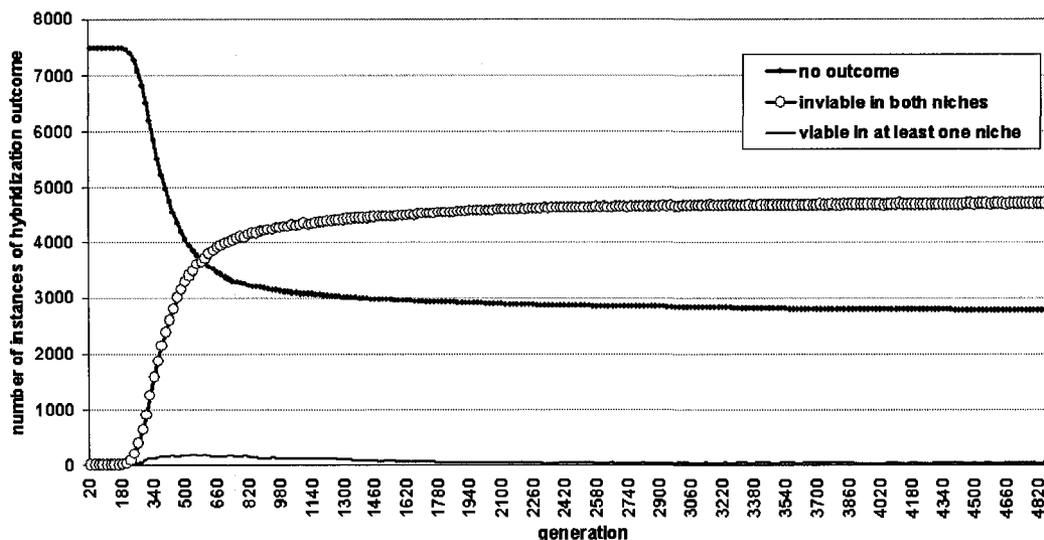


Figure 6.17 Hybridization test outcomes in the two niche exaptive GA.

Finally, Figure 6.17 shows the results of the hybridization tests. The graph shows unambiguously that the offshoot population in the second niche becomes unable to interbreed with the original population in the first niche. This divergence is so rapid that at the twenty-generation logging interval resolution of the graph, the hybridization outcomes in which the offspring are viable in at least one niche amount to only a small bump at the bottom of the graph.

6.3.6 Visualizing Exaptation and Secondary Adaptation in Fitness Space

One way to depict the evolutionary trajectories of the populations during their speciation is with cladograms like the one in Figure 6.11. Such diagrams show only that the two final populations were originally one, but give no indications about changes in fitness values during the process of evolution.

A second way to visualize exaptation in the GA is to plot the trajectories of the populations in fitness space, where each orthogonal axis corresponds to a fitness function. Such trajectories are shown in Figure 6.18. These were produced by a modified version of the two niche exaptive GA. The modifications included using formula (6.1.5-2) in the BG niche so that fitness would not deceptively level off to mask the evolutionary progress still occurring, and the introduction of a third niche whose fitness function is the product of BG fitness and LEA fitness.

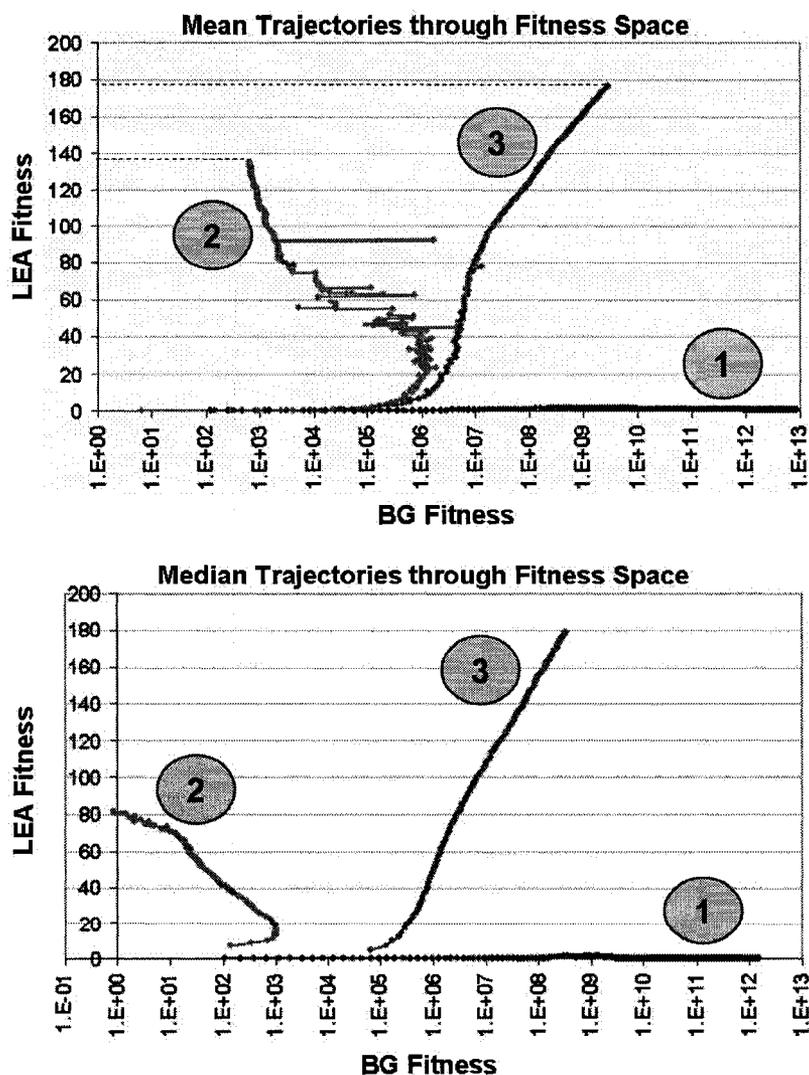


Figure 6.18 Mean and median population trajectories through fitness space in a variant of the two niche exaptive GA.

In order to generate the six trajectories shown in Figure 6.18, each of the three niches had its best individual at the end of each generation evaluated up to three times. The first evaluation used the actual fitness function that drove the evolution in the niche. The second used the BG fitness function, and the third used the LEA fitness function. The final two measurements were treated as coordinates in fitness space. Trajectories labeled “1” correspond to the mean and median (over several hundred runs) fitness values from the original source population in the BG niche. Trajectories labeled “2” are from the LEA niche. As secondary adaptations accumulate in that niche, performance at the BG fitness function, since it is no longer being selected for, begins to decline. Trajectories labeled “3” are from the third niche using the product of BG and LEA fitness value. This third niche was added as an example of exaptation that does not involve the loss of the original function.

An interesting result that can be seen in Figure 6.18 is that the extra niche that combined the new function with the old actually outperforms the regular LEA niche. This can be seen in the way the trajectories labeled “3” reach higher on the vertical axis than those labeled “2”. No particular explanation for this has been tested, but a plausible explanation may be that the changes provoked by the BG aspect of fitness allow the niche population to more easily overcome small fitness saddles in the LEA fitness landscape by forcing changes that would otherwise be detrimental. Alternatively, the BG aspect of fitness may be speeding up traversal of LEA fitness plateaus by providing a slope and turning what might regularly be a random walk into a more direct route past the plateau. Such benefits may be the exception, and not the rule. For example, one need look no further than the

same graphs, in which the third niche outperforms the regular LEA niche, but is itself outperformed by the regular BG niche.

Ultimately, in the long term, the trade-off between BG fitness and LEA fitness, in the niche that combines their fitness values, will mean that the regular LEA niche can eventually overtake it. However, since, in the short period represented by the trajectories shown in Figure 6.18, the combination LEA*BG niche outperforms the regular LEA niche, this scenario may be showing an example where an exaptation approach can outperform a single fitness function approach. If the GA were configured to evolve for N generations using the LEA*BG fitness function, followed by M generations with just the LEA fitness function, the performance may exceed that of an identical GA running $N+M$ generations with just the LEA fitness function due to the extra speed boost before the fitness function changes. This may be the case even without the harsh viability and initialization constraints imposed by the exaptive GA. This experiment was conducted, and is described in detail in section 6.5, below.

6.3.7 Conclusions

Whereas the term species in GAs has often come to refer to isolated subpopulations diverging from one another but working toward the same goal in roughly the same fitness landscape, such as the creation of a highly-fit solution to a combinatorial optimization problem, the term is very often divorced from its biological counterpart's focus on interbreeding (or lack of ability to do so) between the two populations.

Many researchers employing speciation in GAs do so in order to place greater emphasis on the exploration side of the exploration-vs-exploitation tradeoff inherent in such algorithms. The issue of barriers to interbreeding in a GA often arises from population isolation, and not from an inability to produce viable offspring. In many GAs any individual may cross with any other individual, although in some cases they may simply be forbidden from doing so. From this point of view, all individuals in all subpopulations belong to a single species.

For these reasons, and because many GAs make no distinction between viable and inviable offspring, the terms species and speciation have only a tenuous connection to their counterparts in biology. By introducing a notion of viability, and by allowing individuals with the same underlying genetic coding format to evolve under two distinct and very different fitness regimes, an original interbreeding population is allowed to split using an exaptation mechanism and diverge to the point where crossover can no longer produce viable offspring. In doing so, the GA is able to make progress in a new niche which would otherwise have remained filled with inviable solutions⁴. Although simple in design, and very rudimentary, the two niche exaptive GA reliably exhibits a speciation-like phenomenon, achieved through exaptation, which more closely adheres to the meaning of the term as it is used in the biological sciences.

⁴ Although this inviability of random individuals in the LEA niche is central to the speciation exhibited here, it is not true that this need necessarily be the case for this particular toy problem. Relaxation of restrictions on chromosome length and box placement during population initialization may yield viable random solutions at a workable rate in this instance. There may be some problems, however, where the random generation of viable or feasible solutions is a challenging problem in its own right, and for which some mechanism of exaptation may be an alternative avenue of exploration for the would-be EC practitioner.

6.4 The Four Niche Exaptive GA

The four niche version of the exaptive GA provides each of its four niches with a different fitness function. Just as before, the niches share a common genetic encoding and genotype-phenotype mapping, allowing for inter-niche migration of individuals. The same notion of viability is introduced whereby population initialization produces viable individuals in one niche (the BG niche) and is extremely unlikely to do so in all other niches. The niche fitness functions have been devised so as to demonstrate the gradual evolution of a population via multiple exaptation events where migrants seed a new niche at each step, adapt to it, and then spread to another in a predictable sequence. Such migration and exaptation events take advantage of what could be viewed as hidden relationships between fitness functions and allow evolving populations to explore regions of phenotype space that are otherwise inaccessible in a single fitness function GA without special population initialization mechanisms involving the non-evolutionary generation of partial solutions to the problem.

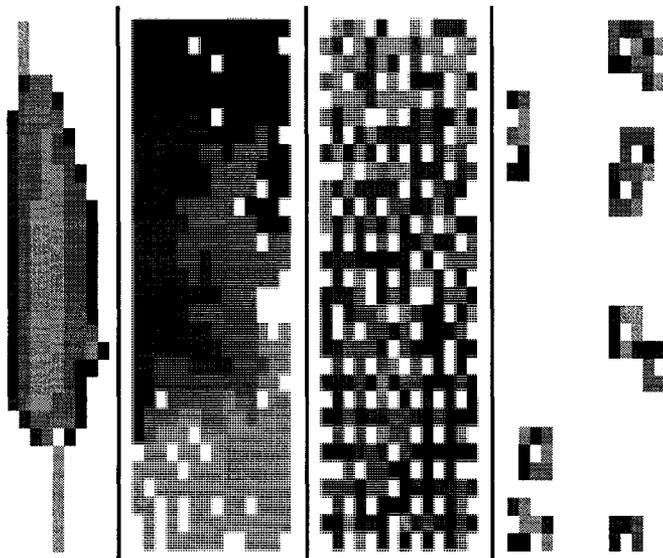


Figure 6.19 Examples of evolved phenotypes from each of the four niches (left to right: BG, PLCR, Sieve, and CRD) in the four niche exaptive GA.

6.4.1 The BG Niche Fitness Function

Like the previous versions of the exaptive GA, the four niche version also uses the BG niche fitness function that penalizes for chromosome length (see formula (6.1.5-1)). All parameters such as mutation and crossover rates, population and tournament sizes, etc, are also the same and apply to all four niches.

6.4.2 The PLCR Niche Fitness Function

The Product of Largest Coloured Regions niche interprets the phenotype as a set of contiguous regions of different box types on the grid. In this case contiguous means that all boxes in a given region can be reached from all others in the same region by a finite sequence of steps from one box on the grid to a neighboring box of the same type. Two boxes are considered neighbors if they are directly adjacent and share the same row or column. All boxes mutually reachable by such a path are considered part of the same region. These definitions partition the phenotype into a set of regions, each with a specific size and colour. The terms colour and box type are used here interchangeably since phenotypes are depicted graphically by using color as an indicator of box type, as in Figure 6.9, for example.

The fitness function for the PLCR niche first determines whether or not the phenotype extends from the top of the grid to the bottom. A path must be found from a region containing a box in the top row, through adjacent regions, to a region containing a box in the bottom row. Two regions are considered adjacent if a box from one is with one row and within one column of a box from the other. Phenotypes that do not exhibit this top-bottom-connectivity are considered inviable and are assigned zero fitness. For those

individuals passing the connectivity test, fitness is simply the product of the number of boxes in the largest region of each of the four types. That formula is

$$fitness(i) = \prod_{j \in \{S,A,L,R\}} \max_region_area(j) \quad (6.3)$$

where *fitness(i)* is the fitness of individual *i* in the population, and *j* iterates through the four box types when calculating the product.

The relationship between the PLCR niche fitness function and that of the BG niche is far from obvious. The reasons why structures that evolve to suit the latter constitute viable solutions for the former can be seen, however, in the phenotypes of the sample individuals shown in Figure 6.19. Highly evolved individuals in the BG niche tend to be variations on the theme shown on the left side of the figure. They possess a cluster of S and A boxes (green and aqua colours, respectively) in the center of the grid. The purpose of the cluster is to multiply the number of game balls while preventing their point values from dropping below zero. To either side of this cluster are flanking layers of L and R boxes (red and blue, respectively). Their purpose is to redirect escaping game balls back into the cluster and to funnel them at the bottom into the sink column where points are eventually tallied. Finally, the columns of A boxes leading from the top of the grid at the source column to the cluster and from the cluster to the bottom of the grid in the sink column boost game ball point values before and after passage through the cluster.

Although this strategy constitutes a sensible approach to the BG niche problem, it's also composed of large adjacent single-color regions connecting the top of the grid to the bottom. This side-effect property is what the PLCR niche seizes upon and adapts further.

They are precisely the structures considered viable in the PLCR niche, and they are never generated by the random population initialization technique used in the GA, as explained above. This relationship allows evolved BG niche solutions to be exapted to seed the PLCR niche population.

6.4.3 The Sieve Niche Fitness Function

The Sieve niche's fitness function does not distinguish between the box types. It interprets blank cells as forming holes in an otherwise solid structure composed of boxes. It's called the Sieve niche by analogy to physical sieves. A highly "fit" sieve is one possessing a large number of small holes and no holes larger than a given size. Two blank cells are considered part of the same hole if they are within one column and within one row of each other. In the Sieve niche, an individual is considered inviable if it possesses any holes larger than five cells in area. Such individuals are assigned zero fitness. Otherwise, an individual's fitness is simply the number of holes in its phenotype.

The relationship between the Sieve niche fitness function and that of the PLCR niche is also not an obvious one. The connection can be seen, however, by examining the sample individuals shown in the middle of Figure 6.19. As the PLCR population evolves, the largest region of each box type expands to fill roughly one quarter of the grid. As these regions expand, they fill in any holes present in the grid. Eventually there are no holes larger than five cells in area, at which point the PLCR individual becomes a viable Sieve individual. These individuals can then migrate to the Sieve niche and begin evolving in a new fitness landscape. Individuals from the original BG niche, however, never migrate directly to the Sieve niche since they always contain at least one prohibitively large hole.

Under the given fitness functions and population initialization method, described above, two exaptation events are required to evolve viable Sieve solutions in this GA; the first is from the BG niche to the PLCR niche, and the second is from the PLCR niche to the Sieve niche.

6.4.4 The CRD Niche Fitness Function

The fourth and final niche, called the *Coloured Ring Distances* niche scans the grid for instances of a particular pattern of boxes called a colored ring. A colored ring is a single blank cell surrounded in all eight directions (four cardinal directions and four diagonals) by a ring of boxes composed of at least two types or colours. The fitness function for an individual in the CRD niche is

$$fitness(i) = \max(0, d^2 - 4c) \quad (6.4)$$

where d is the number of unique pairwise Manhattan distances between the centers of coloured rings on the grid, and where c is the chromosome length. Individuals with zero fitness are considered inviable.

This particular fitness function renders highly fit individuals in the Sieve niche viable. Such individuals tend to possess a large number of coloured rings, as can be seen in the sample sieve individual in Figure 6.19. Many of those rings can be safely removed by accumulated deletion mutations in the CRD niche since they will not contribute to d , the number of unique pairwise Manhattan distances. The $-4c$ component of the fitness function drives a reduction in chromosome length until highly fit phenotypes are produced. One example is shown on the right side of Figure 6.19.

Neither the BG niche nor the PLCR niche produces viable CRD individuals. The population initialization technique is also unable to do so. Thus the GA requires three exaptation events to produce viable CRD solutions. Each exaptation event can be thought of as the GA escaping a zero-fitness plateau in the destination niche's fitness landscape by climbing a hill in the source niche's landscape.

6.4.5 Migration

At the end of each generation in the GA, a check is made to determine whether or not any individuals migrate from one niche to another. There are six separate one-way migration routes available between the four niches. These are shown in Figure 6.20. Routes 1, 2, and 3 represent the sequential stepping-stone exaptation events that allow the GA to produce viable individuals in the final CRD niche. Routes 4, 5, and 6 represent shortcuts that would allow the GA to skip an intermediate step. These latter routes are provided and tracked in order to show that this GA is extremely unlikely to skip these intermediate steps between the first and fourth niches. This is verified experimentally in the results section, below.

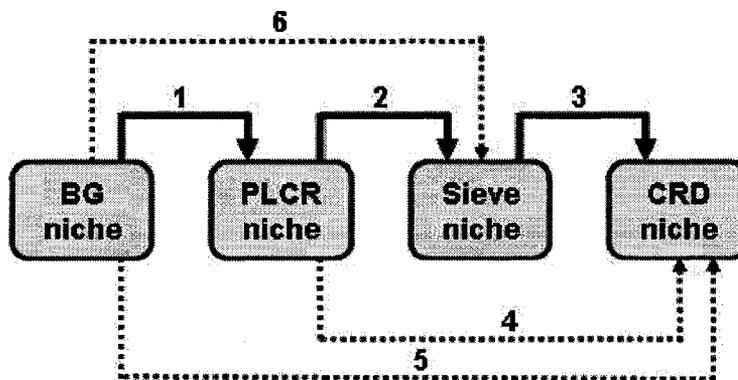


Figure 6.20 Potential and actual migration paths between the four niches in the four niche exaptive GA. Actual paths in bold.

For each potential route, the migration check proceeds as follows.

1. An individual is chosen from the source niche by a tournament of size two. Only viable individuals enter the tournaments. If there are no viable individuals in the source niche, no migration takes place.
2. The chosen individual's fitness is evaluated using the destination niche's fitness function.
3. If the individual is inviable in the destination niche, no migration takes place.
4. If the individual is viable in the destination niche and has higher fitness than the least fit viable individual already present there, it is copied to the destination niche, overwriting the least fit individual. If the destination niche had originally possessed no viable individuals, the migrant has successfully seeded a new population, and will be the sole parent available to produce the next generation (provided no other source niches send a migrant during the migration check).

Any successful migrant in the GA represents an exaptation event. In such cases a structure evolves to suit one function (viability in the source niche) and is co-opted for another function (viability in the destination niche). Three successive cycles of adaptation and migration can lead the GA to viable solutions in the final niche.

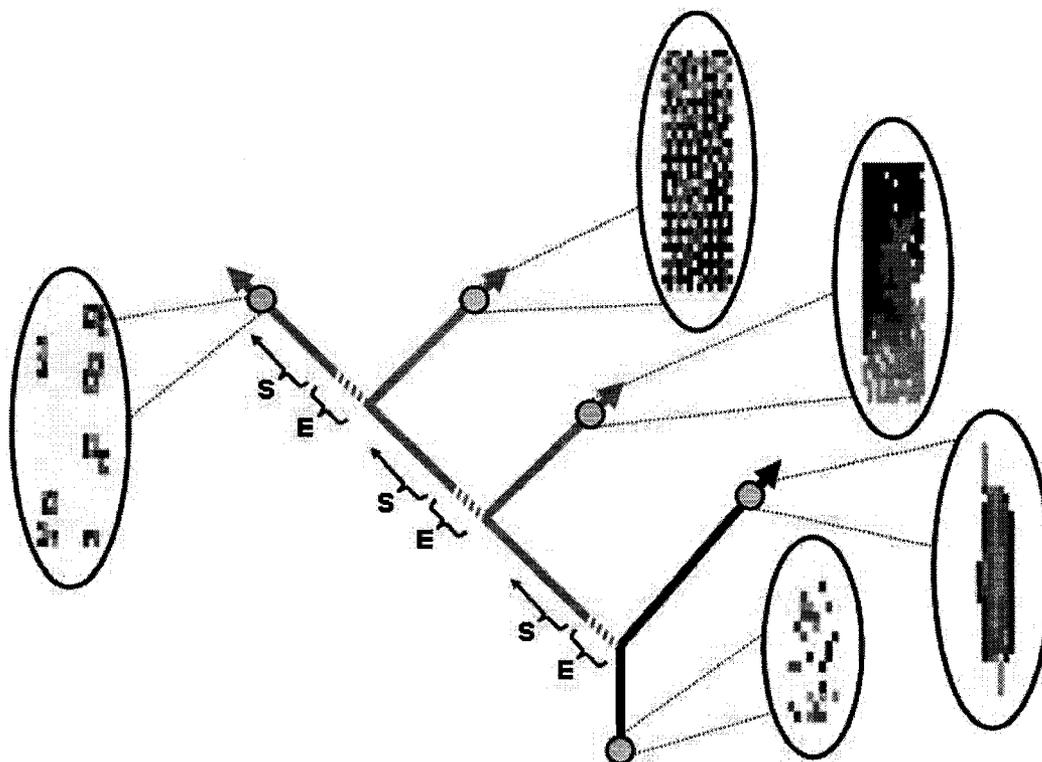


Figure 6.21 A cladogram depicting evolution, exaptation, and secondary adaptation in the four niche exaptive GA.

6.4.6 Results

The four niche exaptive GA was run 455 times to collect the data presented in this section. Each run was terminated after 20,000 generations. A number of population statistics were recorded for each run at regular 80-generation intervals. Each run used the parameters and operators described and used in previous sections. Figure 6.21 shows a cladogram depicting the evolution of the four populations in the GA. Exaptations and periods of secondary adaptation are labeled “E” and “S” in the cladogram. Migration events always occurred in the order indicated in the cladogram, and the phenotypes shown are very representative of those present at the various stages and in the various niches shown.

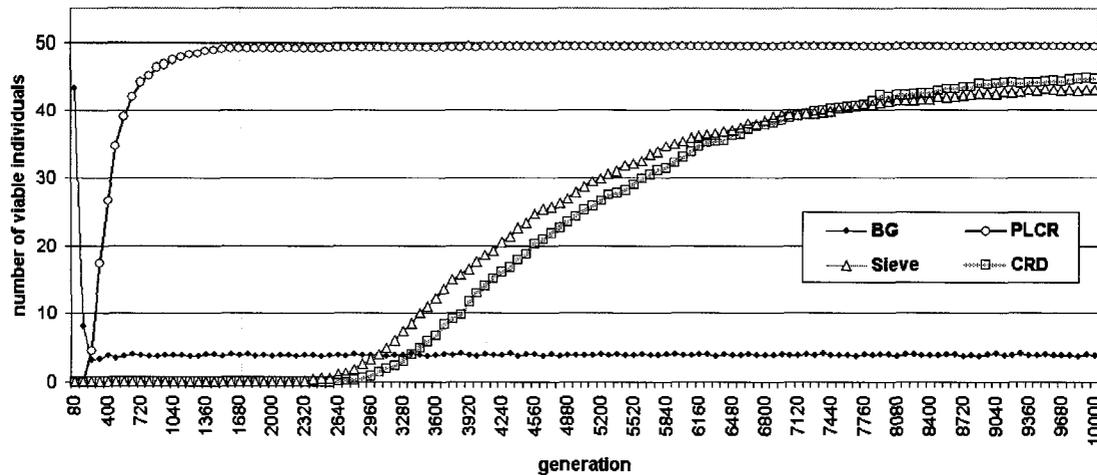


Figure 6.22 Number of viable individuals vs. generation in all niches of the four niche exaptive GA.

6.4.6.1 Appearance of Viable Individuals

In all 455 runs, population initialization produced viable individuals for the BG niche but only inviable individuals in the other three niches. The PLCR, Sieve, and CRD niches received no viable individuals until at least 160 generations into each run. Figure 6.22 shows the mean number of viable individuals in each niche over time. The overall pattern suggests that the population spreads through the niches in order from niche 1 to niche 2 to niche 3 and then niche 4. The results obtained by tracking successful migration, below, show this to be an accurate description of events.

Figure 6.22 shows only the first 10,000 generations. The viability count averages remain effectively constant for generations 10,000 to 20,000 and were omitted for that reason. The low viability counts in the BG niche arise from the way in which the exaptive GA makes duplicates very unlikely, and because of the interlocking complexity of the evolved solutions making them very brittle under the variation operators that produce offspring, as explained above.

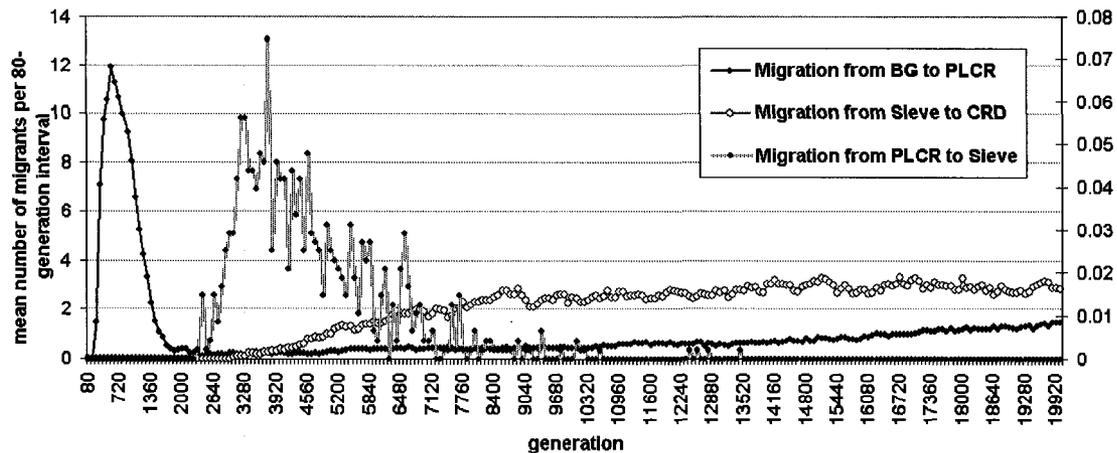


Figure 6.23 Mean number of migrants between niches vs. generation in the four niche exaptive GA.

6.4.6.2 Potential vs. Actual Migration Routes

Figure 6.20 shows the six potential one-way migration routes between the four niches in the GA. In all 455 runs, however, only routes 1, 2, and 3 (between BG and PLCR niches, PLCR and Sieve, and between Sieve and CRD, respectively) were successfully traversed by migrants. Mean migration frequency graphs for routes 1, 2, and 3 are shown in Figure 6.23. Mean migrant counts between the PLCR and Sieve niches are plotted against the secondary vertical axis on the right of the figure. All others are plotted against the primary axis on the left. No individuals appeared in any run that could successfully traverse routes 4, 5, or 6 (see Figure 6.20).

6.4.6.3 Exaptation Events: Sequence and Rates

Together with the fact that population initialization produces viable individuals only in the BG niche, the results from tracking migrations, above, allow us to conclude the following. Without exception, all runs which produced viable individuals in the CRD niche (only 5 of the 455 runs failed to do so) did so by using exaptation three times. First,

structures that evolved for at least 160 generations in the BG niche were exapted for use in the PLCR niche. Second, structures in the PLCR niche evolved further, accumulating secondary adaptations until, as a side-effect of their expanding coloured regions, they met the minimal requirements for viability in the Sieve niche and were exapted for its fitness function. And third, structures in the Sieve niche accumulated new secondary adaptations until they reached minimum requirements for viability in the CRD niche where they were exapted and refined through a third phase of secondary adaptation.

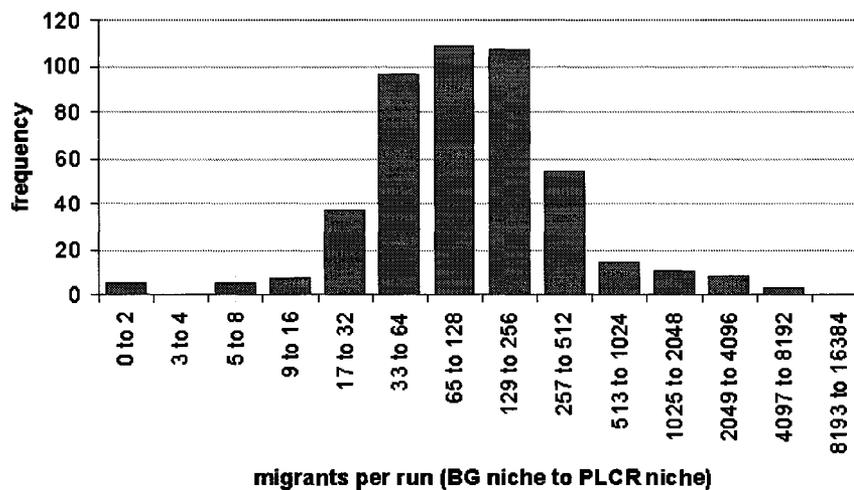


Figure 6.24 Frequency distribution for the number of migrants per run between the BG and PLCR niches in the four niche exaptive GA.

The mean number of migrants per 80-generation logging interval is shown in Figure 6.23 for each of the three migration routes traversed by the populations. With the exception of migration between the PLCR and Sieve niches, these graphs show a single rapid surge followed by a rapid decline. This can be accounted for by newly-seeded destination niche populations increasing in fitness to the point where new potential migrants from the source niche can no longer compete, making further exaptations very unlikely.

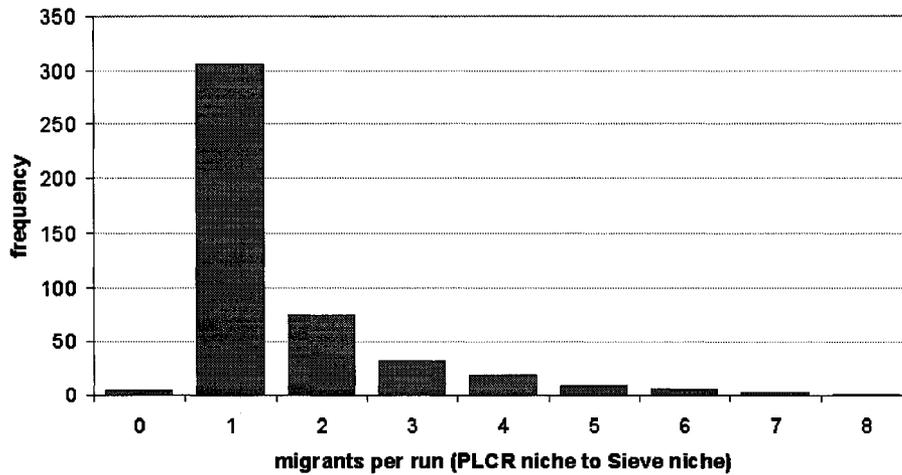


Figure 6.25 Frequency distribution for the number of migrants per run between the PLCR and Sieve niches in the four niche exaptive GA.

The distributions for the number of migrants per run for each of the three traversed routes can be seen in Figure 6.24, Figure 6.25, and Figure 6.26. The mean number of successful migrants per run from the BG niche to the PLCR niche is 257. The mean for route 2 between the PLCR niche and the Sieve niche is 1.6. The mean for the route connecting the Sieve niche to the CRD niche is 486 successful migrants per run.

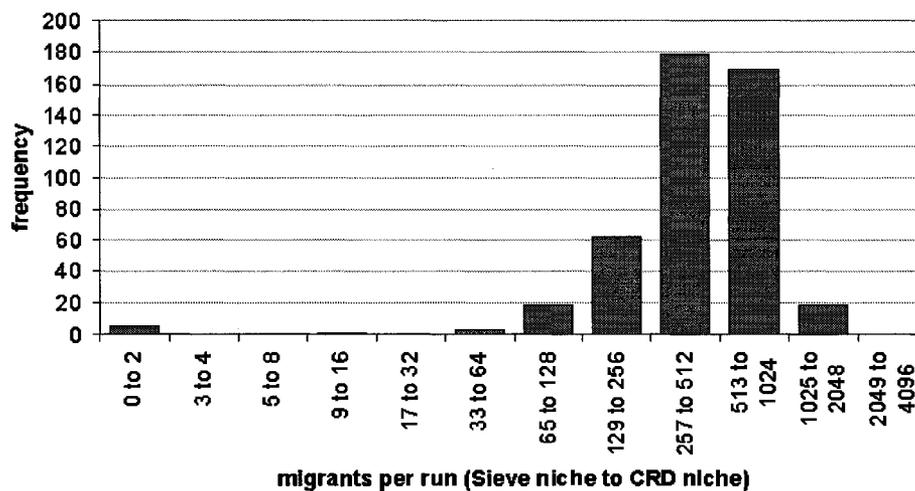


Figure 6.26 Frequency distribution for the number of migrants per run between the Sieve and CRD niches in the four niche exaptive GA.

6.4.7 Conclusions

While it is clear that the fitness functions in the four niche exaptive GA were designed deliberately as a means to an end – to generate exaptations – instead of the other way around, this model may still represent a small step in a potentially fruitful direction.

Given that exaptation is one of the means by which biological evolution has produced a large variety of complex structures and designs, it may be a worthwhile goal within the EC community to explore this potential.

As abstract simulations involving exaptation, models like these may be useful tools for exploration of higher-level evolutionary phenomena such as exaptation, speciation, and others. From an optimization and design perspective, however, if exaptation-facilitating relationships between real-world fitness functions can be constructed or discovered, like those between the niches in this model, then exaptation may provide a valuable problem-solving tool for EC.

In the four niche exaptive GA, exaptations are used to steer a population (divided into niches) into different regions of phenotype space, each time switching focus from adaptive aspects of the phenotype structure to side-effects of those adaptations.

Sequential exaptive GAs for real-world problems could potentially lead the population into more and more complex and structured regions of the search space. These complex regions of the search space, if sought in a single fitness function GA, might require the focusing of domain-specific information and ingenuity on construction of the fitness function in order to create gradients of fitness that slope toward desired outcomes, or on a population initialization technique that starts the GA off with partial solutions instead of

completely random genomes. This does not mean that an exaptive GA approach necessarily reduces the burden placed on the would-be builder of a workable GA. Instead, the focus is shifted away from construction of an effective fitness function or initialization technique to the construction of intermediate selection pressures that change over a run. An exaptive approach, therefore, can be thought of as another possible avenue of pursuit when building a GA or other EC system.

6.5 The Combination-Fitness Exaptive GA

In section 6.3.6 a modified version of the two-niche exaptive GA was constructed that included a third niche combining both the BG and LEA fitness functions. When BG and LEA measures were used as axes in a 2D fitness space, the mean and median population trajectories through this space showed that the niche using the combined fitness function was able to outperform the LEA niche in terms of the LEA fitness measure. The combination-fitness exaptive GA explores this in more detail.

6.5.1 The Fitness Function before Exaptation

The combination-fitness exaptive GA employs only one niche. Exaptation takes place when the initial fitness function used by the niche is replaced by another function after a predetermined number of generations. Prior to the change, the fitness function combines both the BG fitness measure and the LEA fitness measure. The BG measure used is formula (6.1.5-2). This is the version of the BG fitness function that does not involve a chromosome length penalty. The LEA measure is a modified version of the fitness function described in 6.3.3. The modifications are as follows.

- The unbroken connection between the top and bottom of the grid is no longer required.
- No minimum value is imposed on the area of the largest enclosure.
- The largest enclosed area measured is incremented by 1.

These modifications get around the problem of inviable individuals and relax restrictions on the initial structure required before fitness can accrue. Randomly-generated individuals, even those containing no enclosed regions in their phenotypes, are considered viable. This allows the GA to begin working with both BG and LEA measures right away.

An individual's fitness is computed as

$$\begin{aligned} fitness(i) &= 0 && \text{if } BG(i) < 1 && (6.5.1-1) \\ fitness(i) &= (\log_{10}(BG(i)))(LEA(i)) && \text{otherwise} \end{aligned}$$

where $BG(i)$ and $LEA(i)$ are the BG and LEA measures, respectively, for individual i .

6.5.2 The Fitness Function after Exaptation

Once the predetermined generation number has been reached, the fitness function changes to the LEA measure alone. By varying the generation number for the changeover event and observing the resulting fitness trajectories, one can attempt to find the optimal amount of preadaptation that will produce the greatest improvement in fitness compared to a GA where the changeover occurs right away (i.e. a regular GA with LEA fitness).

6.5.3 GA Parameters

The parameters for the combination-fitness exaptive GA are identical to those described in section 6.1.1 and Table 6.1. The mutation and crossover operators are the same as those described in sections 6.1.3 and 6.1.4. Each run extends for 5000 generations, and logging data are collected at 20-generation intervals for changeover generation numbers 0, 200, 700, 1200, 1700, 2200, 2700, 3200, 3700, and 5001. Runs with changeover at generation 0 behave like a regular non-exaptive GA using the LEA fitness function. Since generation 5001 is never reached, runs with changeover at generation 5001 behave like a regular non-exaptive GA with the combination-BG-LEA fitness function. 5000 generations is a run length that is long enough to allow most of the GA configurations to attain fitness values close to the LEA fitness function's global optimum.

6.5.4 Fitness Trajectories

This section shows the mean fitness trajectories for each of the configurations listed above. The combined-LEA-BG fitness measure is ignored in favour of the LEA measure alone since those are the values that make for a proper comparison between the configurations.

6.5.4.1 LEA Fitness Trajectories

Figure 6.27, below, shows the mean best LEA value for each configuration. The nine points of sudden departure from the E5001 fitness trajectory, each 500 generations beyond the last, correspond to different preadaptation epoch lengths (200 generations, 700 generations, 1200 generations.. etc). The remaining trajectory (E0) is that of a normal non-exaptive GA. The number of independent runs collected for each configuration varies from one hundred to over three thousand.

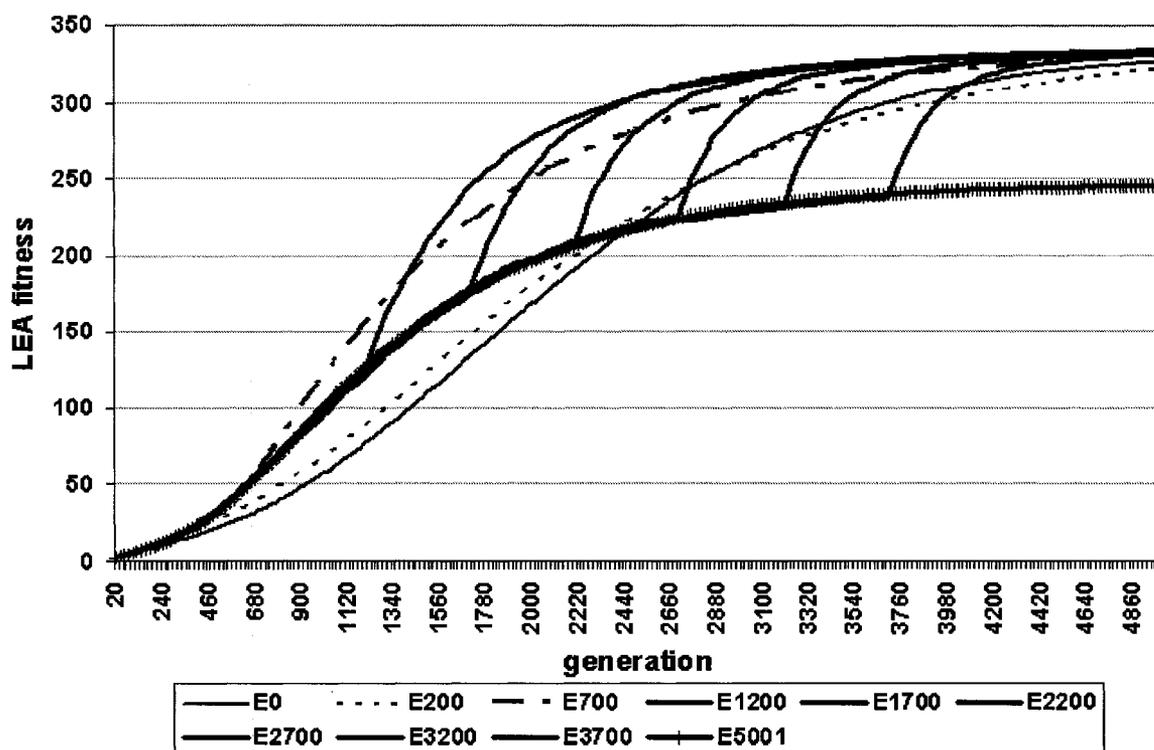


Figure 6.27 Mean best LEA fitness trajectories for the combination-fitness exaptive GA.

The data series in the legend of Figure 6.27 are labeled by the generation numbers of their scheduled exaptation events. The E0 series shows the behavior of a normal non-exaptive GA using the LEA fitness function in all generations. With the exceptions of both E5001, which involves an exaptation event that is never reached, and E200, all trajectories exceed E0 in terms of the mean final best fitness at generation 5000. The

differences at generation 5000 are small since runs of this length provide sufficient time for most populations to converge to fitness values near the global optimum value of 336. Although the fitness differences in generation 5000 are quite small, each of the exaptive configurations shows larger absolute improvements over E0 at other generations in the run. Even those that do not outperform E0 by generation 5000 (E200 and E5001) would outperform it given earlier generation numbers as halting conditions.

Figure 6.28, below, shows a zoomed-in view of the final generations from Figure 6.27. The E5001 series is omitted to allow better scaling of the y-axis. The configuration with the greatest performance at generation 5000, as seen in the figure, is E1700. A two-tailed Mann-Whitney U test comparing final best fitness values from E0 and E1700 gives a p value for the comparison which is less than or equal to 0.0001, meaning that the difference is statistically significant⁵. p values for comparisons between E0 and all other configurations are given in Table 6.2, below.

⁵ This is the minimum p value obtainable using the ALGLIB software library's Mann-Whitney U test [Bochkanov & Bystritsky, 2008].

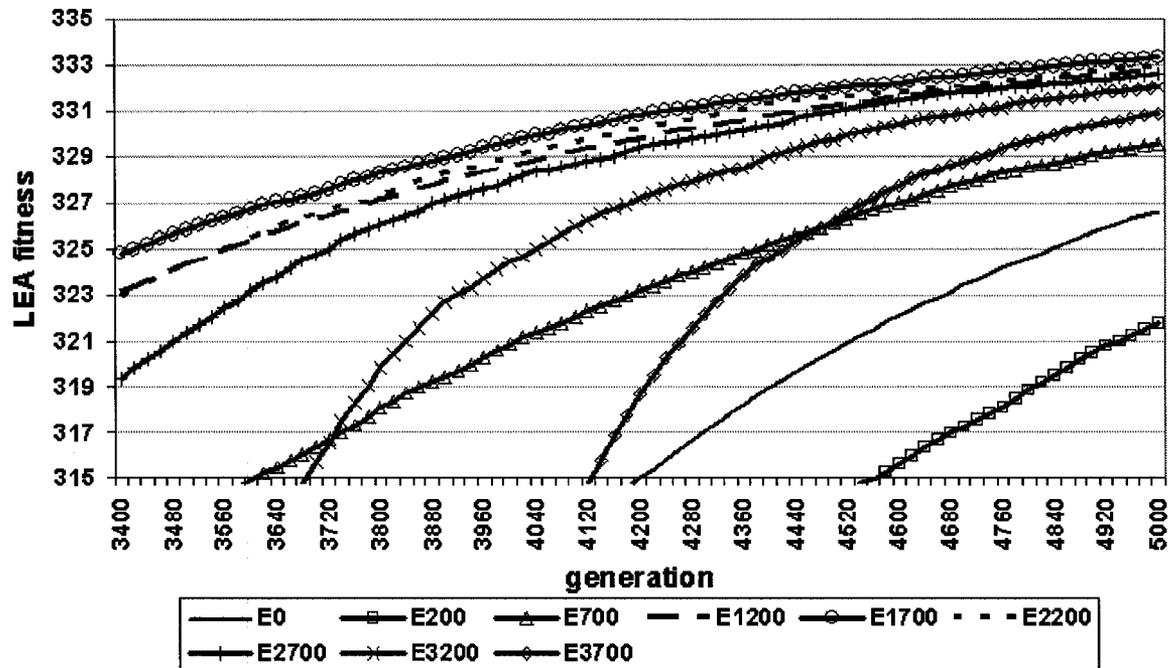


Figure 6.28 Mean best LEA fitness trajectories in the final generations of the combination-fitness exaptive GA.

Table 6.2 and Figure 6.29, below, show the final best fitness values as a function of the amount of preadaptation (the number of generations before the fitness function changeover occurs).

Configuration	Mean Best Fitness	Standard Deviation	5 th Percentile	Mann-Whitney U Test p value (comparison with E0)	Sample Size
E0	326.72	12.968	300.00		3987
E200	321.78	13.206	298.90	0.0001	150
E700	329.51	7.143	316.90	0.944516	150
E1200	332.76	2.811	327.00	0.0001	2065
E1700	333.30	2.103	329.45	0.0001	150
E2200	332.89	2.403	328.95	0.0001	100
E2700	332.62	2.343	328.00	0.001523	100
E3200	332.06	2.477	327.95	0.053505	100
E3700	330.89	2.957	325.00	0.895464	100
E5001	245.74	10.415	225.90	0.0001	150

Table 6.2 Mean best fitness and other statistics for the combination-fitness exaptive GA.

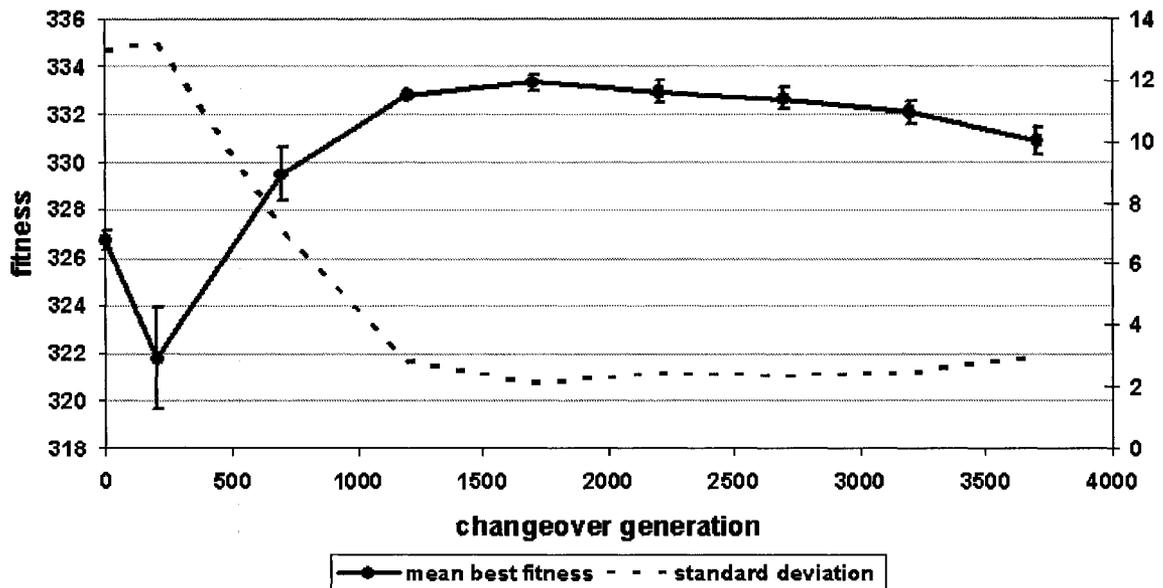


Figure 6.29 Mean final best fitness in the combination-fitness exaptive GA as a function of preadaptation time.

The error bars in the figure are 95% confidence intervals. The figure seems to show that, not only does there exist a single optimal amount of preadaptation for the GA, but that too little preadaptation can be worse than none at all. Furthermore, most of the exaptive configurations have a lower standard deviation (plotted against the secondary axis on the right side of the figure) meaning that those configurations are more robust than the regular GA.

6.5.4.2 LEA Fitness Improvement vs. Generation

Since each exaptive configuration exceeds the performance of E0 by differing degrees depending on the generation number used in the halting condition, an obvious next step is to determine the amount of preadaptation that gives the greatest absolute improvement considered over all generations. With this in mind, an alternative view of the data from

Figure 6.27 is shown in Figure 6.30, below. The figure shows the difference in fitness values between each configuration and E0 instead of the absolute fitness value.

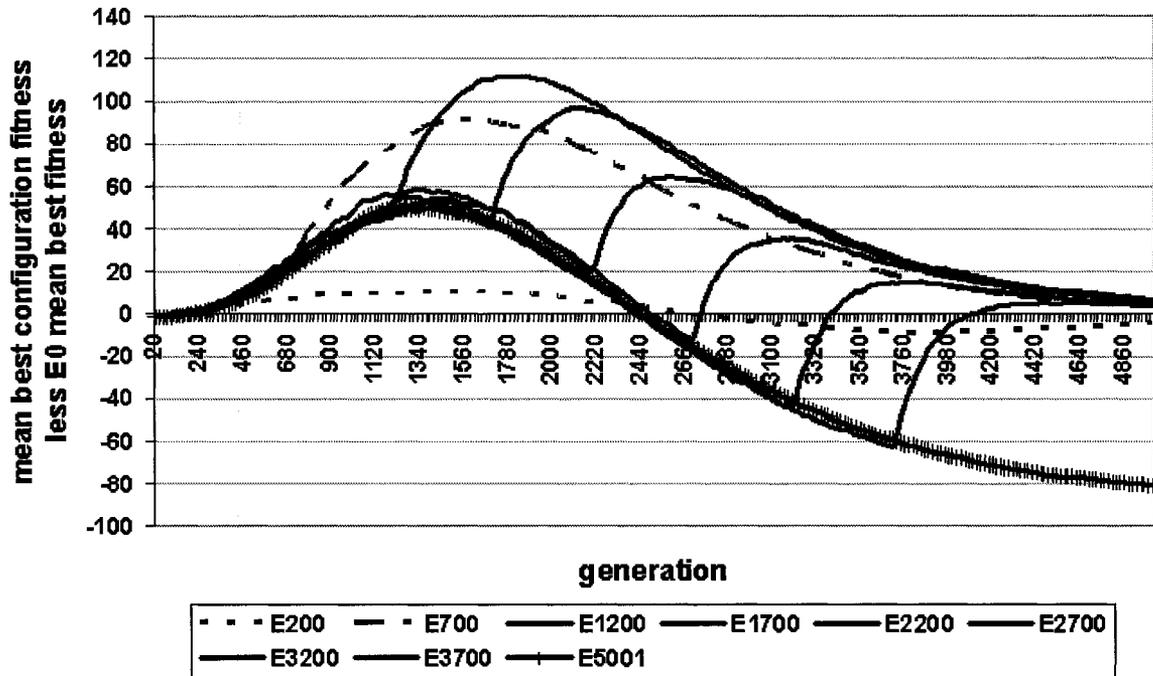


Figure 6.30 Absolute fitness improvements over the non-exaptive configuration in the combination-fitness exaptive GA.

From Figure 6.30 it can be seen that the largest absolute improvement is seen in configuration E1200. By generation 1800, the E1200 exceeds the non-exaptive E0 by over 110 points of LEA fitness out of a maximum of 336. For runs with a length of 1800 generations, this is an extremely significant benefit. If one examines the peak absolute fitness improvement as a function of the amount of preadaptation, one gets the graph shown in Figure 6.31, below. The error bars are 95% confidence intervals.

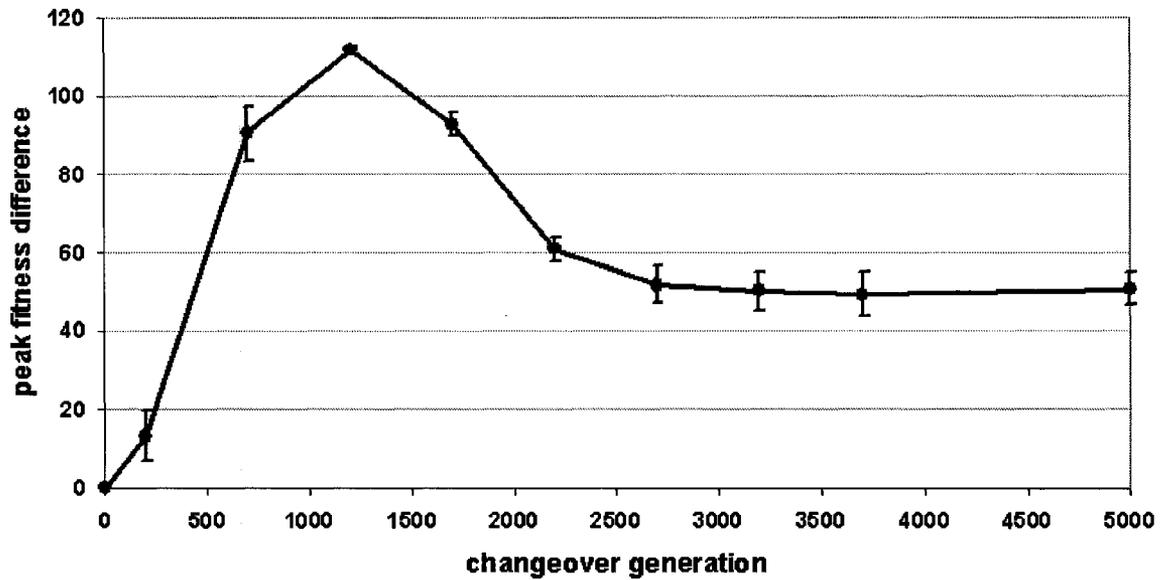


Figure 6.31 Peak LEA fitness improvement over all 5000 generations for the combination-fitness exaptive GA.

6.5.4.3 Success Rates

Since the global optimum for the LEA fitness function is known, one can compute the fraction of runs that reach the optimum at each generation to see whether the success rate data give the same picture as do the mean best fitness data. Since few runs reach the exact optimum of 336, any final best fitness of 334 or greater will be counted as a success.

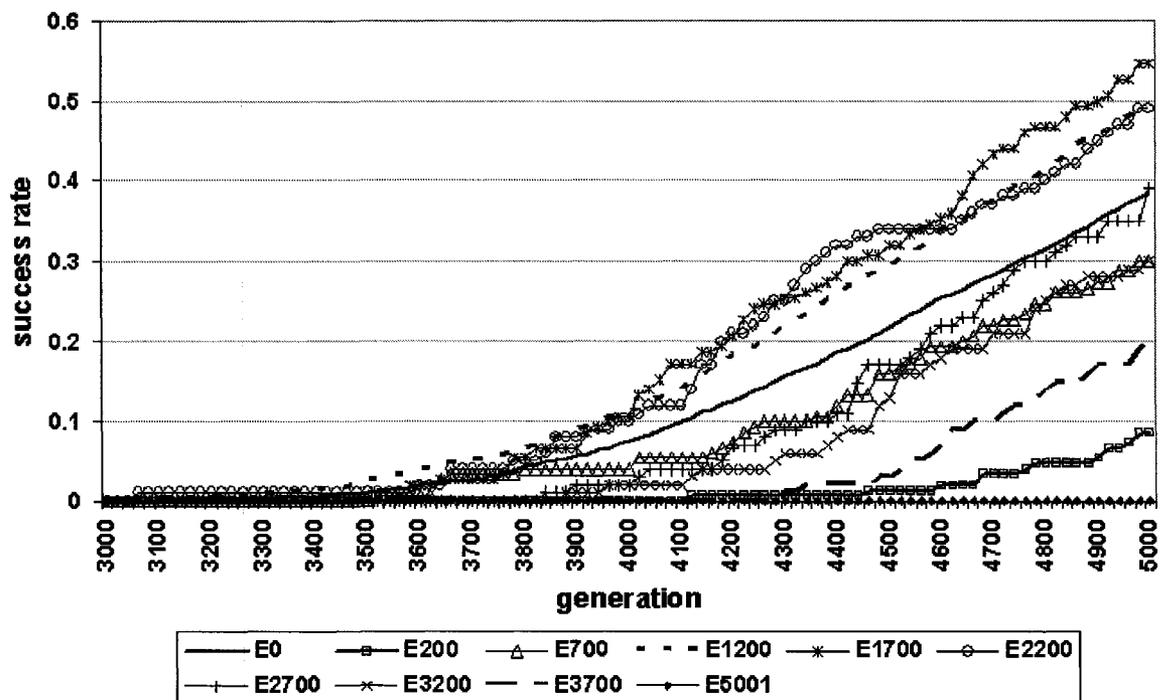


Figure 6.32 Success rates for the combination-fitness exaptive GA.

Figure 6.32 shows the success rates for each configuration over the final two fifths of the run. While nearly all exaptive configurations attained superior performance in terms of mean best fitness values, as shown in Figure 6.29, only a few do so with regard to the success rate⁶. The three that do so (E1200, E1700, and E2200), however, are also the three with the highest mean best fitness performance.

Figure 6.33, below, shows the final success rate as a function of the amount of preadaptation. The same detrimental effect of having too little preadaptation, as seen in Figure 6.29, is evident with this statistic as well. The error bars in the figure are 95% confidence intervals.

⁶ While this is the case with a success threshold of 334, dropping the threshold by as little as 4 points, to 330, lifts all configurations except E200 and E5001 above E0 in terms of the final success rate.

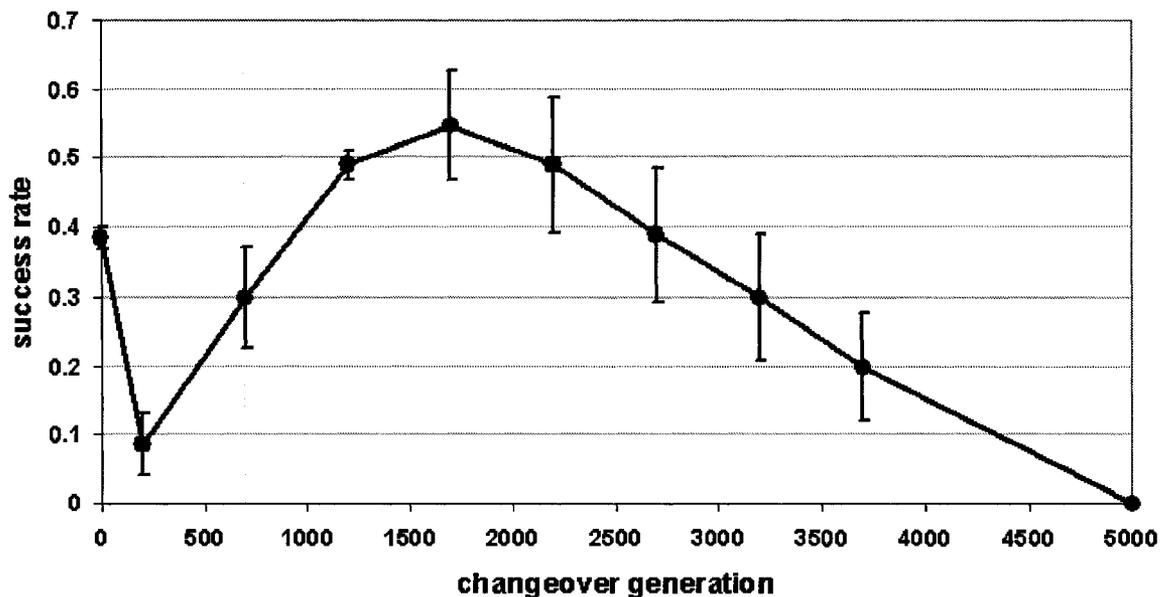


Figure 6.33 Success rate as a function of preadaptation time in the combination-fitness exaptive GA.

6.5.4.4 Trajectories in 2D Fitness Space

Figure 6.34, below, applies the 2D graphing idea described in section 5.3.3. It shows the mean paths taken by each of the configurations through a 2D fitness space⁷. The ten configurations can be easily identified without need of a legend in the figure. The leftmost and rightmost trajectories are the E0 and E5001 configurations, respectively. Those crossing from the latter to the former are E200, E700, E1200, E1700, E2200, E2700, E3200, and E3700. To provide a sense of the speed with which the GA populations travel through the space, the first ten 200-generation intervals are marked with circles for both the E0 and E1200 paths. E1200's ascent along the base of the E5001 s-curve, followed by a sharp left turn to reach the top of E0's more or less straight path, is

⁷ In this case the x-axis is not truly a fitness function used at any point in any of the combination-fitness exaptive GA's configurations. Instead, it is the BG measure, one of the two components used in the preadapting fitness function prior to exaptation. The use of this measure as an axis instead of the mixed fitness keeps the two axes independent of one another, in some sense.

a kind of short-cut allowing it to reach about 100 LEA fitness points ahead of E0's regular GA in an equal amount of time.

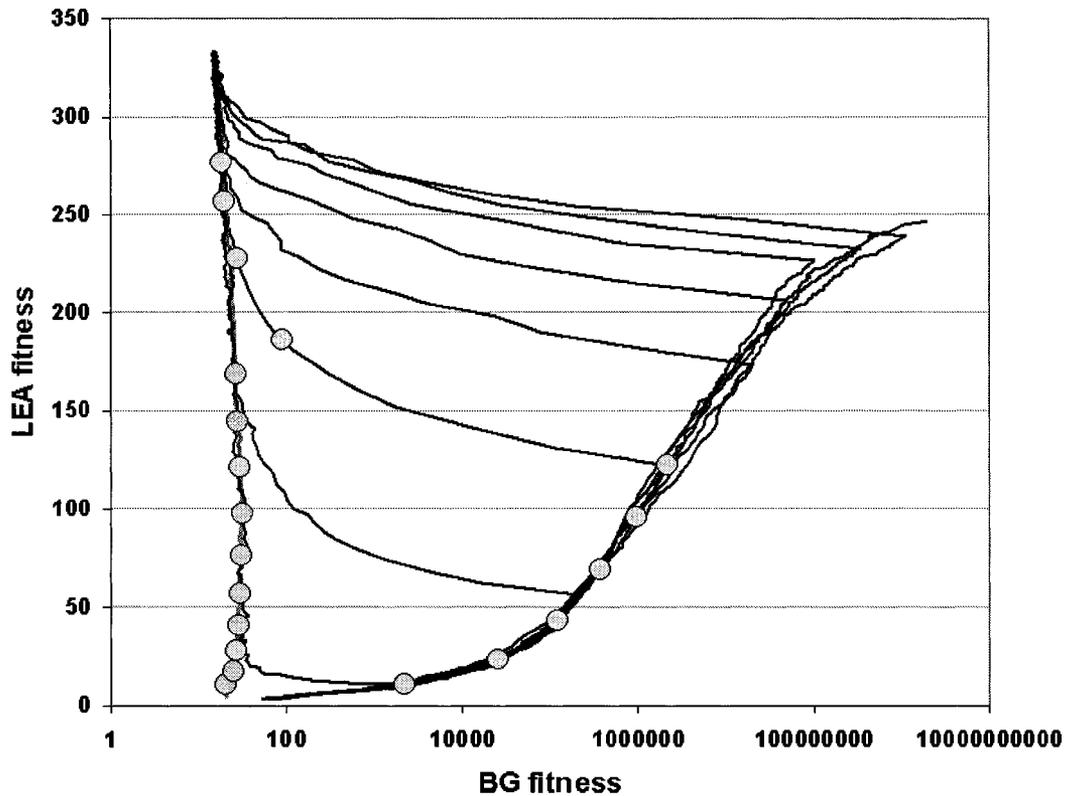


Figure 6.34 Combination-fitness exaptive GA trajectories through 2D fitness space.

6.5.5 The Role of Plateau- and Valley-Crossing

The data presented in section 6.5.4, above, show a marked increase in the speed with which the exaptive GA can improve LEA fitness compared to an otherwise identical but non-exaptive GA, especially in the early and middle generations of a run. The fitness data alone, however, cannot shed light on any of the reasons for this improvement. The present section examines parent-child relationships in both the E0 and E1200 configurations. Included in the logging data of the GA are classifications for the

relationships between the elite individual in a population and its parent⁸. For the E0 configuration the following are the two classes of parent-child relationship.

- The LEA fitness of the child is higher than that of its parent
- The LEA fitness of the child is equal to that of its parent

The elite individual, by definition, can never have a fitness value lower than that of its parent. By examining the frequency of each type of relationship, taken over many independent runs, one can get some idea of the rate with which the E0 GA discovers improvements and of the rate with which it explores plateaus in the fitness landscape by moving from one genotype to a different but equally fit genotype. To compare this data with that from the E1200 exaptive GA, the latter's parent-child relationships are divided into the following six classes. The LEA fitness value is referred to as "L", and the fitness that combines LEA and BG components is referred to as "C".

- Child L and C are higher than parent L and C
- Child L is higher than parent L, but the two have equal C values
- Child L is equal to parent L, but child C is higher than parent C
- Child L and C are equal to parent L and C
- Child L is lower than parent L, but child C is higher than parent C
- Child L is lower than parent L, but the two have equal C values

⁸ In the case of elite individuals produced by crossover, one of the two parents is chosen at random. Note that, even if an elite individual remains elite for many generations, the parent with which it is compared is its true parent, and not the identical individual from which it may have been cloned in the previous generation.

Because of elitism, no classes exist for cases in which the child has a lower fitness (C value) than its parent. By summing the totals from the first two classes, the second two classes, and the last two classes, the E0 and E1200 configurations can be compared directly to see where the differences lie, if any. By splitting each sum back into its two components, some more detail about the nature of those differences may be obtained.

6.5.5.1 Results

Figure 6.35, Figure 6.36, and Figure 6.37, below, show the probability of an elite individual having better, equal, and worse LEA fitness than its parent, respectively. A comparison is made between the non-exaptive E0 configuration and E1200.

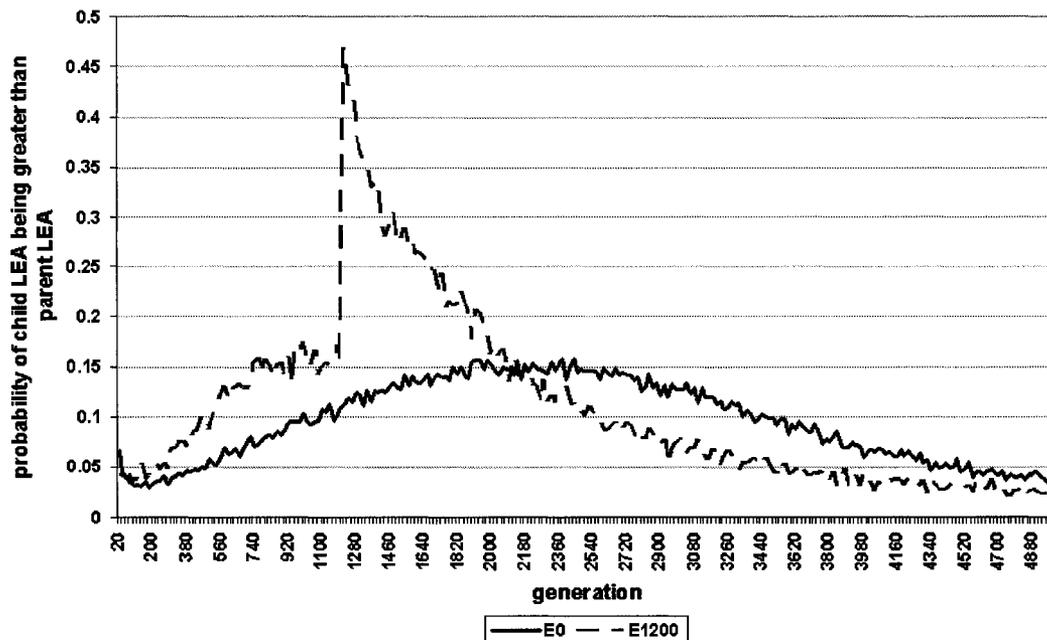


Figure 6.35 Probability of the elite individual having higher LEA fitness than its parent's in the combination-fitness exaptive GA.

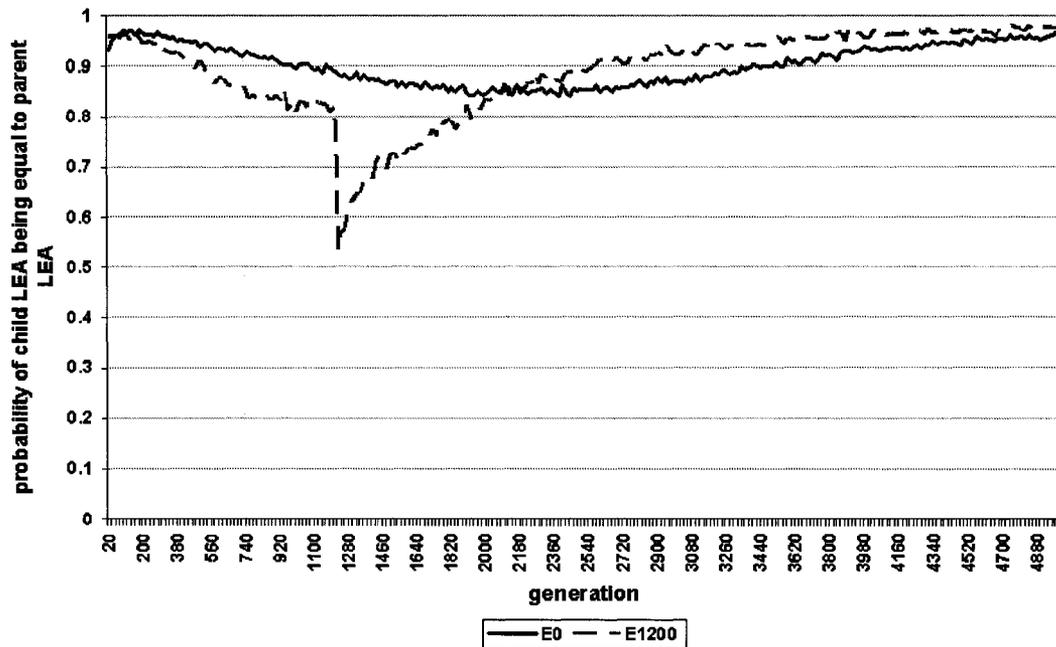


Figure 6.36 Probability of the elite individual having LEA fitness equal to its parent's in the combination-fitness exaptive GA.

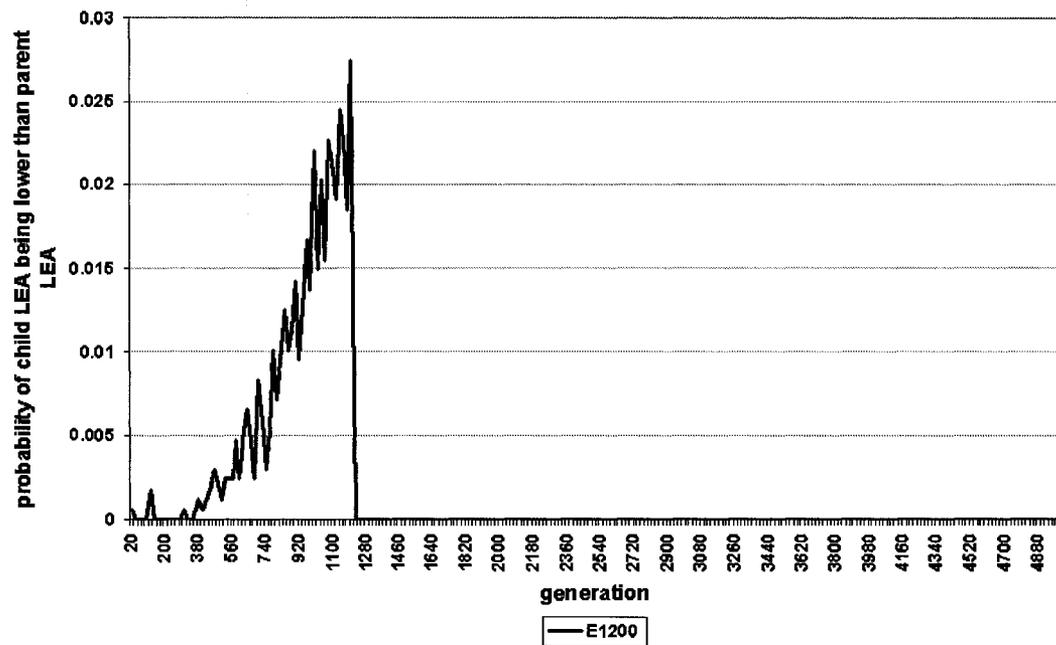


Figure 6.37 Probability of the elite individual having LEA fitness lower than its parent's in the combination-fitness exaptive GA.

Figure 6.36, showing the rates at which elite individuals are produced with LEA fitness values identical to their parents, indicates that this class of parent-child relationship accounts for nearly all elite individuals except during a short period in E1200

immediately after the exaptation event. This is true for both the non-exaptive and exaptive configurations before generation 1200, with the exaptive configuration having only a slightly lower rate for this type of occurrence. The balance swings in the opposite direction following the burst of activity right after generation 1200. The variation operators in the GA, described in sections 6.1.3 and 6.1.4, make it so that offspring are extremely rarely identical to their parents. Since both E0 and E1200 produce so many elite offspring that are LEA-neutral variations on their parents, it seems that both versions of the GA traverse fitness plateaus in a similar manner.

Since nearly all elite individuals either match or exceed their parents in terms of LEA fitness, Figure 6.35 and Figure 6.36 are nearly mirror images of one another. Figure 6.35 shows the second most common relationship class for both E0 and E1200, namely that of elite individuals having LEA fitness values that are superior to their parents'. E1200's rate exceeds that of E0 by about 6% on average before generation 1200. While this is a small absolute difference in rate, similar to that shown in Figure 6.36, it's a large difference relative to the magnitudes of the rates. Prior to the exaptation event, E1200 is about 1.5 as likely to produce elite individuals that are superior to their parents in terms of LEA fitness, and about twice as likely during the brief burst of activity afterwards.

Figure 6.37 shows a parent-child relationship for elite individuals that is only possible in the E1200 configuration since it uses a combination of both LEA and BG values in its fitness function before the exaptation event. The LEA value can decrease provided that the loss is offset by a sizeable increase in the BG value. These occurrences show the exaptive GA's ability to traverse fitness saddles that the non-exaptive GA cannot. Since

these occurrences account for less than 1% of all elite individuals produced before exaptation, it appears that saddle-crossing is not a major contributing factor in the success of the exaptive GA.

Since the E1200 configuration uses the combination fitness function prior to exaptation, and not the LEA fitness function alone, changes in the LEA value between a parent and its child may be different from the corresponding changes in the combination fitness value. Connections between the two may provide more information on the way in which E1200 can outperform E0. Figure 6.38, below, shows the data relating to this aspect of E1200's performance, but only up to generation 1200 since the use of the combination fitness function does not continue past that point. As in section 6.5.5, "L" refers to the LEA value, and "C" to the combination fitness value.

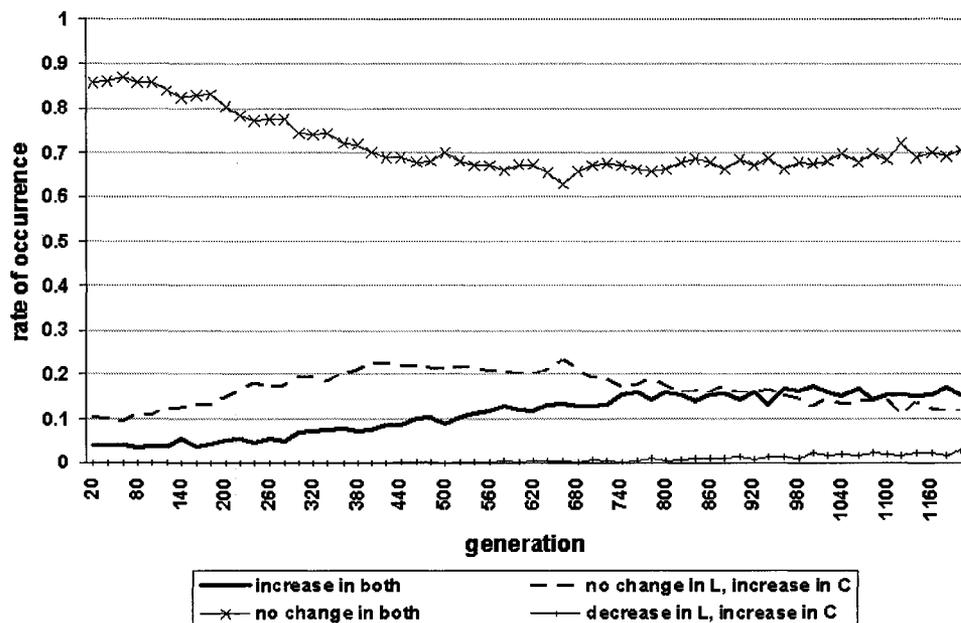


Figure 6.38 Occurrence rates for parent-child relationships in E1200 before exaptation.

Of the six classes of parent-child relationship listed in section 6.5.5, above, two have been omitted from Figure 6.38 because they occur at rates no higher than 3 outcomes in

1000 within the collected sample. Specifically, the omitted cases are those in which the LEA value either increases or decreases while the overall combination fitness value remains the same. Both cases are extremely unlikely given the manner in which BG and LEA values are combined (see formula (6.5.1-1)).

The figure shows that most elite children with LEA values unchanged from those of their parents also have unchanged combination fitness values, while a sizeable fraction correspond to increases in the combination fitness (and therefore increases in the BG component).

6.5.6 Conclusions

While the version of the LEA fitness function used in the combination-fitness exaptive GA has only a single optimum in the fitness landscape and no minimal structural requirements as did the original version in section 6.3.3, and is therefore a fairly simple problem, it can nevertheless shed some light on the possible evolutionary dynamics of exaptive GAs. The following observations and conclusions may be tentatively drawn from this section.

- The results provide strong support for the notion that an exaptation-based approach to some problems can outperform an otherwise normal GA.
- The benefit of an exaptive approach can be an increase in the mean best-of-run fitness, but also a decrease in the variability of the same, and thus can be an improvement in both mean performance and robustness.

- The amount of preadaptation can sway the outcome of an exaptive GA. The benefits of an exaptive approach might only accrue after a minimum amount of preadaptation, with less preadaptation being potentially detrimental. Thus, not only is it important to find a suitable preadapting fitness function, but it may also be important to find the right duration for that epoch in GA runs.
- The preadapting fitness function used in the combination-fitness exaptive GA is an augmented version of the desired fitness function (LEA) that combines the LEA measure with another (BG). While this function is more restrictive than the LEA fitness function since it attempts to maximize two quantities simultaneously instead of maximizing just one, and although this restriction is ultimately a hindrance (see the performance of E5001 in Figure 6.27, and Figure 6.30), it is nevertheless able to stimulate more rapid evolution than the desired fitness function alone. This may represent a strategy that more easily generalizes to other problems, as one may only need to devise an augmented version of the desired fitness function instead of coming up with an entirely separate function for preadaptation.
- In the specific case of the combination-fitness exaptive GA, neither plateau crossing nor saddle crossing appear to play a significant role in conferring an advantage to the exaptive GA over the regular GA. Beneficial variations (by mutation or crossover) are found more often and sooner in the exaptive GA, but the mechanism behind this remains unclear.

Chapter 7 Exaptation Experiments with a 2D GP Navigator

This chapter looks at an attempt to make easier a complex behavior-learning task by starting simple and increasing the difficulty through a sequence of exaptation events. Each exaptation event allows behavior learned in the previous phase to be a starting point in learning the behavior in the next phase. The task chosen is navigation control of a simple agent in a 2D environment. Two versions of the experiment are present, each representing a slightly different configuration and learning task.

7.1 Version 1

7.1.1 Description

This section describes version one of the 2D GP navigator, the simulated physical 2D environment with which the navigator agent interacts, the structure and operation of the navigator including its sensors, trackers, and effectors, the learning tasks to which the GP system is subjected in various epochs, and the parameters of the GP system.

7.1.1.1 The Environment

The navigator's environment is a bounded rectangular region of 2D space. The bounds are enforced by walls with which the navigator (and other objects) will collide. A snapshot of the navigator in the environment is shown below in Figure 7.1. The figure is augmented with labels designating the various types of objects in the environment,

including the navigator. In the figure there are 20 targets in total, but one is being carried by the navigator. The navigator is painted black as an indicator that it is carrying a target.

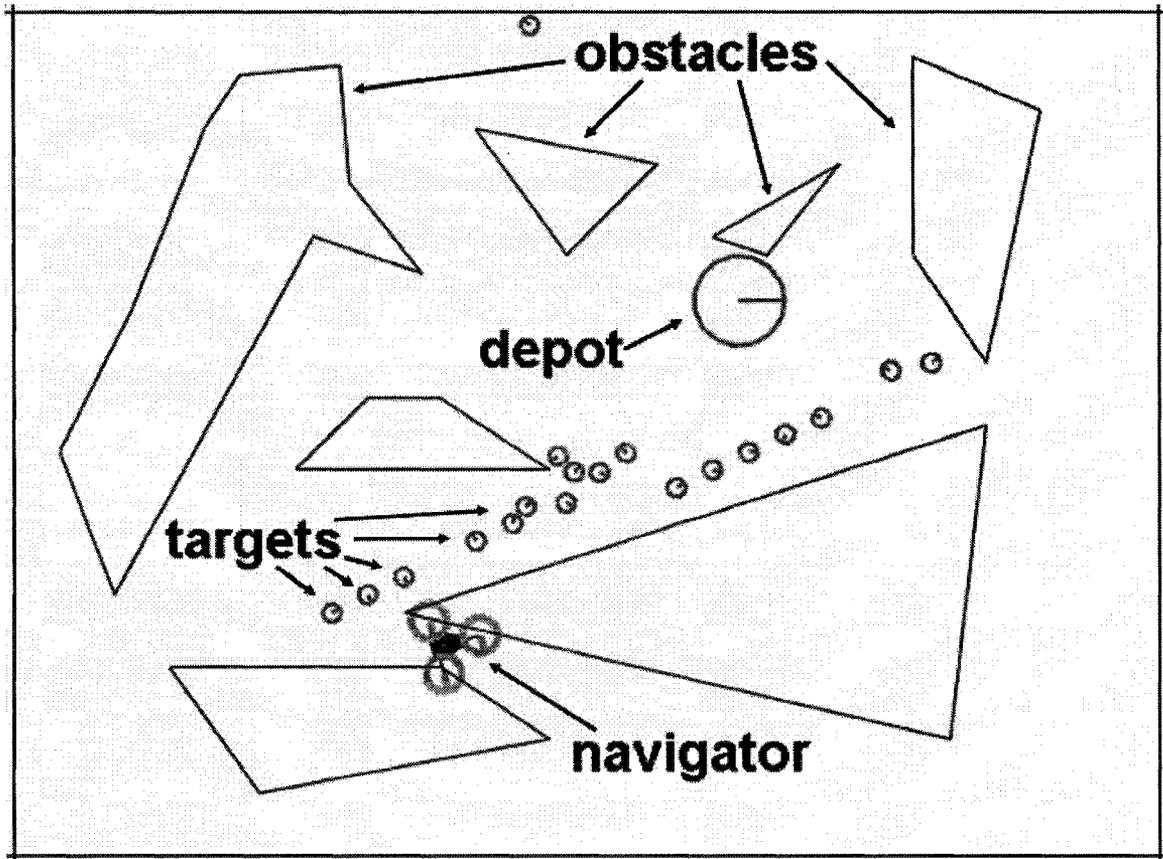


Figure 7.1 GP navigator environment and objects (version 1).

Obstacles in the environment are fixed static polygons whose edges are walls that trigger collisions when in contact with the navigator or with targets. The depot is a fixed static circle that triggers no physical collisions, but acts as a drop-off destination during epochs in which the system's learning task involves the collection of targets. Targets are small circular objects that move (in some cases of their own accord, in some cases due to external forces) and collide both with obstacles and with the navigator.

7.1.1.2 The Learning Tasks

The navigator's task in the environment depends on the epoch in which it finds itself.

There are six different epochs divided into two sequences, one exaptive, the other not.

The parameters and settings of each epoch are shown in Table 7.1, below, and subsequently explained.

	Sequence #1 Epoch 1	Sequence #1 Epoch 2	Sequence #1 Epoch 3	Sequence #1 Epoch 4	Sequence #2 Epoch 1	Sequence #2 Epoch 2
Tracked aspects of performance						
Distance from target(s)	•	•	•	•	•	•
Distance from depot				•	•	•
Maximum net distance traveled	•	•	•			
Number of collisions with walls and obstacles		•	•	•	•	•
Fitness function	A	B	B	C	C	C
Settings						
Epoch termination generation	30	60	90	150	90	150
Obstacles present		•	•	•	•	•
Moving target(s)			•	•	•	•
Multiple targets (20 targets)				•	•	•
Number of evaluation cycles	160	160	160	1000	160	1000
Targets collectable				•	•	•

Table 7.1 Parameters and settings for the six epochs of the 2D navigator GP system.

Epoch sequence #1 is used in the exaptive configuration. Sequence #2 is used in the non-exaptive configuration. The latter's runs are split into two epochs merely as a simple way to keep the total evaluation time and number of generations equal to those of sequence #1. Both sequences involve 90 generations at an evaluation time of 160 cycles followed by 60 generations at an evaluation time of 1000 cycles. Both epochs of sequence #2, as well as the final epoch of sequence #1, use the same fitness function (function C,

explained below) so the mean final best fitness values from the two configurations can be compared to one another.

Tracked Aspects of Performance

This section describes the four performance aspects that are measured during evaluation in various epochs (see Table 7.1, above). These measurement values are used to assign a fitness value to the individual being evaluated.

Distance from target(s)

At all times, when at least one target is present, one of the available targets is designated as the *current target*. This determination is not made by the navigator, but follows a pre-determined sequence. With each evaluation cycle, the Euclidean distance from the navigator to this target is added to a sum (the *target distance sum*). The total number of such measurements is incremented as well (the *target distance count*). This allows the fitness function to determine the average distance of the navigator from its target, over an entire evaluation.

During epochs involving multiple targets, in which a target is collected, brought to the depot, and then a new target is sought for collection, dropping off a collected target at the depot results in an update of a second target distance statistic. This is called the *multiple target-closeness measure*. It begins at 1.0, and with each new target collected, is multiplied by the following amount: $((S+1) / \max(1,N)) / 400.0$. S is the *target distance sum* described above, and N is the *target distance count*. Once the *multiple target-*

closeness measure is updated in this way, the values of S and N are reset to zero. This encourages both the collection of many targets and a speedy approach to each target.

During epochs in which targets are being collected, a target is automatically collected upon contact with the navigator provided that the navigator is not already carrying a target that has yet to be dropped off at the depot. Contact with any target will do; it does not necessarily have to be the *current target*.

Distance from depot

The distance from the navigator to the depot is monitored in much the same way as the distance from the navigator to the targets, described above. Once the navigator has collected a target, its goal is to bring that target to the depot. For this reason, measurements of the distance to the depot are only taken during those evaluation cycles in which the navigator possesses a target that has yet to be dropped off. The relevant values are the *depot distance sum* and the *depot distance count*. A *depot-closeness measure* is also adjusted in the same manner as the *multiple target-closeness measure*, described above for target distances. The measure is adjusted whenever a new target has been brought to the depot and dropped off.

During epochs in which targets are being collected, dropping off a target at the depot is done automatically when the navigator comes into contact with the depot, provided that the navigator is carrying a target to be dropped off.

Maximum net distance traveled

During epochs in which this statistic is needed, the GP system computes the distance between the navigator and its original starting location in the environment. The maximum observed value of this measurement, called the *maximum distance reached*, is available to the fitness function.

Number of collisions with walls and obstacles

During epochs in which this statistic is needed, a check is made during each evaluation cycle to determine whether the navigator is in collision with either an environment-bounding wall or an obstacle. If so, the *collision sum* is incremented. In either case the *collision check count* is also incremented. These two values allow the fitness function to determine the amount of time spent in collision with objects as a fraction of the total time spent in the environment.

Fitness Functions

There are three fitness functions used in the six epochs of the GP system. They are fitness functions *A*, *B*, and *C*, with *C* representing the most complex learning task, *A* being the simplest, and *B* being intermediate. They are each described below.

Fitness Function A

Fitness function A is

$$\frac{T_s}{\max(1, T_c)} - \frac{D_m}{4} \quad (7.1.1.2-1)$$

where T_s is the *target distance sum*, T_c is the *target distance count*, and D_m is the *maximum distance reached*. Each of these is described above. This fitness function is used in the first epoch of the exaptive configuration (sequence #1). It rewards the navigator both for approaching a target object and for simply traveling a large distance from its starting point.

Fitness Function B

Fitness function B is

$$\left(0.7 + 0.3 \left(1 - \frac{C_s}{\max(1, C_c)}\right)\right) \left(\frac{T_s}{\max(1, T_c)} - \frac{D_m}{4}\right) \quad (7.1.1.2-2)$$

where C_s is the *collision sum* and C_c is the *collision check count*. This is an augmented version of fitness function A that takes into account the navigator's collisions with walls and obstacles. Up to 30% of the fitness value can be lost due to collisions.

Fitness Function C

Fitness function C is

$$100000 \left(0.7 + 0.3 \left(1 - \frac{C_s}{\max(1, C_c)}\right)\right) \left(\frac{T_m(T_s + 1)}{400(\max(1, T_c))} + \frac{P_m(P_s + 1)}{400(\max(1, P_c))}\right) \quad (7.1.1.2-3)$$

where T_m is the *multiple target-closeness measure*, P_m is the *depot-closeness measure*, P_s is the *depot distance sum*, and P_c is the *depot distance count*. The final bracketed expression isn't simply $(T_m + P_m)$ as this would not take into account the very last target being pursued or carried at the moment the evaluation ends. Multiplying by 100000 reduces the frequency of fitness values that begin with leading zeros after the decimal,

simply for aesthetics and readability when viewing status updates displayed in a console window while the program executes.

This fitness function rewards the navigator for avoiding collisions with walls and obstacles, and for quickly collecting a target, bringing it to the depot, and then repeating that process as many times as possible before the evaluation time expires. The experiments and data from this chapter will test the idea that fitness functions A and B can beneficially preadapt the population for learning the task implicit in C. This is done by comparing mean best fitness outcomes for runs using sequence #1 and runs using sequence #2.

The GP System

The settings for the 2D navigator GP system are listed in Table 7.2, below.

Setting, Parameter, or Option	Value
Function set	<p>{<i>FA</i>, <i>LA</i>, <i>RA</i>, <i>TDR</i>, <i>TDS</i>, <i>DDR</i>, <i>DDS</i>, <i>S</i>, <i>V</i>, <i>ERC</i>, <i>TAR</i>, <i>SS</i>, <i>SV</i>, +, -, *, /, <i>SQR</i>, <i>SQRT</i>, <i>IF</i>, <i>NEG</i>}</p> <p><i>FA</i>, <i>LA</i>, and <i>RA</i> are front, left, and right antenna sensor values (see section 7.1.1.3).</p> <p><i>TDR</i> and <i>TDS</i> are target direction and target distance tracker values.</p> <p><i>DDR</i> and <i>DDS</i> are depot direction and depot distance tracker values.</p> <p><i>S</i> retrieves the current steering value.</p> <p><i>V</i> retrieves the current velocity value.</p> <p><i>ERC</i> is a so-called “ephemeral” random constant, in the range [-1.0..+1.0].</p> <p><i>TAR</i> returns 1 when the navigator is carrying a target, 0 otherwise.</p> <p><i>SS</i> takes a single argument and uses it to set the steering value, returning the new steering value.</p> <p><i>SV</i> takes a single argument and uses it to set the velocity value, returning the new velocity value.</p> <p>+ is a bounded addition operator returning $\min(1, \max(-1, a+b))$ where <i>a</i> and <i>b</i> are its two arguments.</p> <p>- is a bounded subtraction operator.</p> <p>* computes the product of its two arguments.</p> <p>/ is a bounded division operator that increases the absolute value of the divisor to 0.00001 if it is smaller, and bounds the result in the range [-</p>

	<p>1.0..+1.0].</p> <p>SQR returns a value carrying the same sign as its input, but equal in magnitude to its square.</p> <p>SQRT returns a value carrying the same sign as its input, but equal in magnitude to its square root.</p> <p>IF takes three arguments. It evaluates the first, and, if the result is negative, evaluates and returns the third, else the second.</p> <p>NEG flips the sign on a single argument.</p> <p>All functions take inputs in the range [-1.0..+1.0] and return values in the same range.</p>
Halting condition	150 generations
Population size	50
Elitism	1 elite individual
Duplicates	Duplicates are permitted
Variation operators	<p>50% crossover, 50% mutation</p> <p>Half of all crossovers use the regular GP crossover; the rest use a special crossover that operates like that of the special crossover operator in 3DVCE (see section B.6.1.2.3).</p> <p>Half of all mutations use regular GP mutation, with half of the remaining using a node-lowering mutation that picks a node and makes it a child of its randomly-chosen replacement, and half using a node-raising mutation that picks a node and uses it to replace its parent in the tree.</p> <p>All variation operators are designed to limit the resulting tree size. The chosen bound is 1000 nodes.</p>
Selection	Tournaments of size 4
Population initialization	<p>A tree in the initial population has its size chosen to be from 1 to 100 nodes. The growth method begins with a single node and repeatedly calls a leaf-expansion method until the desired size is reached or exceeded. If exceeded, repeated calls to a leaf-raising method are made until the tree is less than or equal to the desired size. This grow-then-shrink method is repeated a maximum of 10 times in an attempt to reach the desired size.</p> <p>The resulting tree is no bigger than the chosen size.</p>

Table 7.2 Settings for the 2D navigator GP system.

7.1.1.3 The Navigator

The navigator is a triangle that is moved around the 2D space by having its control system directly set the linear and angular components of its motion at regular intervals.

The magnitudes and directions of those movements are determined by the navigator's GP tree. The GP tree executes once during each evaluation cycle. Each evaluation cycle involves four steps.

1. The navigator executes its GP tree.

This may result in the linear and angular components of the triangle's motion being set one or more times. Only the final values set will be used.

2. The physics simulation is advanced by 0.12 seconds.

3. All forces on the navigator's triangle are zeroed.

4. The physics simulation is advanced by 0.28 seconds.

In total, each cycle advances the simulation by two fifths of a second. The physics engine used is the open-source "Chipmunk" engine [Lembcke, 2007].

Sensors, Trackers, and Effectors

The navigator possesses several sensors, trackers, and effectors. Three sensors, called the left antenna sensor, the right antenna sensor, and the front antenna sensor, report on the presence of obstacles or walls within a small radius around the tips of three antennas that protrude from the left, right, and front vertices of the navigator's triangle. These can be seen as three red circles in Figure 7.1, above. Neither the antennas nor the circles representing their zones of sensitivity are objects that collide with other objects in the simulation. The targets, walls, obstacles, and depot can pass through them. Each antenna sensor, when queried, returns a value between 0.0 and 1.0. A value of 0.0 indicates that no wall or obstacle is within a distance of r from the tip of the antenna, where r is the radius of its zone of sensitivity. A value of 1.0 indicates that a wall or obstacle is touching the tip of the antenna. Values between 0.0 and 1.0 reflect different degrees of proximity to an obstacle or wall. For example, a sensor reading of 0.3 indicates that a

wall or obstacle is present at distance $0.7r$ from the tip of the antenna. The **FA**, **LA**, and **RA** GP nodes correspond to the front, left, and right antenna sensors, respectively.

There are two trackers that provide information on the position of the *current target* relative to the position of the navigator. These are accessed through the **TDR** and **TDS** GP nodes. The **TDR** node returns a reading from the target direction tracker. The reading is 0.0 if the target is directly ahead of the navigator, with negative and positive values indicating that the target is to the right or left. The reading is in the range [-1.0..+1.0] with both extremes indicating that the target is directly behind the navigator. Readings of -0.5 or +0.5 indicating that the target is 90 degrees to one side. The **TDS** node returns a reading from the target distance tracker. The distance tracker returns a value equal to

$$\left(1 - \left(\frac{3500}{d^2 + 3500}\right)\right)^2 \quad (7.2.1.3-1)$$

where d is the Euclidean distance between the navigator and the target. The value is close to zero when the target is near and close to one when far. The value 3500 is chosen to give a wide range of tracker readings within the available space of the navigator's 2D environment.

There are two trackers that provide information on the position of the depot relative to the position of the navigator. These are accessed through the **DDR** and **DDS** nodes. Both operate in the same manner as their target tracker counterparts.

The navigator has only two effectors at its disposal. The first is controlled using the **SS** node. This node takes a single argument in the range [-1.0..+1.0] which is then used

directly to set the angular velocity of the navigator. The second of the two effectors is controlled using the *SV* node. This node takes a single argument in the range [-1.0..+1.0]. The linear velocity of the navigator is then reduced by 20%, and a new velocity vector is added to it, with magnitude proportional to the *SV* node's input, and directed in the navigator's forward direction (negative values cause the vector to point in the backward direction). The resulting velocity, if it exceeds a fixed maximum magnitude, is scaled down to that magnitude.

7.1.1.4 Early Termination of Evaluations

In order to cut down on total evaluation time, individuals that do not seem promising are terminated early before their allotted evaluation cycles have expired. Checks for early termination occur at the one-quarter, two-quarter, and three-quarter marks in an evaluation. At the time of a termination check, the individual's fitness is compared to those of the individuals in the previous generation as they were at the time of same check point. If the individual's fitness falls below the median, its evaluation terminates immediately. These checks only take effect after the third generation in an epoch.

7.1.2 Results

Fitness function C (7.1.1.2-3) returns values that range over many orders of magnitude. For this reason, computing its mean over many independent runs produces a result that is dominated by the worst values and does not reflect any central tendency in the data. To compensate, each best-of-generation fitness value, f , recorded from function C was

converted to $f' = \log_{10}(f)$. This value, called the *log-fitness*, was used for all graphs and statistical calculations in this section.

Figure 7.2, below, shows the mean fitness trajectories for the non-exaptive and exaptive configurations of the 2D navigator GP system. Because they use fitness functions A and B, the first three epochs of the exaptive configuration have the means of their raw fitness values calculated and plotted against the primary axis on the left of the figure. Because they use fitness function C, the final epoch of the exaptive configuration and both of the non-exaptive configuration's epochs have the means of their log-fitness values calculated and plotted against the secondary axis on the right side of the figure.

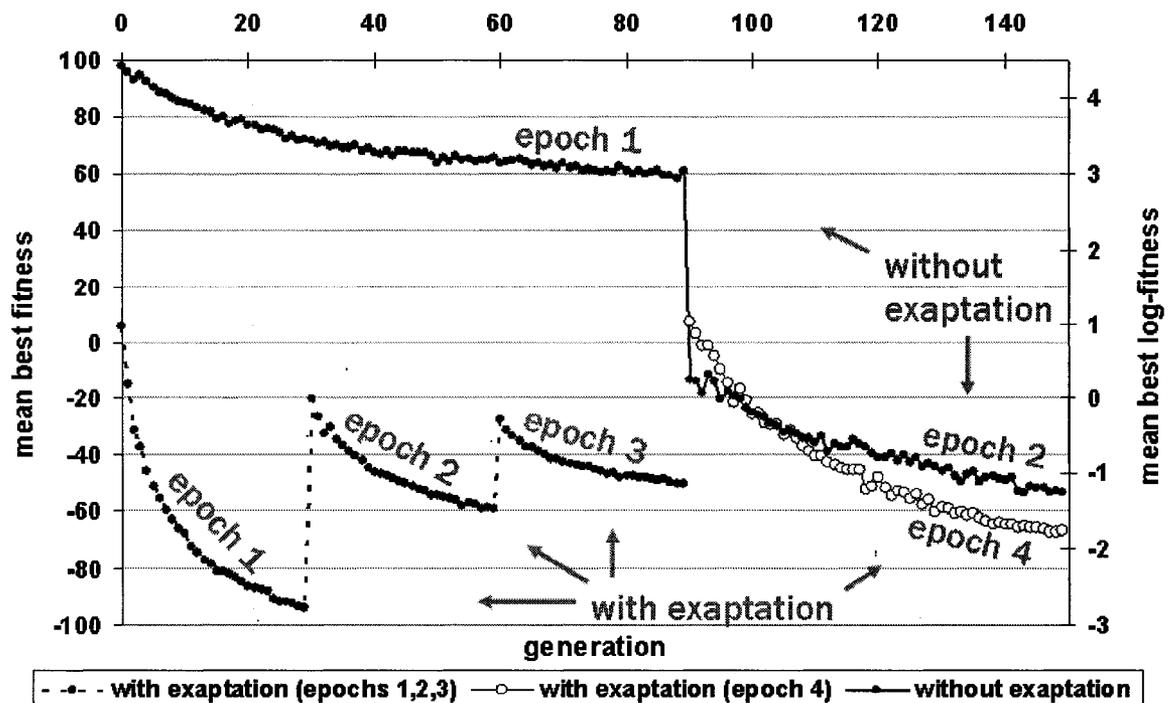


Figure 7.2 Mean best fitness and log-fitness vs. generation in the 2D navigator GP system.

The transitions from one epoch to the next are easy to discern in the figure at generations 30, 60, and 90. Since epochs 2 and 3 of the exaptive configuration both use fitness

function B, the abrupt change at generation 60 is due entirely to changes in the navigator's environment.

Table 7.3, below, gives the final mean best log-fitness statistics for both configurations.

Configuration	Mean best log-fitness	Standard deviation	95% confidence interval	Samples
Without exaptation	-1.24	2.24	± 0.26	283
With exaptation	-1.75	2.06	± 0.21	347

Table 7.3 Best fitness statistics for the 2D navigator GP system.

The same data are shown graphically in Figure 7.3, below.

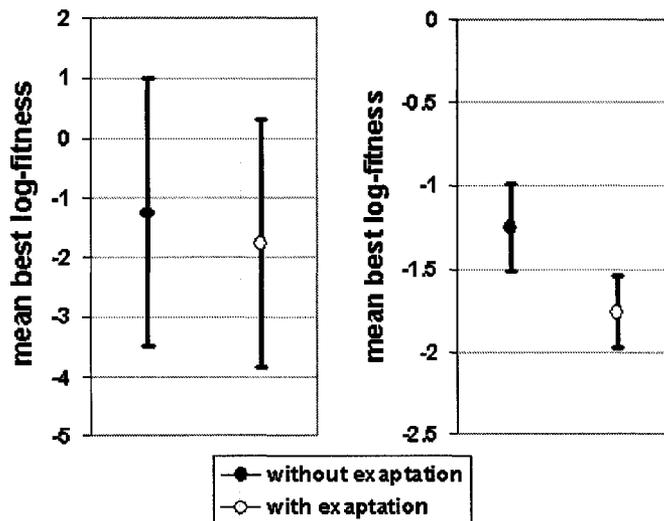


Figure 7.3 Mean best log-fitness values for the 2D navigator GP system, with standard deviations (left side) and 95% confidence intervals (right side).

Both configurations show roughly the same variability; neither is considerably more robust or consistent than the other. Enough independent runs were gathered to attain a good degree of statistical power in separating the two means. A Mann-Whitney U test comparing the best log-fitness values⁹ from both configurations gives a strong p value of 0.004942.

⁹ The Mann-Whitney U test is based on ranks and not on raw data values. Therefore,

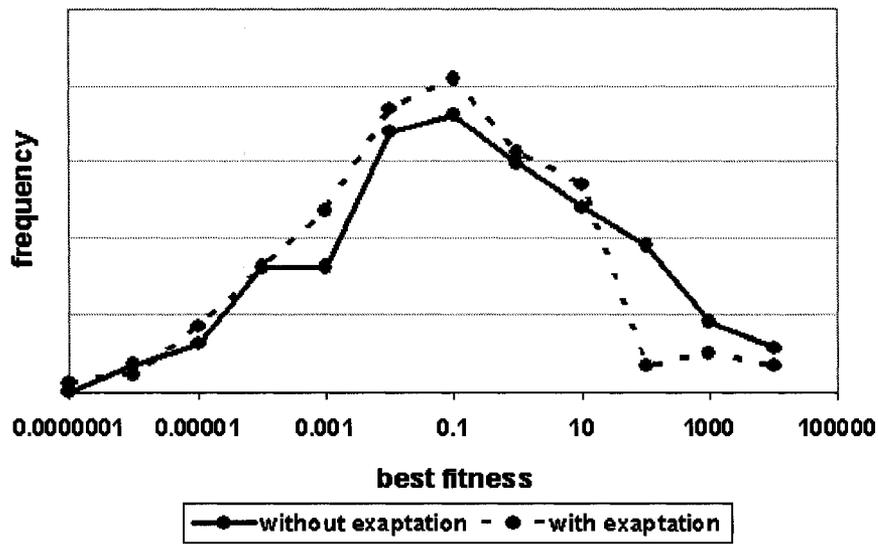


Figure 7.4 Best fitness outcome distributions for the 2D navigator GP system.

Figure 7.4, above, shows that the best fitness outcome distributions are roughly the same when considering the frequency of highly fit solutions (at left in the graph). The main differences show themselves in the mid-range and low-end solution frequencies. The exaptive configuration produces more of the former and many fewer of the latter. The distributions also show that the decision to work with log-fitness values was appropriate. Taking the mean of log-fitness values (both of which fell between -1 and -2) does meaningfully convey the central tendency in the data¹⁰.

whether this test is performed on fitness values or log-fitness values makes no difference.
¹⁰ The means computed using raw fitness values both fell between 100 and 200. Both raw fitness means are strongly influenced by the chance appearances of rare large-magnitude outcomes in each distribution at the far right tails and are not especially meaningful.

7.2 Version Two

7.2.1 Description

The second version of the 2D navigator reflects an attempt to get around problems that arose in the first version, and to amplify the effect of exaptation on the learning of a complex task. Version one's most significant departure from an ideal experimental outcome stems from the ability of the GP system to frequently and successfully learn the navigation task even without the use of preadapting epochs. Ideally, one would like to demonstrate the benefit of an exaptive approach in the domain of a learning task in which the non-exaptive approach fares much more poorly.

A number of modifications were made in version two and are described in the sections below. The two greatest enhancements, in terms of rendering the learning task more difficult for the non-exaptive GP system, are additions of two crucial new responsibilities for the navigator. Firstly, in version one, the trigger allowing the navigator's trackers to track a new target after a carried target had been brought to the depot was handled automatically when the depot and navigator came in contact. Version two moves the burden of this responsibility onto the navigator itself, which must determine on its own when to focus its trackers on a new target. Secondly, in version one, the navigator carried a target to the depot simply by touching the target and then traveling to the depot. Version two requires the navigator to 'physically' push the target to the depot.

7.2.1.1 The Environment

The environment encountered by the navigator in version two differs from that of version one in the following ways.

- **Open arena**

Version one kept the navigator and its targets bound within a rectangular region using four impenetrable bounding walls. These walls are removed in version two, granting the navigator the ability to wander unhindered in any direction. This makes the learning task more difficult since the navigator is not constrained to remain near the targets with which it's meant to interact.

- **Dispersed targets**

The twenty targets are distributed, more or less evenly, throughout the on-screen visible portion of the open arena. Some results from version one showed that navigators were able to take advantage of the closely clustered targets, picking up many of them without using trackers to locate them.

- **Multiple depots**

There are twenty depots, one for each target, also distributed more or less evenly around the arena. Target-depot pairings are chosen at random before each generation begins. This makes the learning task more difficult since each target's position relative to its depot will rarely be the same twice in a row. Relative positions of targets and depots, therefore, cannot be taken advantage of in the manner allowed by version one.

- **No obstacles**

The obstacles that were present in version one have been removed. While this makes the problem somewhat easier, it may be partly offset by the fact that the GP nodes for

antenna sensor inputs, still in the available function set, are now useless, requiring that the GP system learn to ignore them.

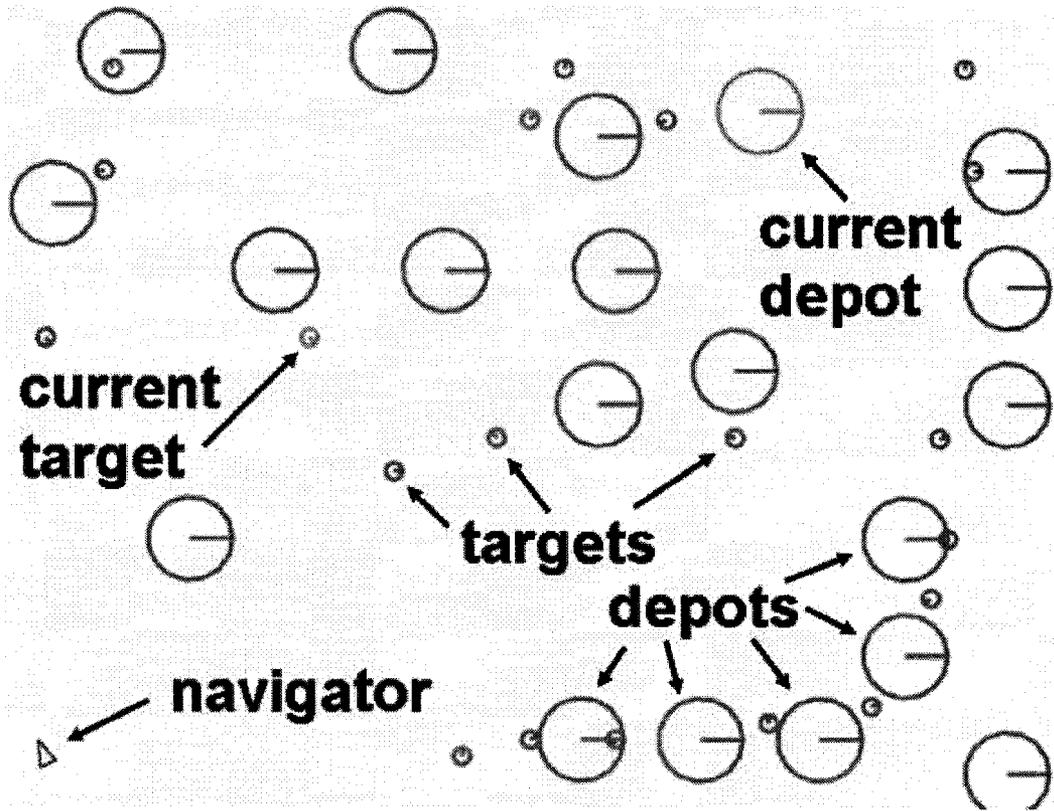


Figure 7.5 GP Navigator environment and objects (version 2).

7.2.1.2 The Learning Tasks

The navigator's task in the environment depends on the epoch in which it finds itself.

There are three different epochs divided into two sequences, one exaptive, the other not.

The final learning task in each sequence is to maximize the number of targets that are inside their depots at the end of the evaluation time. The parameters and settings of each epoch are shown in Table 7.4, below, and subsequently explained.

	Sequence #1 Epoch 1	Sequence #1 Epoch 2	Sequence #2 Epoch 1
Tracked aspects of performance			
Timing of first attempted target/depot switch	•		
Total navigator movement	•		
Distance of navigator from target	•		
Total target movement	•		
Distance of target from depot	•		
Fitness function	A	B	B
Settings			
Generations	80	71	79
Targets	1	20	20
Depots	1	20	20
Evaluation cycles per individual	300	3000	3000
Evaluation cycles per epoch	1.2×10^6	1.065×10^7	1.185×10^7

Table 7.4 Parameters and settings for the three epochs of the 2D GP navigator (version two).

Sequence #1 constitutes the exaptive approach while sequence #2 corresponds to a more standard non-exaptive approach. Both sequences involve the same total amount of simulation time (1.185×10^7 cycles). Sequence #1 begins with a preadapting epoch using a fitness function that rewards intermediate work in bringing targets to their depots, using only a single target-depot pair and one tenth of the normal evaluation time. The second epoch involves the same fitness function as is used in the non-exaptive sequence #2. This fitness function rewards navigators based entirely on the number of targets that are either inside their depots or in contact with their depots upon completion of the evaluation time.

Tracked Aspects of Performance

This section describes the five performance aspects that are measured during evaluation in the preadapting epoch of sequence #1 (see Table 7.4, above). These measurement values are used to assign a fitness value to the individual being evaluated. The fitness function of sequence #2 and that of the second epoch in sequence #1 do not involve any aspects of navigator performance that are measured or tallied during evaluation. They make a single measurement – the number of targets either inside or touching their depots once evaluation time has expired.

Timing of First Attempted Target/Depot Switch

Since the preadapting epoch involves only a single target-depot pair, using the *SW* node (explained below) to divert the navigator's trackers to a new pair has no effect.

Nevertheless, in order to preadapt the population for the more complex environment of the second epoch, the use of the *SW* node for making such tracker switches is monitored and factored into the fitness function (see *fitness function A*, below). The *SW* node switches the trackers to a new target-depot pair whenever the last *SW* call in a GP tree¹¹ receives an argument with a value greater than 0.5. At the moment such a switch is first made during evaluation, the current evaluation cycle number (between 1 and 300) is recorded for use by the fitness function. Navigators that wait longer before triggering the switch receive a greater reward.

¹¹ A GP tree may include many instances of the *SW* node. Only the last of these to be executed is used to determine whether or not the sensors will switch to a new target-depot pair.

Total Navigator Movement

The preadapting epoch divides fitness values into four tiers. The lowest of these rewards navigators, at a bare minimum, for simply moving. Total navigator movement during evaluation is tracked by computing the Euclidean distance between a navigator's current position and its position in the previous evaluation cycle. These piecewise distance measurements are summed over all 300 evaluation cycles.

Distance of Navigator from Target

The second tier rewards navigators not just for moving, but for moving closer to the target. To that end, the distance between the navigator and the target is measured before the evaluation begins and after it ends. The difference can be used to determine whether the navigator's net movement was toward the target or away from it, as well as to determine how much closer or farther it moved.

Total Target Movement

The third tier is for navigators that move close enough to the target to make contact with it. Navigators in this tier are rewarded for simply pushing the target around. Like *total navigator movement*, above, the distance between the target's current position and its position in the previous cycle is calculated during each cycle and summed over the entire evaluation time of 300 cycles.

Distance of Target from Depot

The fourth and final tier rewards navigators that reach the target, push it around, and move it closer to the depot. The distance between the target and the depot is measured

and recorded both before the evaluation begins and after it finishes. The difference between the two distances provides the information needed to determine whether a navigator has reached this performance tier and to determine how well it performed. In addition to these two measurements, the distance of the target from the depot is also recorded at the moment an *SW* node is first used to switch the navigator's trackers to a new target-depot pair. This allows the fitness function to reward the navigator based on the timing of its use of the *SW* node with respect to the final crucial step in its task – pushing the target to its depot – alongside the usual reward based on the timing relative to the overall evaluation time.

Fitness Functions

Version two of the 2D GP Navigator uses two fitness functions. Both are described below. Only fitness function A makes use of the five performance measures described above. Fitness function B, the more difficult of the two learning tasks, measures only the number of targets either in contact with or completely inside their depots at the end of an evaluation.

Fitness Function A

Fitness function A's return values are divided into four performance tiers, each separated by one million points to prevent overlap. The first tier rewards navigators simply for moving. The second tier rewards navigators for moving closer to the current target. The third tier rewards navigators for pushing the target. The final tier rewards navigators for pushing the target closer to the depot. Each tier also takes into account the behavior of the navigator with respect to its use of the *SW* node. The *SW* node is an addition to the

function set in version two of the GP navigator. It takes a single argument and causes the navigator's trackers to track a new target-depot pair if the argument's value is greater than 0.5. Fitness function A is evaluated in a context with only a single target and a single depot in the navigator's environment. For that reason, the use of the *SW* node can be factored into the fitness function without the severe negative consequences that its use or disuse can produce in an environment with multiple targets and depots.¹²

¹² In the environment containing twenty targets and twenty depots, using the *SW* node to trigger a switch to a new target-depot pair can result in the navigator abandoning any partial work it might have done on its current target-depot pair. Many randomly-generated individuals trigger such switches often, resulting not only in little to no work being done in bringing targets closer to their depots, but also resulting in erratic 'random' movement for the duration of evaluation time. By contrast, navigators that never trigger such a switch cannot expect to bring more than a single target to its depot. To bring all twenty targets to their depots a navigator must time the use of the *SW* node carefully.

Pseudo-code for fitness function A is given, below.

```

fitness_A()
{
    // Timing value used to penalize early use of the SW node
    timing = 1
    if (navigator switched target-depot pair)
    {
        delta = cycle number of first target-depot switch
        timing = 9 * delta / (total evaluation time)
    }

    // Tier 4: target was pushed closer to depot
    if (target was pushed closer to depot)
    {
        if (SW node was activated at all)
        {
            d1 = initial distance of target from depot
            d2 = distance when SW was first activated
            boost = 100 - (10 * d2/d1)
        }
        else boost = 10
        d = initial distance of target from depot - final distance
        return( 1000000 - (d * timing * boost) )
    }

    // Tier 3: target was pushed
    if (total target movement > 5.0)
    {
        return( 2000000 - (total target movement * timing) )
    }

    // Tier 2: navigator moved closer to target
    d = initial distance of navigator from target - final distance
    if (d > 5.0)
    {
        return( 3000000 - (d * timing) )
    }

    // Tier 1: navigator moved
    return( 4000000 - (total navigator movement * timing) )
}

```

By dividing fitness values into multiple non-overlapping tiers, fitness function A allows the population to learn its task in stages and at its own pace. Individuals scoring in the higher tiers will win out in selection tournaments against individuals in lower tiers, but will defeat individuals in the same tier only when outperforming them at the part of the task evaluated in that tier.

Fitness Function B

Fitness function B is a straightforward count of the number of targets that are either within or touching their respective depots after the final evaluation cycle has completed.

This function provides *very* impoverished performance feedback to the evolutionary process, and provides no reward for any of the intermediate work required to bring a target to its depot. Its value is

$$\frac{1}{T + 1} \quad (7.2.1.2-1)$$

where T is the number of targets that are inside or touching their assigned depots at the end of an evaluation.

The GP System

The settings for the 2nd version of the 2D navigator GP system are in Table 7.5, below.

Setting, Parameter, or Option	Value
Function set	{ <i>FA, LA, RA, TDR, TDS, DDR, DDS, S, V, ERC, TAR, SS, SV, +, -, *, /, SQR, SQRT, IF, NEG, SW</i> } This function set is identical to that of version one in Table 7.2 but for the addition of <i>SW</i> . <i>SW</i> takes a single argument. If the argument's value is greater than 0.5 in the last <i>SW</i> call made in the GP tree, the navigator's trackers begin tracking a new target-depot pair. Because there are no walls or obstacles, and because targets are pushed instead of carried, nodes <i>FA, LA, RA, and TAR</i> are effectively mere constants.
Halting condition	151 generations (sequence #1) 79 generations (sequence #2)
Population size	50
Elitism	1 elite individual
Duplicates	Duplicates are permitted
Variation operators	50% crossover, 50% mutation The crossover and mutation operators and constraints are identical to those of version one, described in Table 7.2.
Selection	Tournaments of size 4
Population initialization	Population initialization is performed in a manner identical to that of version one, described in Table 7.2.

Table 7.5 Settings for the 2D Navigator GP system, version two.

7.2.1.3 The Navigator

The navigator in version two is identical to the navigator in version one but for two minor changes. The first is the addition of the SW node, described above, which can be thought of as affecting the control system as opposed to the navigator itself. The second is that the circles drawn in the display to represent the antenna sensors are not shown in version two. This is a change in the visual representation of the navigator only. The circles are not shown because there are no obstacles or walls with which the antenna sensors can interact. These sensors are still present, but inputs from them remain constant.

7.2.1.4 Early Termination of Evaluations

Early termination of evaluations was not used in version two of the 2D GP Navigator.

7.2.2 Results

Since fitness function B (7.2.1.2-1) uses inversion to convert a maximization problem into a minimization problem. All logged fitness values from epochs using fitness function B have converted $f=1/(T+1)$ to $T=(1/f)-1$ in order to compute and record T , the number of targets brought to their depots. This will be referred to as the *converted fitness*.

Figure 7.6, below, shows the mean best converted fitness versus evaluation time for both the exaptive and non-exaptive epoch sequences. Values based on fitness function A in the initial preadapting epoch of the exaptive sequence have been omitted from the graph, leaving only two trajectories based on fitness function B.

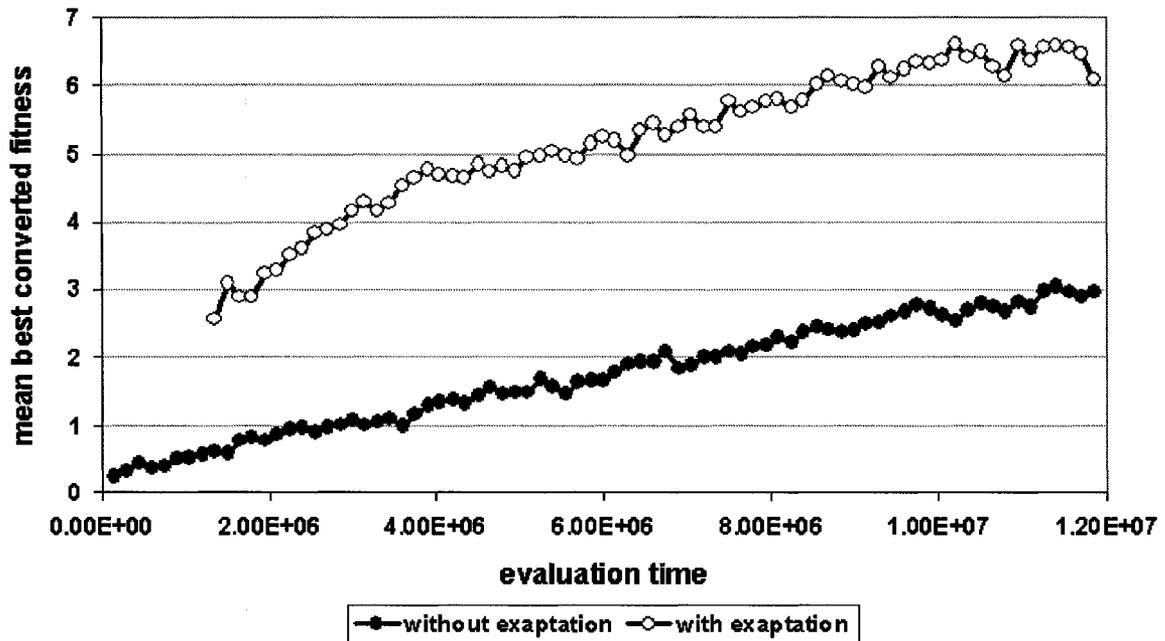


Figure 7.6 Mean best converted fitness trajectories for the 2D GP Navigator, version two.

The best fitness statistics for the final generation in each epoch sequence are given in Table 7.6, below.

Configuration	Mean best converted fitness	Standard deviation	95% confidence interval	Samples
Without exaptation	2.96	2.84	± 0.63	78
With exaptation	6.10	3.62	± 0.98	52

Table 7.6 Best fitness statistics for the 2D GP Navigator, version two.

The data from the table are plotted graphically in Figure 7.7, below.

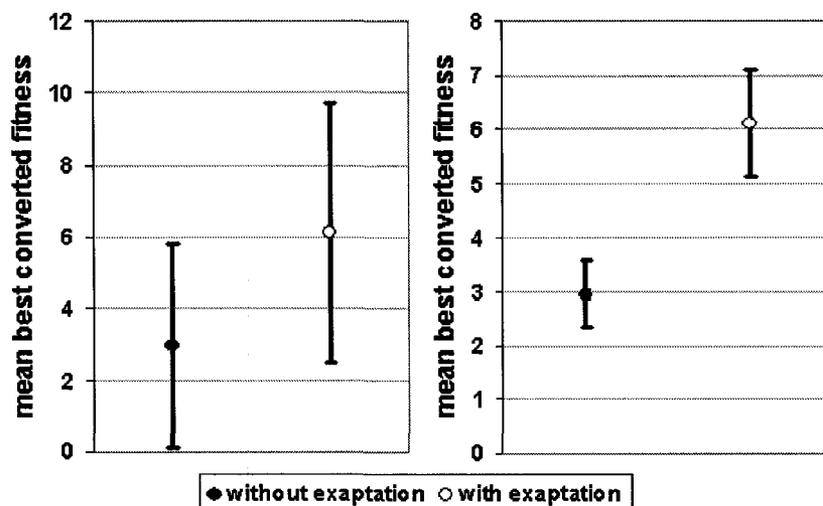


Figure 7.7 Mean best converted fitness values for the 2D GP Navigator, version two, with standard deviations (left side) and 95% confidence intervals (right side).

A Mann-Whitney U-test comparison gives $p \leq 0.0001$. Both the exaptive and non-exaptive configurations have a comparable amount of variation in their outcomes, as shown by the standard deviations. The overlap in outcome distributions, however, is less extreme than that of version one, showing a greater benefit for the exaptive approach.

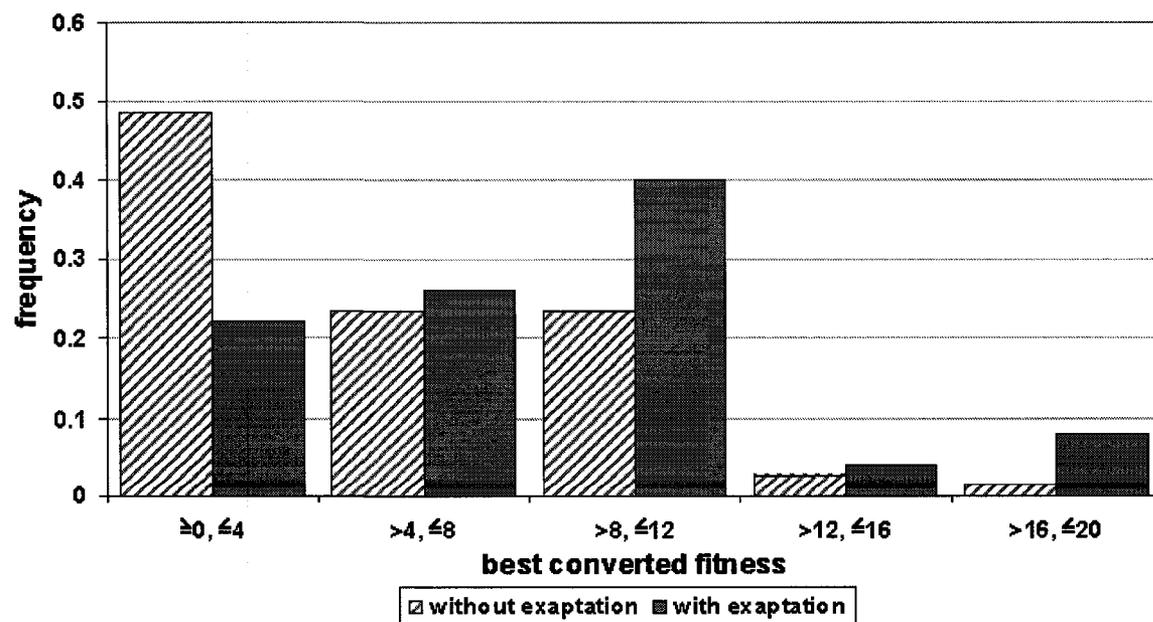


Figure 7.8 Best converted fitness outcome distributions for the 2D GP Navigator, version two.

Figure 7.8, above, shows how the best converted fitness outcomes were spread across the possible range of zero to twenty targets brought to their depots. As was the case with version one, the most obvious difference between the exaptive and non-exaptive approaches is in the frequencies for the poorest outcomes, with the exaptive approach being only about half as likely as the non-exaptive approach to produce navigator control systems that perform extremely poorly. The balance of outcomes shows the exaptive approach exceeding the non-exaptive approach in outcome frequencies, to greater or lesser degrees, in all other bins in the graph.

7.3 Conclusions

The 2D navigator GP systems provided examples of complex task learning with and without prior training on simplified versions of the learning task or rewards for intermediate steps. The exaptive approaches, which added new complexity to the desired task with each epoch or tier (approaching a target, avoiding collisions, pushing a target, bringing a target to the depot, and so on), successfully outperformed otherwise identical non-exaptive GP systems. In the case of version one, the benefit was only moderate in magnitude, whereas the improvement in version two was more substantial.

Several unexpected aspects of the behavior of the 2D navigator systems seem to be evident in the performance data and in recorded animations of the navigators in action. In version one, the non-exaptive configuration was able to produce highly fit solutions. At the outset this had seemed unlikely given the complexity of the task. Observations of the behavior of fit individuals from version one, such as the navigator shown in the video at [Graham, 2008b], indicate that the placement of the twenty targets in the environment,

which can be seen in the snapshot of Figure 7.1, allow for the collection of many targets without necessarily requiring, on the part of the navigator, the ability to sense them. As many of the targets remain just below the depot, a navigator can expect to collect many of them and bring them to the depot simply by weaving about within a small area of the 2D environment. Enough collisions with targets and with the depot can be expected that many targets will be collected, even if none are designated as the *current target* and tracked by the trackers while being collected.

This behavior was evident in both the exaptive configuration and the non-exaptive configuration in version one. The exaptive configuration, despite taking advantage of the same weaknesses in experiment design as did the non-exaptive configuration, was able to perform this task more efficiently on average. The outcome distributions in Figure 7.4 show that the boosted advantage stems less from producing a greater number of the most highly fit solutions, and more from producing considerably less of the very poorest solutions. In other words, the exaptive approach is less likely to converge to a very weak solution, with its epochs of preadaptation acting as a safeguard against such outcomes.

One other noteworthy aspect of the performance data can be seen in Figure 7.2 where, in the final 60 generations, the performance of the exaptive configuration is initially worse than that of the non-exaptive configuration, but quickly overtakes it. This seems to support the idea that the earlier epochs endow the population with beneficial building blocks that can be harnessed in the final epoch. So, although the exaptive population enters the final epoch with poorer performance, on average, than a regular GP system, it possesses a store of genetic material that allows it progress more quickly.

The results from version two of the 2D GP Navigator tell a similar story, but with an even more exaggerated benefit for the exaptive approach. The task was made more difficult than that of version one in a variety of ways and this seems to have helped in highlighting the strength of exaptation. What's perhaps most remarkable about version two is the ability of the non-exaptive GP system to tackle the problem at all. Given the extremely crude and impoverished feedback provided by the fitness function (there are only 21 possible fitness values), it was initially thought that the non-exaptive approach would make no progress whatsoever, making its successful runs, despite being far less frequent than those of the exaptive approach, somewhat surprising.

Chapter 8 Exaptation Experiments with 3D Virtual Creatures

8.1 General Description

As exaptations in biology occur within a framework of morphology and behavior for embodied autonomous creatures in a physical environment, one candidate realm in which to experiment with exaptation in EC is in systems that are perhaps closer analogs to biology than many other EC systems (with the possible exception of ALife systems). At least in some respects, the problem of evolving morphology and control for embodied autonomous agents (often simply called creatures [Sims, 1994a]) in a simulated physical environment is such an analog. This section describes a system called 3DVCE, or 3D Virtual Creature Evolution, which was constructed to perform exaptation-related experiments involving co-evolving morphologies and control systems for simulated robots.

The section begins with an overview of the literature pertaining to evolutionary robotics, particularly those systems and projects similar to 3DVCE and based on evolving populations of virtual robots similar to those generated in the original and widely-known work of Dr. Karl Sims [Sims, 1994a & b] in the 1990's. After putting 3DVCE into this context, a brief description of the system is provided, with a more thorough and detailed description available in Appendix B. This is followed by a description of two exaptation experiments conducted using 3DVCE.

8.2 An Overview of Related Projects in Evolutionary Robotics

The virtual creature evolution work of [Sims, 1994a & b] set a trend in evolutionary robotics [Cliff, Harvey, & Husbands, 1993] that inspired the generation of many similar systems in the decade and a half since [Lipson & Pollack, 2000], [Taylor & Massey, 2001], [Miconi & Channon, 2005 & 2006], [Chaumont, Egli, & Adami, 2007], [de Jong, 2008], [Krčah, 2007], [Klein, 2003 & 2007], [Lassabe, Luga, & Duthen, 2007], [Komosinski, 2000]. Sims evolved embodied physical simulated robots for swimming, walking, and jumping, competition for resources, as well as for the ability to following a light source in the environment.

Both the control systems and the morphologies of Sims' creatures were co-evolved together in an environment governed by the rules of a rigid-body Newtonian physics engine. Creature bodies were tree structures, somewhat like L-Systems¹³ [Lindenmayer, 1968] with nodes of different types where each type has its own branching pattern, allowing for both recursive and non-recursive body structures. The control systems consisted of neural networks, with many different types of neurons that computed a variety of functions on their inputs: sum, product, divide, sum-threshold, greater-than, sign-of, min, max, abs, if, interpolate, sin, cos, atan, log, expt, sigmoid, integrate, differentiate, smooth, memory, oscillate-wave, and oscillate-saw. Both the morphologies and the control systems were represented in creature genomes by directed graphs.

¹³ An L-System is a grammar-like system of symbols, variables, and replacement rules that can generate fractal-like patterns with self-similarity properties.

Individual cuboid body parts in Sims' creatures were connected to one another by a variety of different types of joints: rigid, revolute, twist, universal, bend-twist, twist-bend, and spherical. Each joint was associated with motors that applied torques whose magnitudes were specified by the neural network control systems. Maximum motor strength was proportional to the maximum cross-sectional area of the two cuboids connected by the joint, which is similar to the way muscle strength scales in real animals. Recursion in construction of the body was terminated using a genetic "recursive limit" parameter that determined the number of times a node of a given type should repeat in recursive cycles. Node types also included "terminal-only" flags which allowed certain branches to be present only at the ends of recursive chains. These allowed hand- and foot-like structures to form at the ends of segmented limbs.

Sims' control systems received inputs from the environment and from the body through three types of sensors. The first were joint angle sensors that relayed the current angle of a joint. The second were contact sensors that would report whether or not the body part cuboid was in contact with another body part, with the ground, or with other objects in the environment. The third were photosensors that relayed information about the relative angle and intensity of the light source in the environment. All neurons were embedded within and associated with a specific body part with the exception of several creature-level neurons that were not associated with specific body parts yet were available (through interneuronal connections) to all.

The physics simulation proceeded in intervals of 1/30th seconds of simulated time. The control system was updated twice for each interval. Simulation began without friction,

and with joint motors disabled. This lasted until the limp creature's center of mass came to rest¹⁴. Once it was determined that a creature was at rest, joint motors were enabled and fitness evaluation began. This was done to prevent "cheaters" that would gain horizontal movement by virtue of being tall and therefore being able to fall a long way sideways.

This problem of tall creatures falling large distances and other similar cheating-creature problems have been a recurring theme in most of the virtual creature evolution projects since [Sims, 1994a & b]. Often the most intuitive and straightforward means of evaluating fitness opens the door to some form of undesirable cheater strategy that satisfies the fitness function while defying the intent of the programmer. Most fitness measurement techniques in such systems are made slightly more complex by subtle 'bandage' modifications to reduce the frequency of such cheaters.

Sims' GA used a population size of 300, with 1/5th, or 60-creature elitism. During population initialization, if less than 60 creatures attained non-zero fitness, initialization was restarted. The selection scheme was fitness proportional. The mutation operator used a rate that was always scaled for any given graph (either a morphology graph or a neural network graph) so that the most likely number of alterations was one. In addition to a crossover operator, there was another two-parents-to-one-child operator called "grafting". These operators were simply two different ways to hybridize directed graphs.

¹⁴ The center of mass was never perfectly at rest due to the nature of the physics engine, and would constantly experience a very minute jiggling motion. Instead, a small threshold was used to determine whether or not the creature had come to rest.

The GA and simulation ran on a 32-processor Connection Machine (CM-5), taking about three hours for a 100-generation run. The results were a number of unique successful evolved creatures, displaying a variety of strategies and behaviors which can be seen in several videos that have received, and continue to receive, a wide audience of both researchers and members of the general public on the internet.

Another virtual creature evolution project that has become very widely known, both to EC researchers and to the general public is the GOLEM project of [Lipson & Pollack, 2000]. Like Sims, Lipson and Pollack co-evolved robot morphology and behavior in simulation, but unlike Sims they used a rapid-prototyping 3D printer to construct their evolved robots in the real world.

Their robots had no sensors and relied on their control systems to produce signals that would make them move across the ground without any feedback. Limb movements were generated by linear actuators that extended and contracted. Instead of Sims' cuboids, their bodies were composed of cylindrical bars connected by ball joints. These allowed a wide range of movement types while rendering the joints printable using the 3D printer. Since the printer could only produce solid plastic parts and not electronics, the actuators were snapped in by hand when the printing was complete. In order to increase the likelihood that creatures evolved for performance in simulation would produce similar performance in the real world, the physics engine used "quasi-static motion". This meant that every frame was stable and that all movements involved low momentum.

Their GA ran for between 300 and 600 generations in the reported experiments, with a population size of 200. Instead of the typical random-genome population initialization, all

creatures in their initial generation consisted of zero bars in their bodies and zero neurons in their control systems. In most runs, the first several dozen generations would pass without any creatures producing movement since they had to build up complexity from this initial low. Their description of GOLEM included a mutation operator, but no crossover. Performance was measured by net Euclidean travel distance over a fixed simulated time.

When the real robots were evaluated, they found examples where reality did not reflect the simulation. They concluded that their friction model was not sufficiently realistic, particularly because of observations of creatures' limbs slipping on surfaces in the real world but not in simulation.

Since little had been done to replicate or extend the work of [Sims, 1994a & b] between 1994 and 2001, a very similar system was developed by [Taylor & Massey, 2001] but with several differences. First, they evolved "crawlers" only (creatures that travel across a flat level surface). Their joint motors were PD (Proportional-Derivative) actuators, or servos. These took desired orientations as inputs instead of raw torque values. They claimed that preliminary experiments showed that the servos produced useful movement more quickly than the regular motors. Their creatures' body parts were "sphyls" (cylinders capped with half-spheres) instead of cuboids. This made collision detection faster. Their bodies were more limited than Sims' at 4 to 10 sphyls per body, and their control systems were kept small as well at 5 to 8 controller parts per sphyl. They found that oscillator primitives (sine wave generators) tended to be useful in producing beneficial movement patterns, and added a bias that made creatures more likely to use

them. Their GA parameters were roughly comparable to those of Sims, but with tournament selection instead of fitness-proportional. Instead of a multi-processor CM-5, their system ran on a 400MHz Celeron, and achieved speeds slightly faster than real time.

A particularly informative aspect of Taylor and Massey's research was their discussion of many of the common pitfalls and problems that designers and users of such virtual creature evolution systems are likely to face, and how these can be handled. One example highlighted how intuition can be misleading when it comes to designing a fitness function.

“..the choice of fitness function even for seemingly straightforward behaviors is not trivial and usually requires considerable experimentation to get right. The function that successfully produces the desired behaviors can often be somewhat more complicated than might initially have been thought.. ..The general problem is, no matter what fitness function is used, there often seems to be a way for creatures to score highly on it while not performing the sort of behavior that we, as designers of the function, had hoped for.”

Another pitfall they discussed pertained to a type of run-away error condition common in physics engines, sometimes called “numerical explosions”.

“Despite various attempts to limit the magnitude of the forces applied to joints, creatures would still sometimes evolve whose movements entailed forces and velocities that were too great for the physics engine to resolve at the given size of the integration step. In these cases, the physics engine tended to accumulate numerical errors to a point where the creature unrecoverably [sic] exploded (i.e. the constraint solver failed to converge on a solution, and the integrator then generated incorrect velocities, giving the impression that the body parts had blown apart in random directions).”

To solve this problem, they proposed coding special procedures for monitoring the behavior of creatures during simulation to catch and punish numerical explosions. A commonly used method for doing this, and one that Taylor and Massey adopted, is to kill

creatures whenever any body part exceeds a predetermined velocity threshold. They suggested that stability checks of this sort are likely to be a requirement for any such evolutionary system.

Another approach to developing a virtual creature evolution system was taken by [Miconi & Channon, 2005 & 2006]. Their system for evolving “autonomous articulated structures” was very much like those of [Sims, 1994a & b] and [Taylor & Massey, 2001], but with harsh restrictions. The first restriction they imposed was the removal of oscillator primitives from the control system, calling them “high-level, ad hoc elements” representing a “..significant amount of a priori knowledge given to the system.” Such repetitive pattern-generating components are forced to evolve from scratch in their system, if at all. They described this approach as being more general than those used in previous systems. A second restriction was placed on the available types of joints. They used hinge joints only, which have just a single degree of rotational freedom. A third restriction came from the fact that, like Sims, Taylor, and Massey before them, their genetic specification was a graph of nodes, but, unlike their predecessors, they did not allow for “segmentation” (recursion) in their body structures, feeling that it wasn’t necessary to reliably get “meaningful and efficient behaviors.” A fourth restriction was an upper limit of 11 on the total number of cuboids per body.

Their evolutionary algorithm was a steady-state GA (SSGA) with a population size of 500. Selection was via tournaments of size three. In a tournament, the worst of the three randomly-chosen contestants was replaced with either a mutated version of the best of the three, or an offspring produced by either the crossover or grafting operators applied to the

remaining two contestants. Fitness was based on distance traveled along the x-axis, or, in experiments involving co-evolution, on control of a resource (a virtual cube in the environment) in one-on-one competitions like those of [Sims, 1994b].

They divided their evolved morphology and control strategies into three categories: “crawlers”, “snakes”, and “bouncers”. Crawlers used two symmetric limbs to push along the ground. Snakes undulated in a manner that produced locomotion. Bouncers used quick motions and momentum to make repeating jumps. Their most efficient creatures were bouncers. These remained in a constant state of dynamic instability by jumping. They noted a tendency for creature body plans to converge and stabilize very early on in a run. The same tendency is observed in the 3DVCE GA, described below.

In the same vein as the pitfalls covered by [Taylor & Massey, 2001], Miconi and Channon discussed two further problems, also observed in 3DVCE. The first was a rapid twitching motion they dubbed “dispendious jittering”. Some populations hit upon strategies involving tiny rapid movements that produced an overall momentum in a single direction. Creatures with dispendious jittering would travel by shaking their way across the ground. The second, dubbed “parasite movement”, involved creatures making body part motions that either did not contribute to locomotion, or, in some cases, hindered it. These parasite movements were found to decrease or disappear as the population evolved. Parasite movements were more of an observation about common evolutionary dynamics and strategies than a pitfall to be avoided.

The most recent work in this area is that of [Chaumont, Egli, & Adami, 2007]. They described the search problem of evolving 3D creature morphology and behavior as a

“high-dimensional space of almost unknown structure”, and considered the use of evolutionary techniques to be critical for solving problems of that nature. They used their systems to evolve both “walkers”, and “catapults”. Walkers were evolved for traveling across a flat terrain. Catapult creatures had one special “projectile” body part connected to the rest of the body by a unique joint which could be voluntarily or involuntarily severed. The catapults were evolved for the ability to launch their projectiles as far as possible by coordinating body movement with the severing of the special joint.

Like [Taylor & Massey, 2001], they spent considerable time describing a number of pitfalls and gave tips for avoiding them. These included numerical explosions, the evolution of “cheaters”, or undesirable strategies when using straightforward fitness measurement techniques, Miconi and Channon’s “dispendious jittering” which they describe as creatures “violently shaking”, and population convergence to very suboptimal solutions, caused by using a measure of creature size to scale distance values used in fitness evaluation. The latter was an attempt to take body size into account when measuring distances traveled. A similar scaling method is used in the 3DVCE GA, described below, but does not seem to be as problematic. Another problem described by Chaumont, Egli, and Adami is that of short evaluation times. They found that creatures would sometimes evolve movement patterns that would not persist for long beyond the regular duration time for evaluations used during evolution. They recommended longer evaluation times as a means of increasing the likelihood of producing locomotion behaviors that continue when creatures are permitted to travel indefinitely.

The GA was similar to that of [Sims, 1994a] with a population size of 300 creatures, runs of 100 generations, and 60-creature elitism. The same three variation operators were used: crossover, grafting, and mutation. They found that “realistic” walker gaits were produced, as well as a variety of projectile-throwing strategies for catapults. When the dimensions of the projectile cuboid were placed under genetic control, they even noticed the evolution of very flat projectiles that would roll for long distances after being thrown, somewhat like a wheel. They provided an interesting insight into the behavior of physics engines when they evolved a population of walkers with completely disabled controllers. Evolution was still able to make some limited progress by taking advantage of numerical instabilities in the physics engine that would cause the limp creature bodies to slowly move across the ground.

The following sections describe another system called 3DVCE. Developed as a tool for experimenting with exaptive approaches to the problems of co-evolving morphology and behavior of autonomous robots, the overall scheme resembles those of the systems described above, with one major difference – GP is used in place of neural networks for creature control systems. The evolved morphologies and behaviors, many examples of which can be seen at [Graham, 2008a], are subjectively comparable to those seen in the systems described above, and show an extremely broad range of body plans and strategies.

8.3 The 3DVCE GA

The 3DVCE system is set up to use a fairly normal GA with a constant population size, elitism, crossover operators, mutation operators, and so on. The operators themselves are

specialized for the problem and the chosen genetic representation, but otherwise, from a high level view, the GA is not especially unusual. The following sections give a brief description of the GA. A more complete description of many aspects of it, including specific physics engine settings, listings and descriptions of the GP node types available to creature control systems, details of the crossover and mutation operators, and much more is given in Appendix B.

8.3.1 Evolutions, Runs, Fitness Measurements, and Evaluations

The 3DVCE GA operates on configurations called evolutions, whose parameters are specified by the user. Each evolution consists of a number of runs. The number of runs in an evolution can be set by the user to any positive number. It can also be set as indefinite, in which case the evolution will always start a new run each time an old one finishes. Such evolutions, when executed by the GA, stop only when the user intervenes to terminate or interrupt them. Each run consists of a number of generations. This value can also be set to any positive number or be set as indefinite in which case the evolution will consist of only one run that continues producing new generations until the user intervenes. Each generation involves the production of a number of virtual creatures, determined by the population size setting. Each creature produced undergoes a fitness measurement. Each fitness measurement consists of a number of evaluations. This can be any positive number, but may not be set as indefinite. Each evaluation involves construction and placement of the creature in a simulated physical environment, followed by a fixed period of simulation time after which some aspect of creature performance is translated into a single fitness value for the evaluation. Allowing for multiple evaluations

permits the GA to take an average. This can partially compensate for the inevitable noise arising from non-determinism in the physics simulation.

8.3.2 Fitness Schedules

An evolution contains within it most of the settings that configure the GA. These include population size, tournament size, mutation rates, crossover rates, and others. They also contain fitness schedules. A fitness schedule is an ordered sequence of entries. Each entry specifies a particular fitness function that will be used during a certain stretch of generations, dividing runs into epochs. Epochs and schedule entries have a one-to-one correspondence. All generations in the same epoch use the same fitness function, and are all consecutive. All fitness schedules consist of at least one entry specifying the fitness function that takes effect at the beginning of a run (in generation zero) for the first epoch.

Each fitness function specifies the number of simulation cycles that will be used in an evaluation. Each cycle corresponds to 0.01 seconds of simulated time, plus a single creature control system update event (i.e. creatures process inputs from their environment and respond 100 times per second in the virtual world). Fitness functions also specify a number of weighting values used to combine different performance measures such as the distance traveled by the creature during evaluation time, for example. The manner in which these weights are used is described below, in the section on fitness evaluation. Fitness functions also specify one of three different simulated physical environments for creatures to inhabit.

The fitness schedule is the means by which deliberate exaptation is achieved in the GA. When the morphology and control of creatures evolved for one function are subjected to a new fitness function after one epoch ends and another begins, emphasis switches to a different aspect of their performance. Thereafter, secondary adaptations accumulate as the population adapts to its new task.

8.3.3 Producing Offspring

Offspring are produced in one of three ways in the GA.

- 1. Elitism**

The creature may be an unaltered copy of the individual with the highest fitness in the previous generation. This is single-individual elitism in the GA. The first creature in every generation is produced this way.

- 2. Crossover**

Two parents are chosen via two independent selection tournaments from among those in the previous generation. They are then crossed to create a single hybrid offspring.

- 3. Mutation**

One parent is chosen via a selection tournament from among those in the previous generation. This offspring is a copy of this parent, and it has the mutation operator applied to it.

8.4 The Physics Engine

3DVCE is written in C++, and uses the open-source object-oriented physics engine called OPAL, which stands for Open Physics Abstraction Layer. OPAL is a wrapper for the true underlying physics engine, called ODE [Smith, 2007], which stands for Open Dynamics Engine. ODE is a rigid-body Newtonian physics simulator that models the dynamics of solid 3D polygons, joints, motors, gravity, friction, momentum, and so on. Details for the specific settings used for OPAL/ODE by 3DVCE are given in Appendix B. Graphics rendering for 3DVCE is handled by OGRE [OGRE, 2007], which stands for Object-oriented Graphics Rendering Engine, an open-source rendering engine for 3D graphics. OGRE wraps functionality provided by either the underlying OpenGL or DirectX graphics APIs.

8.5 Creature Phenotypes and Genotypes

8.5.1 Morphology and Control

8.5.1.1 Segments

Figure 8.1, below, shows twelve randomly generated 3DVCE creatures. Every creature's body is a branching recursive tree of cuboids (or, more technically, "rectangular parallelepipeds") called segments. Segments are six-sided, three-dimensional polygons, all with the same density, but with sizes and proportions determined by the genotype. The number of segments in the tree, its depth, and its structure are determined both by the creature's genotype, and by the process of embryogenesis that maps the genotype to a phenotype, described below.

Segments come in a finite number of types. Any given type of segment can appear zero, one or more time in a creature's body. All segments of the same type share many of the same properties, such as colour, proportions, control programs, branching structure, various parameters affecting the orientation and position of a joint connecting it with its parent segment, and more. The number of available segment types is a fixed parameter of the particular evolution being run. The properties of each segment type are determined by the genotype.

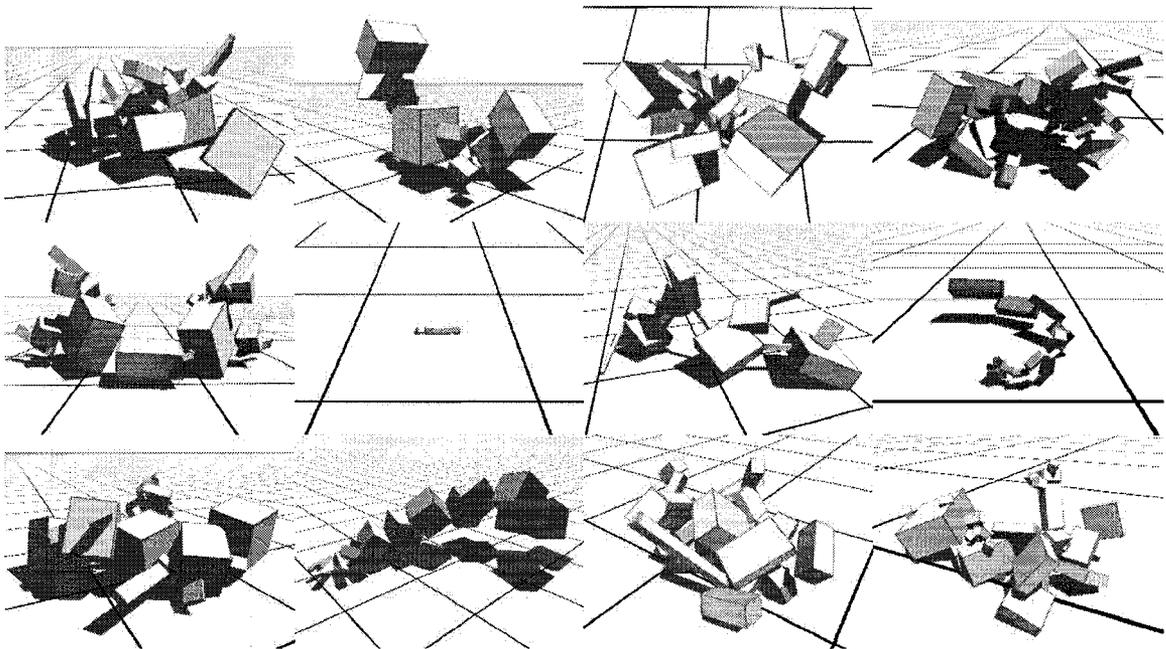


Figure 8.1 Twelve randomly generated 3DVCE creatures.

8.5.1.2 Joints

The tree is a recursive branching pattern of segments connected to other segments by simulated universal joints. These are joints with two axes of limited rotational freedom. The human wrist is somewhat like a universal joint. It can rotate left-right (adduction and abduction), and up-down (extension and flexion). These are analogous to the two degrees of rotational freedom provided by all creature joints used in 3DVCE. Note that the wrist

has a third degree of rotation freedom that universal joints do not. This extra freedom allows, primarily through movements of the radius and ulna bones in the forearm, and not the wrist joint itself, for a twisting motion (*pronation* and *supination*) such as when turning a doorknob.

8.5.1.3 Sensors

There are three different types of sensors that are present within every segment in a creature's body. The inputs from them are accessible for use by the creature's control system through several types of GP nodes (see Table B.1 in section B.3.2.2 of Appendix B for a complete list of all GP node types) available in the evolution of creature control systems. At each body segment, there are five sensors in all, as follows.

- **The Contact Sensor**

This sensor indicates whether or not the segment is in physical contact with any other solid objects, such as the ground, excluding those that are part of the creature's own body. This involves querying the physics engine for a list of all objects in contact with a segment, and then filtering out objects which are part of the body. This ability would be quite difficult to achieve in a real world contact sensor, but is straightforward in the virtual environment.

- **Proprioceptor #1**

This sensor indicates the current angle of the joint that connects a segment to its parent segment, for the first of the two axes of rotational freedom.

- **Proprioceptor #2**

This indicates the angle of the joint's other axis of freedom.

- **Incline Sensor #1**

This sensor indicates the current tilt of a segment about the z-axis. This is somewhat like the sense of orientation provided by the inner ear. The z-axis runs parallel to a line between the centre of the face containing the joint connecting a segment to its parent, and the opposite face.

- **Incline Sensor #2**

This indicates the tilt about the x-axis. The x-axis is horizontal for the root segment, which is always initially aligned with the global world axes, but for other segments its direction is determined by the initial orientation of the segment at the moment it is constructed.

8.5.1.4 Control

Creature control systems are composed of a large number of GP trees. Two of these trees are associated with the entire creature. These are referred to as the creature-level trees. They each calculate a value, one called output, the other called state, which is available, via its own terminal GP node, to all GP trees in the control system whether they are creature-level trees or not. Creature-level trees cannot directly access sensors in the body since they are not associated with any particular segment, but they can access output values produced by the root segment.

Each segment in the body also contains GP trees as part of the control system. There are four such segment-level trees in each segment. Two of them, also called output, and state, function in a manner similar to their counterparts at the creature level, except that the output value of a segment is accessible to any segment directly connected to it by a joint

(including itself), whereas state is only available locally to other GP trees in the same segment. The other two segment-level GP trees are called angle and torque. The angle tree specifies desired angles for servo motors that automatically generate torques on the segment's joint in an attempt to reach those angles. A segment may have many joints connecting it to other segments, but it controls only the joint connecting it to its parent segment in the body tree. The torque tree specifies the maximum amount of torque and speed that the servo will be permitted to use to reach its desired angle.

There are two servo motors at each joint, one for each axis of rotational freedom, yet there is only one angle tree and only one torque tree in the segment. The way in which each servo can receive different signals is through the use of a GP node type called `AXIS`. The `AXIS` terminal node returns the value 1.0 when GP tree outputs are being calculated for the second servo, and returns the value 0.0 at all other times, such as when calculating for the first servo, or when used in non-servo-related trees. In this way, it is hoped, the control of movement in both rotational axes can be evolved to be coordinated without being identical.

A creature's control system is updated before every simulation cycle during evaluation. A single simulation cycle corresponds to 0.01 second of simulated time, giving the control system an update frequency of 100Hz. Physics simulation time does not pass while GP tree outputs are being calculated, meaning that there is not cost or penalty associated with larger trees.

8.5.1.5 Oscillators

At the creature level, and within every body segment, there is an oscillator. Oscillators are simple sine wave signal generators. Their parameters (phase and wavelength) are determined genetically, and their outputs are accessible to the control system via particular GP function node types. The oscillator-accessing GP nodes take a single argument which amounts to a phase shift in the wave signal being sampled. Wave oscillations change with simulation time, so that different parts of the wave are sampled at different times during simulation. Analogous repetitive pattern generators are known to be involved in locomotion mechanisms in some types of real creatures [Delcomyn, 1985].

Oscillators are meant to make easier the task of evolving the kinds of repetitive movements conducive to locomotion. Several experimental runs in which the oscillator-accessing GP node types were made unavailable showed that, while the behaviors exhibited were quite different without oscillators available, creature populations are capable of evolving locomotion without oscillators as well as with them. Nevertheless, in all experiments described here, oscillators were available as components of control systems.

8.5.2 Creature Genotypes

Creature genotypes consist of a number of integer, real, and Boolean variables, as well as GP trees. As depicted in Figure 8.2, the structure is hierarchical with the creature-level oscillator parameters, two GP trees, and a vector of segment specifications at the highest level.

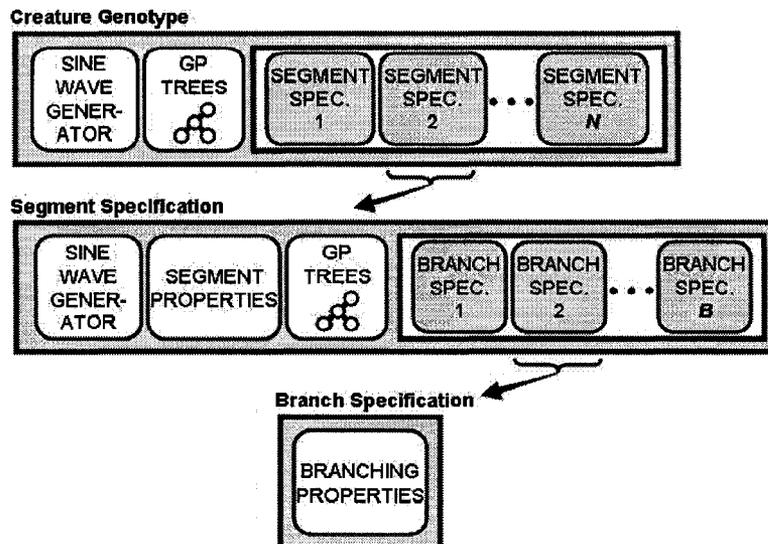


Figure 8.2 An illustration of the 3DVCE creature genotype.

The number of segment specifications, N , is a fixed constant specified in the settings for the evolution in which the creatures are participating. Each segment specification describes one of the types of segments available for construction of the body. Not all segment types in the genotype will necessarily be expressed in the phenotype. Segment specifications include parameters for the segments' oscillators, various other parameters such as dimensional proportions and colour, four GP trees, and a branch specification for up to $B = 5$ branches that extend from segments of the particular type. B is a fixed constant.

Each branch specification describes the manner in which a child segment in the body tree is to be attached to the parent, and the segment type of that branch. The type of a branch is simply an index into the segment specification vector. The specifications found at that index determine the properties of the segment forming the branch. This creates the potential for both direct and indirect recursion in the branching patterns in the phenotype. Since some segment specifications may not be expressed as one or more segments in the

phenotype, and since every branch specification contains an enable flag that can disable the branch when set to false, not every branch specification in the genotype will be expressed in the phenotype. A more detailed description of branch specifications and their effects on the phenotype is given in section B.3.2.3.1 of Appendix B.

Several values determine the range of sizes for creature genotypes. As mentioned above, the fixed system parameter determining the number of branch specifications per segment specification is one, and the evolution setting determining the number of segment specifications in the genome is another. In addition to these, there are upper and lower bounds on the sizes of GP trees in the genome. By necessity, the lower bound is a single node. The upper bound is set to 1,000 nodes. These bounds are enforced by all variation operators with the exception of one of the crossover operators which, at the time of this writing, can still potentially construct trees that exceed the upper bound.

8.5.3 Embryogenesis

The mapping from a particular genotype to a phenotype is referred to as embryogenesis in 3DVCE. Processes of embryogenesis in biology are, of course, vastly more complex than any existing EC system, including 3DVCE. Nevertheless, the term is borrowed here since the mapping follows something akin to a process of growth.

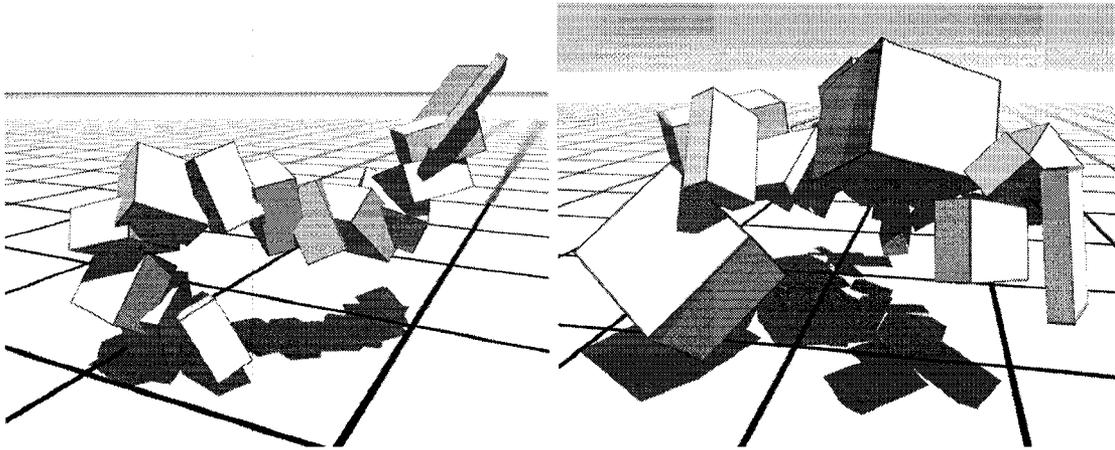


Figure 8.3 Two example screenshots of evolved 3DVCE creatures.

When configuring an evolution, two parameters are used to place constraints on the construction of creature bodies. These are the maximum number of segments, and the maximum depth. The maximum number of segments places an overall upper bound on the number of segments that a body can possess. This is not the same as the number of distinct segment types in the genome, which is also an evolution parameter, since it is possible to have many segments of the same type in a body. The maximum depth parameter specifies the maximum number of segments along any path from the body's root segment to one of its extremities (also called leaves or external nodes).

The process of embryogenesis proceeds in a depth-first fashion. It begins with the construction of the root segment according to the parameters given in the first entry of the segment specification vector. The root segment is always of the type specified in the first entry. It then proceeds to recursively construct the subtree of the first enabled branch from the branch specification vector. When that's complete, it moves on to the next enabled branch, and so on.

When a new segment cuboid is constructed, the physics engine is queried to determine whether any other body segments intersect it. If the segment intersects any other, with the exception of its parent, it is immediately removed and its subtree grows no further. This ensures that once a creature's body has been completely constructed, there are no intersecting segments that will cause the physics engine to generate strong repulsion forces once the simulation starts, in order to eliminate the intersections. No repulsion forces are generated between intersecting segments that are directly connected to one another by a joint. This is an optional feature of the physics engine and is necessary to allow the full range of motion at the joint.

A second place at which growth is truncated in the body tree is at segments that have reached the maximum depth from the root. This terminates what could otherwise be an infinite recursion during embryogenesis.

While building the body in the manner described above, if the number of segments exceeds the overall limit, growth is terminated and the body is deleted. The process then starts over with the maximum depth constraint reduced by one. This particular loop will repeatedly delete and restarting growth until a depth bound is found which results in a body that satisfies the segment count bound. The resulting body is then ready for evaluation.

8.6 Fitness Evaluation

There are six fitness components involved in the calculation of a fitness value. Each component is a particular aspect of creature performance that is measured and combined

in a weighted combination with all other components to come up with a single fitness value. The six components are as follows¹⁵.

1.DIST

This measures the maximum net horizontal distance traveled during evaluation. This is used to encourage evolution of the ability to traverse whatever terrain is present in the environment. This value can be either scaled or not scaled. When scaled, the distance value computed is divided through by a measure of creature body size. In this way, large creatures would be required to travel a greater distance than small ones to attain the same DIST measurement.

2.MAXH

This measures the maximum net upward vertical distance achieved during evaluation. MAXH is short for maximum height. This is used to encourage evolution of the ability to jump, although in principle it would also reward creatures that extend their bodies upward without leaving the ground. Like DIST, this value can also be scaled or not scaled to either take into account or to ignore creature body size.

3.AVGH

This measures the average height above the ground of the creature's centre of mass during evaluation. This encourages the evolution of tall creatures, but can, in principle, encourage the evolution of a jumping behavior. This value can be scaled or not scaled.

¹⁵ Note that all distance measurements are made between point-locations of a creature's centre of mass at two different times.

4.TOG

TOG is short for time on ground. This measures the percentage of evaluation time during which the creature remains in contact with the ground. This can be used to discourage jumping behavior. In combination with DIST this can be used to encourage a more stable form of locomotion. In combination with MAXH this can be used to encourage the evolution of creatures that extend their bodies upward without leaving the ground.

5.Sphere

This component can only be used in the Spheres environment, described below.

During evaluation, a sphere appears high above the ground at regular time intervals and falls. Spheres are deleted the moment they come in contact with the ground. The sphere fitness component begins at a value of zero and increments once during every simulation cycle for every sphere that hasn't yet touched the ground. This is used to encourage evolution of the ability to catch as many falling spheres as possible and to prevent them from reaching the ground.

6.Transport

Like the Sphere component, this component can only be used in the Spheres environment. This measures, over all spheres still present after the final evaluation cycle, the maximum horizontal distance that a sphere has been transported away from where it first appeared. This is used to encourage evolution of the ability to catch at least one falling sphere and to carry it as far as possible.

The six component measurements described above are not all as straightforward as their descriptions imply. In many cases, the most intuitive and obvious methods for measuring

the quantities described leave open initially unforeseen loopholes that allow undesirable solutions to evolve. Very tall creatures might attain a high DIST component value simply by falling over, for example. More details on the various ways in which fitness evaluation and its six components have been constructed to avoid these and other loopholes are given in Appendix B.

8.6.1 Environments

Together with the weighting values used to combine the six fitness components, each entry in a fitness schedule also specifies one of three environments in which the fitness measurements will take place. The first environment is a simple flat terrain. The second is a rough terrain in which the ground is made from a grid of solid blocks. Each block has the same width and length, but their heights vary to create jagged bumps and dips in the surface. The rough terrain was originally devised during early development to penalize the undesirable “dispendious jittering” strategies described by [Miconi & Channon, 2005]. This problem was eventually brought under control, but the rough terrain was kept since it was found to be more challenging than flat terrain, but not impossible. The third is the Spheres environment, which is identical to the flat terrain environment except that spheres of radius 1.0 appear every 100 simulation cycles high above the Cartesian origin of the ground plane, where creatures are spawned. Each sphere falls under gravity and disappears the moment it makes contact with the ground. Creatures cannot sense the location or presence of spheres unless they do so indirectly through contact sensors.

8.6.2 Combining Fitness Components

The six components described above are used in weighted combination to determine the assigned fitness measurement for a creature. The weights themselves are specified by the user during the configuration of an evolution. The component measures are not normalized since their ranges will depend on several factors such as the amount of simulation time provided for evaluation, whether or not they are scaled, and the particular environment in which fitness evaluation takes place.

Dozens of short videos showing many examples of 3DVCE creatures evolved in the flat terrain, rough terrain, and Spheres environments, and using a variety of weighted combinations of the six fitness components can be seen in The Zoo at [Graham, 2008a].

8.7 Variation Operators

Since some aspects of creature genotypes are GA-like, and some are GP-like, there are a number of quite different variation operators that act upon different parts of the genome in different ways. They can all be partitioned into crossover operators and mutation operators. They are described briefly here, and in detail in section B.6 of Appendix B.

Some operators are also domain specific in that they cause alterations that make the most sense when considered not from the point of view of the changes to the genotype, but from the point of view of the effects on the final phenotype. Examples of these include a mutation operator that copies a segment specification while preserving self-loops in the branching information so that entire structures that are directly recursive can be preserved

in the copying, a segment-splitting mutation operator that attempts to preserve overall body structure while splitting all segments of a particular type into two, and others.

8.7.1 Crossover Operators

At the level of the full creature genotype, crossover begins by choosing one of the two parents at random, giving the offspring a copy of that parent's creature-level oscillator. This is followed by an iteration through the segment specification vector, at each index, i , creating a new specification by crossing specifications i from both parents. Finally the two creature-level GP trees, output and state, are generated by the 3DVCE GP tree crossover operator. The methods for crossing segment specifications and GP trees are explained briefly below, and in detail in section B.6.1 of Appendix B.

8.7.1.1 Segment Specification Crossover

Crossing two segment specifications starts by choosing one of the two parents at random and copying its segment specification oscillator and all of its integer, real, and Boolean parameters to the offspring. This is followed by crossing the four GP trees, state, output, angle, and torque, using the GP tree crossover method described below.

8.7.1.2 GP Tree Crossover in 3DVCE

The first step in crossing a GP tree in one parent with a GP tree from the other parent is to decide whether or not the corresponding tree in the second parent will be crossed, or whether another GP tree will simply be chosen at random from among those available in the other parent. The probability of choosing a tree other than the corresponding one is a

parameter of the evolution configuration and is kept low so that most crossings are between related trees.

The second step is to decide whether the crossover will be done with the regular GP tree crossover method, or with the special GP tree crossover method. This probability is also a parameter of the evolution configuration. In all of the 3DVCE experimental setups, this was set to 50%.

The regular GP tree crossover method copies the first parent tree. It then chooses a subtree uniformly at random from the tree. A subtree to replace it is chosen uniformly at random from the second parent tree and is copied across to the offspring. The replacement subtree is chosen from among those in the second parent tree that would result in the offspring tree having a node count that doesn't exceed the upper bound of 1000 nodes.

The special GP tree crossover method traverses both parent trees in lockstep, and depth-first (more precisely, this is a "pre-order" traversal) beginning with the root and working downward through the subtrees. When the two parent tree nodes being visited in the traversal are exact matches, the offspring GP tree ends up with the same node in the same location. When the two nodes differ, one of the parent trees is chosen at random to contribute a copy of its entire subtree at that location to the offspring tree. At the time of this writing, the special crossover method is capable of generating offspring trees that exceed the node count limit of 1000 nodes.

8.7.2 Mutation Operators

When offspring are produced from only a single parent, the mutation operator calls three other mutation operators. The first is an operator for mutating oscillators. This simply makes random changes to the oscillator's parameters. The second is the operator for mutating a segment specification, described below. This is called once for each segment specification in the genome. The third is the GP tree mutation operator, described below. It is called once for the output tree, and once for the state tree.

8.7.2.1 Segment Specification Mutation

The segment specification mutation method mutates the oscillator parameters, all the specification's integer, real, and Boolean parameters, and each of its four GP trees. There are five other speciation segment-related mutation methods that may be called as well.

These are as follows.

- **Segment Copying**

This mutation copies a segment specification from one index in the vector to another.

It does so in such a way that self-loops in the branching pattern of the body tree are preserved in the copying.

- **Segment Splitting**

This mutation attempts to preserve the overall architecture of a creature's body while splitting all segments of a particular type into two.

- **Segment Randomizing**

This mutation simply replaces a segment specification with a randomly-generated replacement.

- **‘Stub’ Growing**

This mutation produces a segment branch that consists of a single segment which has no branches of its own. This mutation is meant to increase body complexity in a way that minimizes the negative impact on fitness that tends to accompany changes in morphology.

8.8 Population Initialization

Genotypes in the first generation of a run in 3DVCE simply assign to each integer, real, and Boolean parameter in the genome a value which is chosen randomly from that parameter’s legal range.

8.8.1 Creature Bushiness

In the case of the enable flags for branch specifications, these are set to true using a probability called the bushiness value. The bushiness value lies between 0.1 and 0.9 and is assigned a new uniform random value from that range before the construction of the genotype of each randomly-generated creature in the population. In this way, creature morphologies are more widely distributed across the range from very sparse to very bushy in their branching patterns.

8.8.2 Randomly-Generated GP Trees

In the case of the GP trees in the genome, these are very limited in size in randomly-generated creatures. The population initialization method attempts to produce random GP trees that contain between 1 and 10 nodes instead of sizes in the full legal range of 1 to

1000 nodes. This choice was made based on early experimentation that seemed to show more subjectively interesting behavior from randomly-generated small trees than from large ones.

8.9 Exaptation Experiment: Travel across Rough Terrain

The first exaptation experiment in 3DVCE involves a comparison between two evolution configurations. The first, configuration #1, evolves populations using a fitness function that measures the distance a creature can travel in the rough terrain environment, described above. The second, configuration #2, is identical to the first except that runs are split into two epochs. The first epoch is relatively short, and evolves the population using a fitness function that rewards the ability to jump. The second epoch is longer, and uses the same fitness function used in configuration #1. The two fitness functions used in configuration #2 have different evaluation times, but the total amount of simulated time per run is the same for configurations #1 and #2.

The motivation behind this experiment is to see whether, given otherwise identical settings, an evolution involving the change of function occurring in configuration #2 can outperform configuration #1 in which the fitness function remains fixed throughout the run. A jumping ability was chosen as the means for preadapting the population in configuration #2 since it was hoped that that ability might allow creatures to more easily get past the jagged bumps and dips present in the rough terrain environment simply by jumping over them.

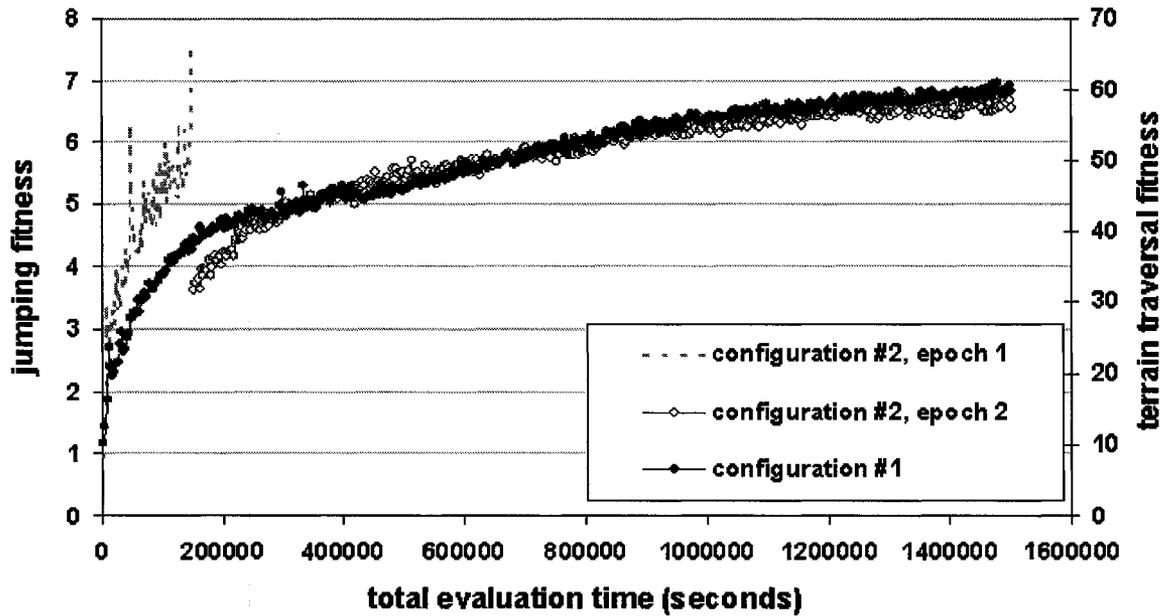


Figure 8.4 Best fitness vs. generation, averaged over all runs, for 3DVCE configurations 1 and 2.

Configuration #1 uses a population of 100 creatures evolving for 500 generations. Each creature is evaluated for 3000 simulation cycles. Configuration #2 uses the same population size, but the first epoch lasts 150 generations and has an evaluation time of 1000 cycles. The second epoch lasts 450 generations and has an evaluation time of 3000 cycles. The total evaluation times per run are equal between the two configurations. A complete list of the settings used in configurations #1 and #2 are given in section B.8 of Appendix B.

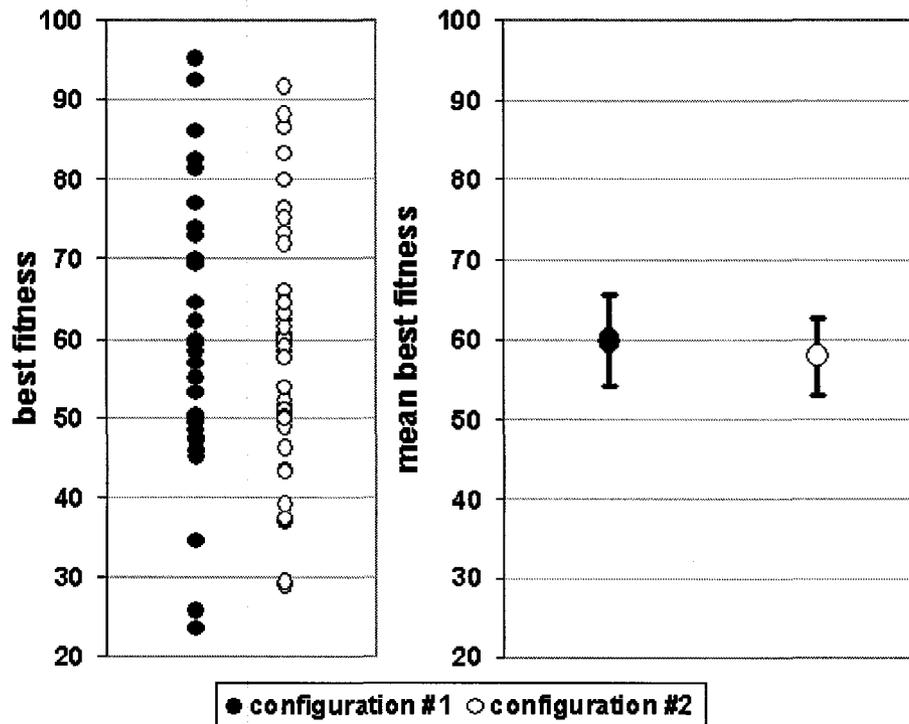


Figure 8.5 Best-of-run fitness values and means thereof from independent runs of 3DVCE configurations #1 and #2.

The results are shown in Figure 8.4. The mean best-of-run fitness for configuration #2 is slightly lower than the corresponding statistic for configuration #1. Owing to large expense in terms of processing time required for evaluations, only 35 runs for configuration #1, and 40 runs for configuration #2 were completed. Figure 8.5 shows the actual best-of-run fitness values from the 75 runs. The values for the non-exaptive approach have nearly the same mean as the non-exaptive approach, but they also have a slightly wider distribution than those of the exaptive approach. Although the pattern may be an artifact of the small sample size, observations of the behaviors of the evolved creatures reveal an interesting pattern. Some of the runs in which creatures first develop jumping ability before switching to rough terrain traversal become trapped on a jumping strategy that imparts no consistent direction of travel to the creature. Each new jump, though it may throw the creature far, is in a random direction. Since the population size is

100, there will almost always be a number of offspring that, by chance, manage to travel far from the starting position. These offspring will be assigned high fitness values for rough terrain traversal, and will tend to keep the population pinned to this suboptimal strategy.

Those runs that don't encounter this problem, either by evolving a jumping strategy that imparts a more or less consistent direction of travel or by discovering a means of adapting the jumping strategy so that it becomes consistent, seem to produce very fit individuals that travel a good distance across the terrain with repeated jumps in a consistent direction. Nevertheless, even the best of these are outperformed by the best from the non-exaptive configuration.

If the observed wider distribution for the best-of-run fitness outcomes for the non-exaptive approach is not merely an artifact of the small sample size, this would be an interesting result even if the average best-of-run fitness were lower for the non-exaptive approach (the observed mean was higher, not lower, however). In such a case, if runs are performed in batches of N independent runs instead of singles then, for large enough N , the best-of-batch fitness statistic for the non-exaptive approach will be higher than the corresponding statistic for the exaptive approach. As it stands, the only benefit for the exaptive approach that can be seen in the available data is a tendency to avoid some of the worst outcomes that the non-exaptive approach produces. This seems to come at a cost of also avoiding some of the best outcomes. The sample size is too small to allow any strong conclusions to be drawn. A Mann-Whitney U test comparing the two sets of outcomes gives a p value of only 0.804380, which is not significant.

8.10 Exaptation Experiment: Catching and Transporting a Sphere

The second exaptation experiment in 3DVCE involves a comparison between two evolution configurations. The first, configuration #3, evolves populations using a fitness function that rewards creatures for maximizing the *Transport* component of fitness, described above. To maximize this component, creatures must catch at least one of the spheres that appear at regular intervals in the *Spheres* environment and carry it as far as possible in the allotted evaluation time. The second configuration, #4, is identical to the first except that runs are split into two epochs. As with the differences between configurations #1 and #2, the first epoch in #4 is relatively short, and evolves the population using a fitness function that rewards the ability to catch as many spheres as possible and to prevent them from reaching the ground. The second epoch is longer, and uses the same fitness function used in configuration #3.

The motivation behind this experiment is to see whether, given otherwise identical settings, an evolution involving the change of function occurring in configuration #4 can outperform configuration #3. The ability to simply catch spheres was chosen as the means for preadapting the population in configuration #4 since it was hoped that the resulting problem will have been simplified. It is hoped that in the first epoch the creatures learn only how to catch the spheres. Once they can do so, the second epoch may involve learning only the ability to move. The intention is for the two-step learning task to be easier to adapt to than the combined version created by the *Transport* fitness component alone.

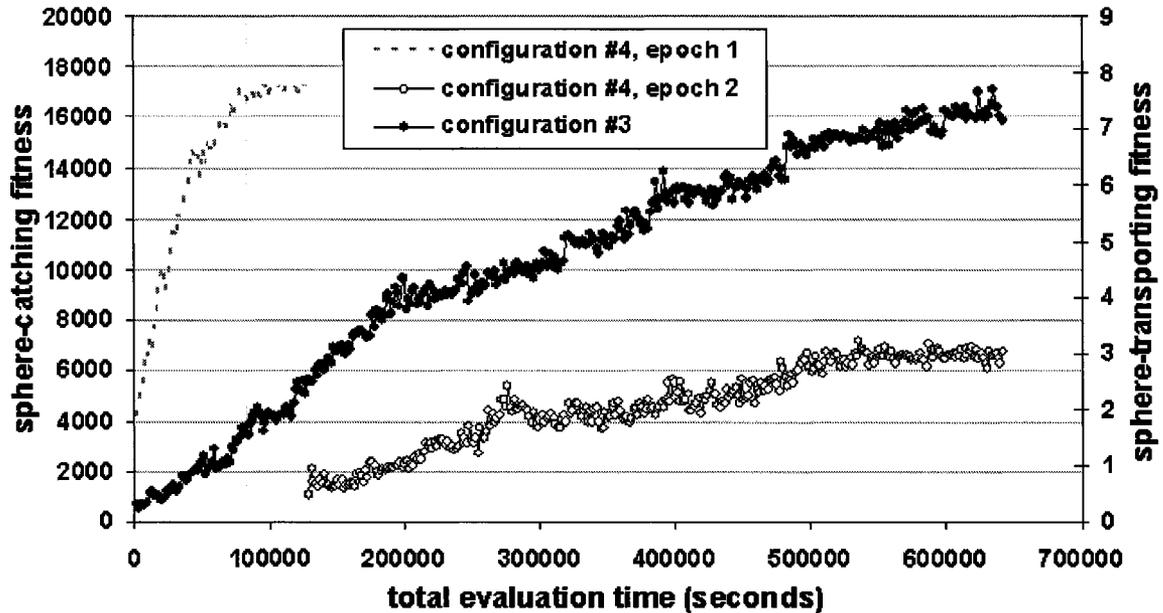


Figure 8.6 Best fitness vs. generation, averaged over all runs, for 3DVCE configurations 3 and 4.

Configuration #3 uses a population of 80 creatures evolving for 400 generations. Each creature is evaluated for 2000 simulation cycles. Configuration #4 uses the same population size, but the first epoch lasts 80 generations. The second epoch lasts 320 generations. Both epochs use an evaluation time of 2000 cycles. The total evaluation times per run are equal for the two configurations. A complete list of the settings used in configurations #3 and #4 are given in section B.8 of Appendix B.

Only a small handful of independent runs were completed for these two configurations. The extreme cost in terms of processing time was a deciding factor early on in testing when available processing power was directed away from configurations #3 and #4 and diverted to configurations #1 and #2 since the results at that time had seemed more promising for the latter pair. The result is that only 14 runs were completed between the two configurations, 9 with configuration #3 and 5 with configuration #4. The outcome is negative but carries little statistical strength.

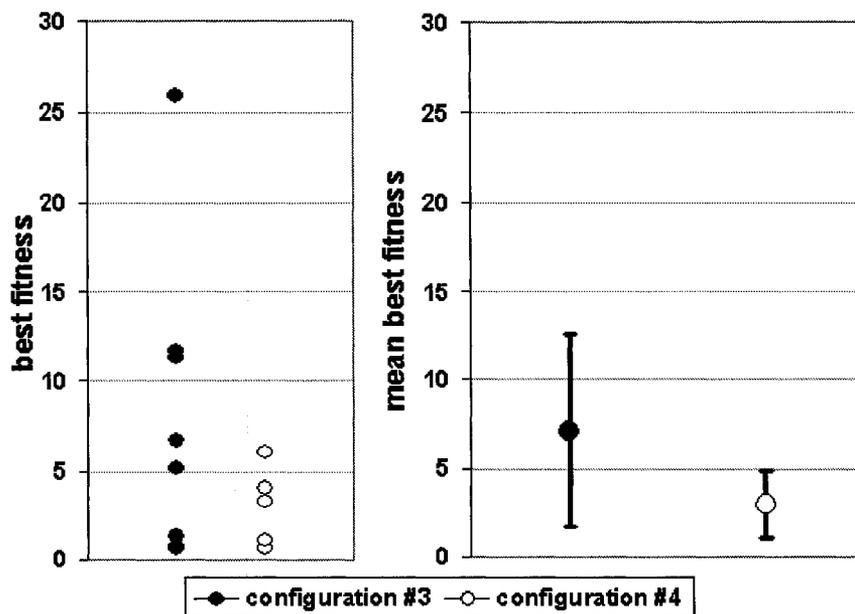


Figure 8.7 Best-of-run fitness values from independent runs of 3DVCE configurations 3 and 4.

The results are shown in Figure 8.6 and Figure 8.7. The exaptive approach appears to be detrimental in this instance. A Mann-Whitney U test comparing the data gives a p value of only 0.437440.

Chapter 9 Exaptation Experiments with Symbolic Regression using GP

This chapter details experiments to test two scenarios for the use of exaptation using a GP system for symbolic regression problems. The first, in section 9.1, involves an attempt to preadapt populations by evolving solutions to simplified versions of a multivariate target function. The second, in section 9.2, examines whether, for a single-variable regression problem, a population can be beneficially preadapted by evolving a function that approximates the target's derivative.

9.1 Piecewise Build-Up to a Target Function

The motivation behind the experiments described in this section is to see whether a multivariate symbolic regression learning task can be made easier by building up to the target function in steps of increasing complexity. Normally the target function is not known ahead of time in symbolic regression; hence it is the purpose of a regression method to find an approximation. In this case the target function is decided ahead of time in order that it can be approached in steps. The idea is to see whether these steps make the learning task easier.

9.1.1 Target Functions

The target function used is a function involving three variables. It is as follows.

$$(2.4x^2z) + (1.1xy) + (0.7yz) - xyz \quad (9.1.1-1)$$

The specific function was chosen for no reason other than that is it, more or less, a normal polynomial. Both a two- and three-step version of the GP system was tried in this

experiment. The two-step version used the following target function prior to switching to (9.1.1-1).

$$(2.4x^2z) + (1.1xy) + (0.7yz) \quad (9.1.1-2)$$

The three-step version used the following target function before switching to (9.1.1-2) and later (9.1.1-1).

$$(2.4x^2z) + (1.1xy) \quad (9.1.1-3)$$

9.1.2 The GP System

The GP system used was ECJ version 18 [Luke et al., 2008], a Java-based evolutionary computation research system. The GP algorithm settings were as shown in Table 9.1, below. Most of the settings in the table are the defaults for Koza-style GP in ECJ.

Setting, Parameter, or Option	Value
Function set	{+, -, *, x, y, z, <i>ERC</i> } +, -, and * have arity 2; all others are terminals. <i>ERC</i> is the so-called 'ephemeral' random constant in the range [-1.0..+1.0]. x, y, and z are variables in the same range.
Halting condition	500 generations
Population size	100
Elitism	1 elite individual
Duplicates	100 retries for duplicates
Variation operators	90% crossover, 10% mutation
Selection	Tournament (size 7)
Population initialization	Ramped Half and Half

Table 9.1 GP system settings for the 'piecewise' symbolic regression experiments.

Each run of the GP system independently chooses 100 triples, (x,y,z), with x, y, and z chosen uniformly at random from the range [-1.0..+1.0]. This set of triples remains fixed for the duration of a run and is reset upon the start of a new run. Individuals are evaluated on each triple and are assigned a score equal to the sum of absolute differences between

their computed outputs and the outputs from the target function. This score, s , is adjusted to become the fitness value $F = 1.0/(1.0+s)$, which falls between 0 and 1. Any absolute difference less than or equal to 0.01 between an individual's output for a given triple and the output of the target function is counted as a hit. The individual with the best fitness value at the end of each generation has both its hit total and its fitness logged.

The number of generations per epoch was not fixed, and transitions from one epoch to the next were triggered by the discovery of any individual with a hit count greater than or equal to a fixed threshold of 20. The only exception to this is that the transition to the final epoch always occurs 50 generations prior to the end of a run whether the hit count threshold is reached or not.

9.1.3 Results

Figure 9.1, and Figure 9.2, below, show the distributions of best fitness values and best hit counts¹⁶ for the non-exaptive one-epoch configuration and two- and three-epoch exaptive configurations. No appreciable differences between the three configurations are seen in these graphs.

¹⁶ The recorded hit counts are not necessarily the highest hit counts from each run. Instead, they are simply the hit counts from the individuals with the highest fitness.

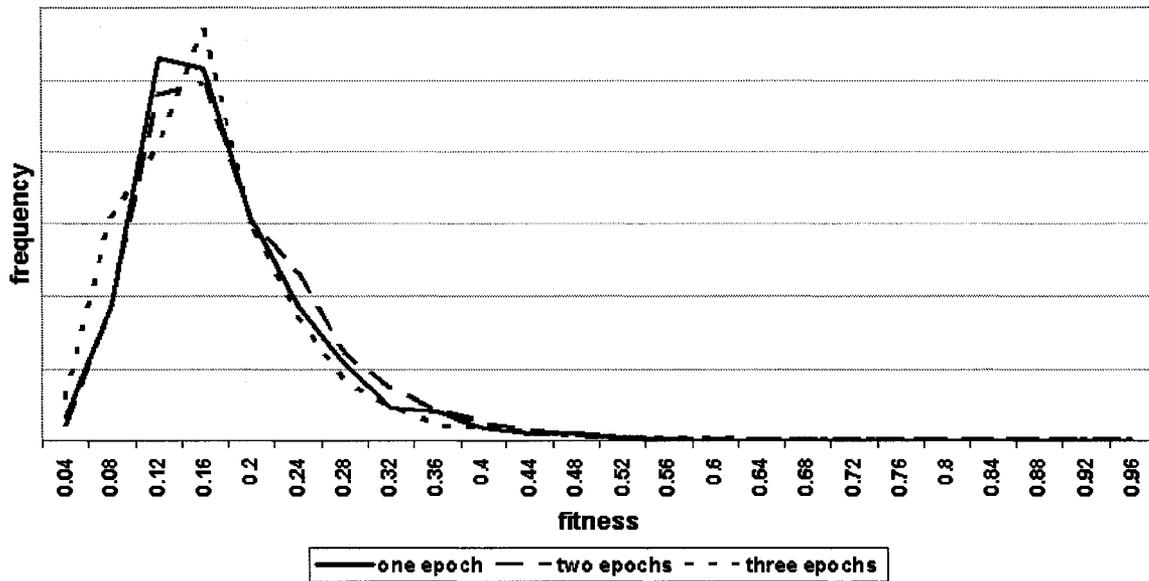


Figure 9.1 Best fitness outcome distributions for the 'piecewise' exaptive GP system.

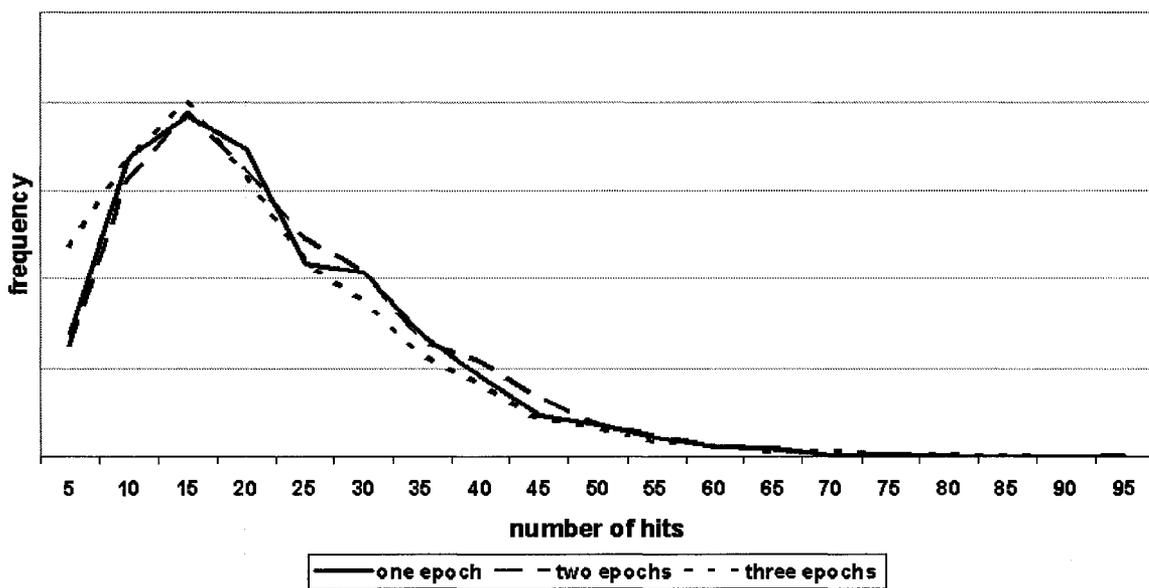


Figure 9.2 Outcome distributions for the number of hits from the best individuals in the 'piecewise' exaptive GP system.

The mean best fitness values and mean hit counts are plotted on orthogonal axes, below, in Figure 9.3.

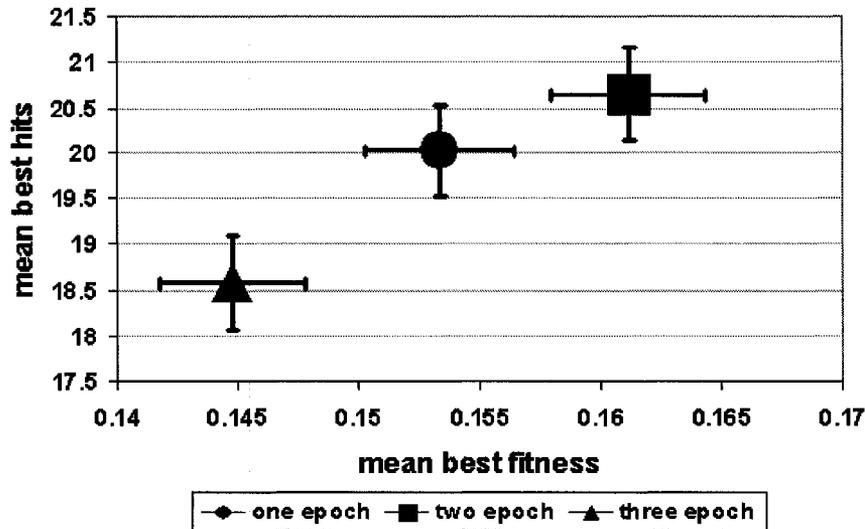


Figure 9.3 Mean best fitness and hits, with 95% confidence intervals, for the 'piecewise' exaptive GP system.

The figure shows that, in terms of mean performance from the GP system, there are differences between the three configurations. The three-epoch configuration exhibits the worst performance, on average, and the two-epoch configuration exhibits the best performance. The non-overlapping confidence intervals along the mean best fitness x-axis show that the benefit obtained from the two-epoch configuration over the non-exaptive one-epoch configuration is statistically significant. A comparison between their best hit values, using the Mann-Whitney U test, results in a p value of only 0.083388 which is not convincingly significant.

9.1.4 Conclusions

There is little in the results of the 'piecewise' exaptive GP experiments that lend support for the use of this particular type of exaptation. While the aim was to see whether a population could be "lead by the hand" down a particular evolutionary trajectory by slowly building toward the final fitness function, the results were both mixed and

unimpressive. Had the 3-epoch configuration outperformed the others, this might have lent some support to the ‘piecewise’ approach. As it turned out, however, at least in this single instance, the 2-epoch configuration exceeded the others in performance, and none of the performance differences were notably substantial. It may well be that there is something peculiar about the particular GP settings and fitness function used, but this seems unlikely given that (9.1.1-1) is a fairly ordinary polynomial and that many of the settings in Table 9.1 are the default options for the ECJ project’s symbolic regression sample.

9.2 Using the Target’s Derivative for Preadaptation

In this section a very similar GP system to the one described in section 9.1, above, is used to test an exaptive approach on a single-variable symbolic regression problem. Instead of simplifying a target function that is known ahead of time, this system uses the set of pairs of the form $(x, f(x))$, that represent the regression problem, to produce a second set of pairs that approximate the target function’s derivative. The motivation behind this idea comes from the fact that there is often a large amount of similarity between the symbolic representation of a function and the symbolic representation of its derivative. Although the two may produce curves that differ drastically from one another, the symbolic representation of a function’s derivative, within a GP population, may contain useful subtrees that prime the population for evolving the function itself. This section tests this notion using several different target functions.

9.2.1 Target Functions

Five different target functions were tried, each taking input variable x over the range $[0..1]$. They are as follows.

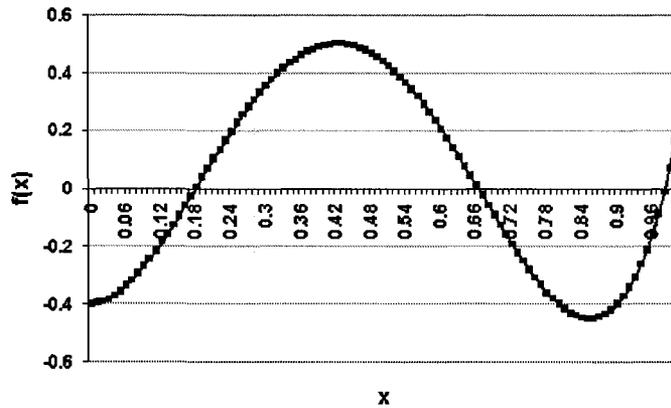


Figure 9.4 Target function #1. $f(x) = 28x^2(x - 0.8)(x - 0.9) - 0.4$

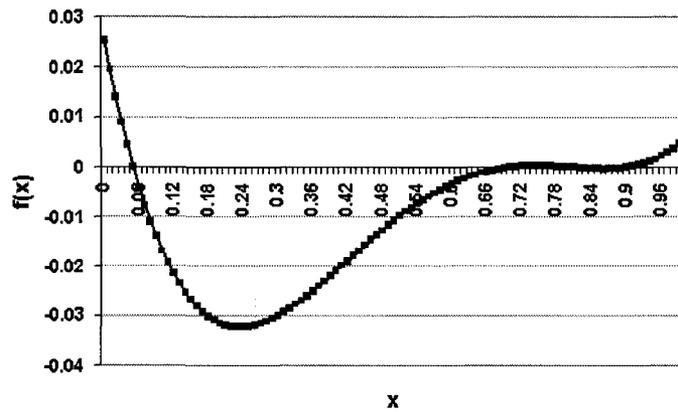


Figure 9.5 Target function #2. $f(x) = (x - 0.05)(x - 0.7)(x - 0.8)(x - 0.9)$

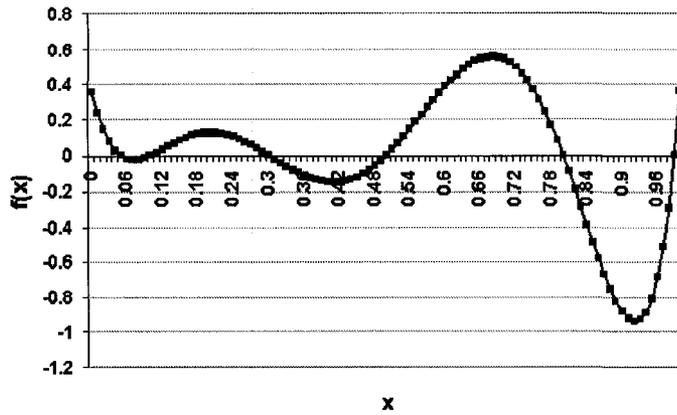


Figure 9.6 Target function #3. $f(x) = 600(x - 0.05)(x - 0.1)(x - 0.3)(x - 0.5)(x - 0.99)$

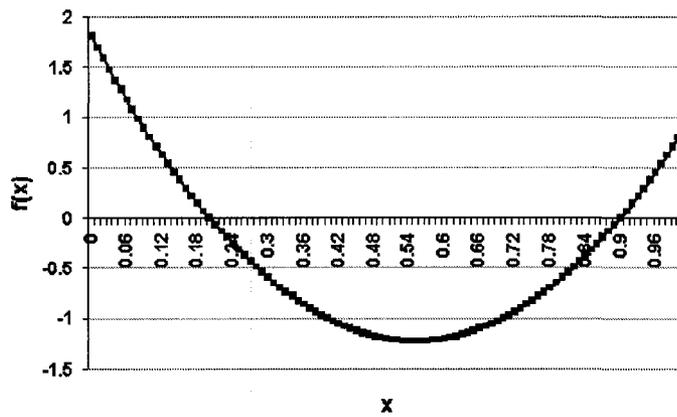


Figure 9.7 Target function #4. $f(x) = 10(x - 0.4)(x - 0.7) - 1$

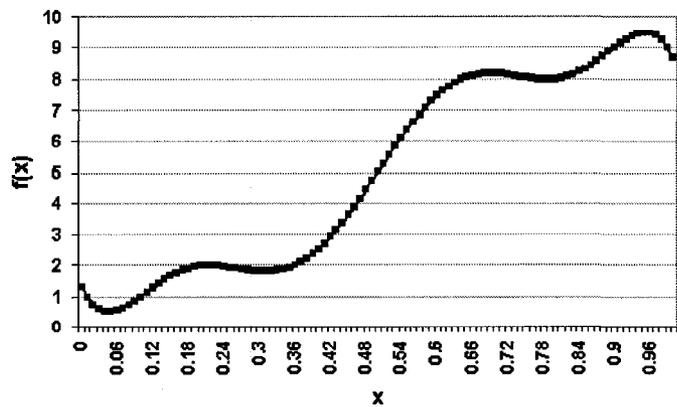


Figure 9.8 Target function #5. $f(x) = 100000x(x + 0.3)(x - 0.05)(x - 0.1)(x - 0.2)(x - 0.5)(x - 0.8)(x - 0.9)(x - 0.95)(x - 1.3)$

An approximation of the target function's derivative is attained by taking the n pairs available, $\{(x_1, f(x_1)), (x_2, f(x_2)), \dots, (x_n, f(x_n))\}$, sorted by their x values, and computing $n-1$ new pairs of the form $((x_i + x_{i+1})/2, (f(x_{i+1}) - f(x_i))/(x_{i+1} - x_i))$ where i runs from 1 to $n-1$.

9.2.2 The GP System

The GP system used was identical to the ECJ system described in section 9.1.2 with the exception of the settings listed in Table 9.2, below.

Setting, Parameter, or Option	Value
Function set	$\{+, -, *, x, ERC\}$ $+$, $-$, and $*$ have arity 2; all others are terminals. ERC is the so-called 'ephemeral' random constant in the range $[-1.0..+1.0]$. x , is a variable in the same range.
Halting condition	300 generations

Table 9.2 Settings for the 'derivative' exaptive GP system.

For this system, 40 values for x are generated uniformly at random for each run. The values are in the range $[0.0..1.0]$ and are sorted in ascending order. Once the x values are chosen and sorted, the corresponding 39 x values for the approximated derivate curve are computed and stored. Fitness values and hit counts, no matter the target function, are computed in the same manner described above in section 9.1.2. Transitions from the preadaptation epoch to the normal epoch are triggered by the discovery of any individual with a hit count meeting or exceeding a specified threshold. Except where it is stated otherwise, this threshold is 5 hits.

9.2.3 Results

Table 9.3, below, shows best fitness statistics and corresponding hit count statistics for each of the five targets functions, both with and without exaptation.

Target function	Mean best fitness without exaptation	95% confidence interval	Mean best fitness with exaptation	95% confidence interval	Mean best hits without exaptation	95% confidence interval	Mean best hits with exaptation	95% confidence interval	Samples (without exaptation)	Samples (with exaptation)
#1	0.501	±0.043	0.393	±0.045	20.60	±2.10	14.37	±2.12	100	100
#2	0.893	±0.014	0.906	±0.014	35.12	±0.92	35.89	±0.89	250	250
#3	0.284	±0.022	0.182	±0.018	8.95	±1.08	4.41	±0.75	100	100
#4	0.552	±0.064	0.417	±0.062	22.94	±2.70	16.82	±2.61	100	100
#5	0.072	±0.007	0.047	±0.002	2.64	±0.40	1.22	±0.20	100	100

Table 9.3 Best fitness and hits statistics for the 'derivative' exaptive GP system.

The same data are shown graphically in Figure 9.9, below.

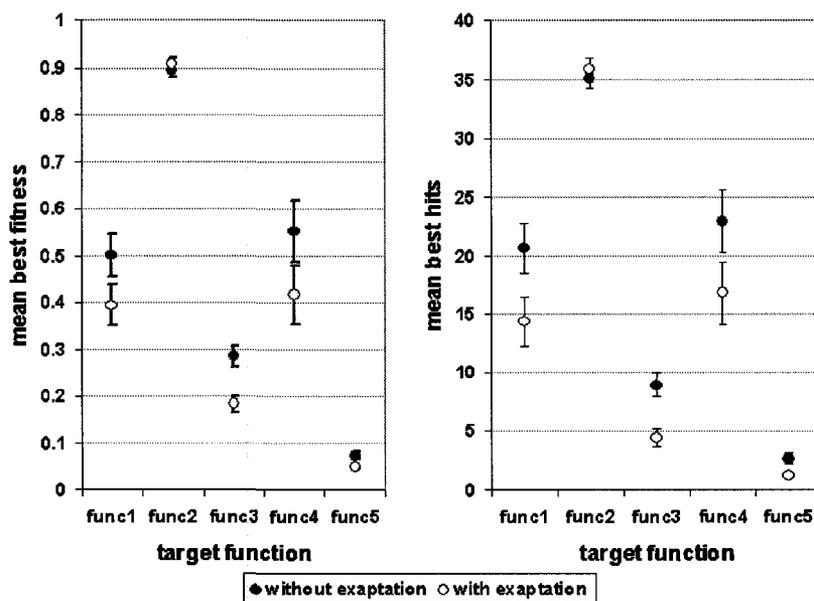


Figure 9.9 Mean best fitness values and hits, with 95% confidence intervals, for the 'derivative' exaptive GP system.

All target functions, with the sole exception of #2, show decreased performance when using the exaptive approach. Target function #2 seemed the most likely candidate to explore further. The results obtained from the combination-fitness exaptive GA in Chapter 6 showed that the amount of preadaptation can strongly influence the

performance of an exaptive EC system. With this in mind, the number of generations of preadaptation was varied for target function #2 and more data were collected. Since the preadaptation epoch terminates upon a hit count threshold, the amount of preadaptation was varied by varying this threshold. The results are listed, below, in Table 9.4.

Hit threshold	Mean best fitness	95% confidence interval	Mean best hits	95% confidence interval	Samples
0	0.8990	± 0.0103	35.36	± 0.65	450
1	0.9198	± 0.0089	36.70	± 0.55	450
2	0.9085	± 0.0099	35.82	± 0.64	450
3	0.9192	± 0.0088	36.69	± 0.54	450
4	0.9225	± 0.0089	36.80	± 0.55	450
5	0.9108	± 0.0095	36.19	± 0.60	450
6	0.9227	± 0.0086	36.94	± 0.51	450
7	0.9182	± 0.0089	36.81	± 0.53	450
8	0.9187	± 0.0095	36.50	± 0.60	450
9	0.9153	± 0.0102	36.53	± 0.62	450
10	0.9002	± 0.0121	35.61	± 0.75	450

Table 9.4 Mean best fitness values and hit counts for several hit thresholds for target function #2.

These results are displayed graphically in Figure 9.10 and Figure 9.11, below.

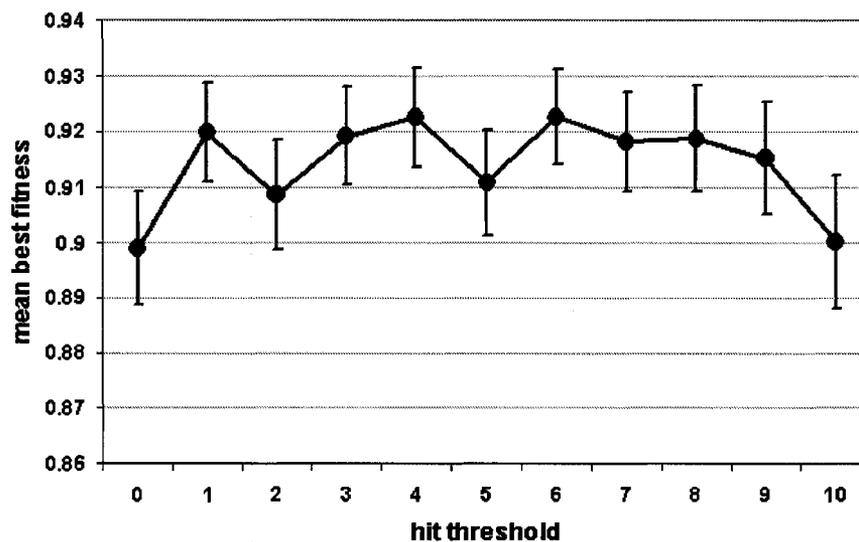


Figure 9.10 Mean best fitness values, with 95% confidence intervals, for several hit threshold values for target function #2.

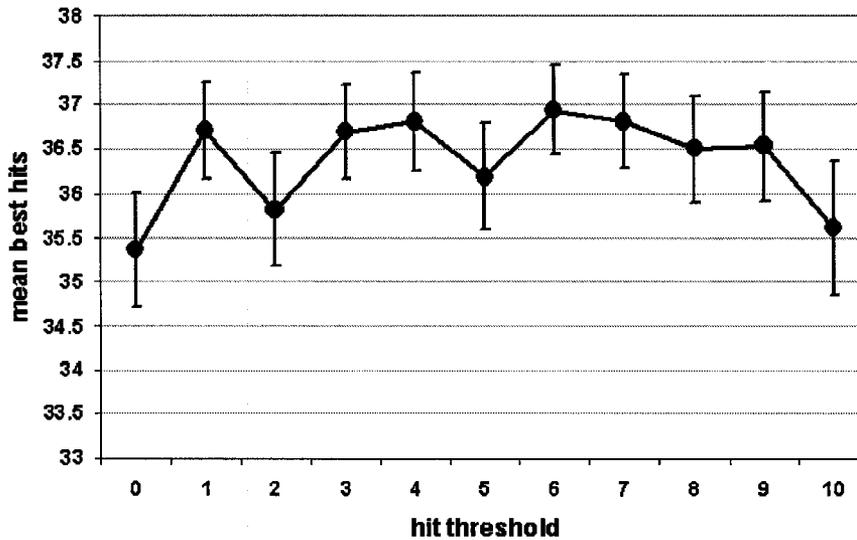


Figure 9.11 Mean best hits, with 95% confidence intervals, for several hit threshold values for target function #2.

A threshold value of zero corresponds to a normal non-exaptive GP system. Although the error bars are not as tight as might be desired, the general trend is toward a small improvement in performance for each of the exaptive configurations.

9.2.4 Conclusions

When the results obtained from comparisons of the non-exaptive and exaptive approaches are considered over all of the five target functions together, it appears that the exaptive approach is most often a detriment to the performance of the GP system. When the results obtained from comparisons of various amounts of preadaptation for target function #2 are considered, there is some potential but small benefit to this exaptive approach.

If at least a minor benefit can be found in a sample of only five target functions, then perhaps this 'derivative' exaptive approach has some merits. It is difficult to say why it

should be beneficial in some cases and detrimental in others. The answer may lie partly in the manner in which a GP system comes to represent its solution functions. The motivation behind the ‘derivative’ approach stemmed from the fact that many functions share structural similarities with their derivatives when their symbolic representations are examined. This, however, will vary depending on the representation, and a GP system will use whatever useful representation it should happen to find. As an illustrative example, target function #4, which is only an order two polynomial, can be represented in many ways, including the following eleven.

- | | | | |
|-----|-------------------------------|-----|------------------------------|
| 1. | $10(x - 0.4)(x - 0.7) - 1$ | 2. | $(10x - 4)(x - 0.7) - 1$ |
| 3. | $(x - 0.4)(10x - 7) - 1$ | 4. | $(5x - 2)(2x - 1.4) - 1$ |
| 5. | $(2x - 0.8)(5x - 3.5) - 1$ | 6. | $10x^2 - 11x + 1.8$ |
| 7. | $10(x - 0.1\bar{9})(x - 0.9)$ | 8. | $(10x - 1.\bar{9})(x - 0.9)$ |
| 9. | $(x - 0.1\bar{9})(10x - 9)$ | 10. | $(5x - 0.\bar{9})(2x - 1.8)$ |
| 11. | $(2x - 0.3\bar{9})(5x - 4.5)$ | | |

These are only a few of the ways in which function #4 can be written. Depending on whether or not a GP system includes parsimony pressure, and on the degree to which it is successful, any number of variations on these and other forms may evolve. Included within the space of solutions producible by GP is a large number of functions that merely approximate the target but which are not equivalent to it. Given that the same abundance of possibilities exists for the target’s derivative, the ability of the preadapting epoch to produce useful subtree building blocks that improve the overall outcome is not a straightforward issue. Nevertheless, if even a slight benefit can be found, such as was seen for target function #2, then this approach, which easily generalizes to any single-variable symbolic regression problem, may be a worthwhile avenue for further pursuit.

Chapter 10 Exaptation Experiments with a GP Maze Runner

This chapter examines a genetic programming system for guiding a simple agent through a series of mazes of varying sizes and difficulties. The system uses a two-epoch exaptive approach that first trains the population using a simpler learning task as the fitness function in an initial preadaptation epoch followed by an epoch that uses the desired fitness function measuring maze-solving ability. The GP system is compared with an otherwise-identical system that uses no exaptation, to see whether or not the exaptive approach can produce superior performance.

10.1 Description

This section describes the GP system controlling the agent, the agent itself, the structure of the maze environments, and the fitness functions that apply in each epoch.

10.1.1 The Mazes

All mazes are embedded in a 2D grid of cells. Each cell can be either a *blank* or a *wall*.

The agent can move within the maze by stepping from one *blank* cell to another adjacent *blank* cell. Cells are considered adjacent only in the four cardinal directions – north, south, east, and west. To keep the agent in the maze at all times, all cells at the extreme edges of the grid are *wall* cells.

In total, there are seven mazes used during fitness evaluation. Each is shown as a figure in Appendix C. The mazes range in size from 10 by 10 to 30 by 30. Some mazes contain

cycles while others do not. Some contain large open spaces with no walls, while others consist entirely of paths that are only a single cell wide.

Each maze has one of its cells designed as the *start* cell, and another designated as the *finish* cell. An agent will always begin maze navigation in the *start* cell, with the goal of reaching the *finish* cell.

10.1.2 The Agent

The agent possesses a state, which is its current row and column number in the maze grid as well as the direction it is currently facing. Using various functions in its GP tree, an agent can sense the contents of the four adjacent cells around it as well as the contents of the cell in which it currently resides. Its actions include moving forward or backward, examining the contents of nearby cells, and changing the direction in which it is facing.

In addition to being able to move from one *blank* cell to an adjacent *blank* cell, the agent can also paint a symbol on its current cell. This symbol is an “arrow” pointing in one of the four cardinal directions. The agent can sense whether or not its current cell or a cell next to it contains such a mark and can sense the direction of the arrow. This provides the agent with the ability to leave virtual “bread crumbs” as it moves. These could be of some use in navigating through a maze.

10.1.3 Fitness Functions

The maze runner GP system uses two fitness functions. The first, described in 10.1.3.1, is meant to quantify how well the seven mazes were solved. The second, described in

10.1.3.2, is meant to encourage exploration of the mazes, with no regard for the location of the *finish* cell. The maze navigator GP system attempts to determine whether the former can beneficially preadapt a population for that latter.

When an individual is evaluated in a given maze, it is permitted to execute its GP tree control system $2rc$ times where r is the number of rows in the maze and c is the number of columns. This is sufficient time to visit every cell in the grid twice, and since many cells are *wall* cells that cannot be visited, this is more than enough time to explore the entire maze if moves are chosen intelligently.

10.1.3.1 The Maze Solving Fitness Function

The fitness value for measuring an individual's ability to solve the seven mazes is

$$\sum_{m=1}^7 \left(area(m) \left(\frac{1}{d_m + 1} \right) \right) \quad (10.1.3.1-1)$$

where m is an index for the seven mazes and d_m is the shortest-path distance between the agent and the *finish* cell upon completion of its evaluation in maze m (this is set to zero if the maze was solved, even if the agent wanders away from the *finish* cell before stopping). Each maze's contribution to fitness is weighted by $area(m)$, the total number of cells in the maze, so that larger mazes are worth more than smaller ones. Evaluation in a given maze terminates when the allotted time expires (see 10.1.3) or when the agent visits the *finish* cell¹⁷.

¹⁷ An agent may make multiple moves during a single execution of its GP tree. As long as the *finish* cell has been visited, the evaluation will terminate after the GP tree execution completes. Agents that visit the *finish* cell and then move away from it and stop elsewhere are treated as though they had stopped on the *finish* cell.

10.1.3.2 The Preadapting Fitness Function

The fitness function for the preadaptation epoch measures an individual's ability to explore the seven mazes with no regard to whether or not the mazes are solved. The function is

$$\sum_{m=1}^7 (area(m)(v_m)) \quad (10.1.3.2-2)$$

where m is an index for the seven mazes, $area(m)$ is the number of cells in maze m , and v_m is the number of distinct cells visited by the agent during evaluation. Evaluation in a given maze terminates only when the allotted time expires.

10.1.4 The GP System

The parameters for the maze runner GP system are given in Table 10.1, below.

Setting, Parameter, or Option	Value
Function set	<p>{<i>BLANK</i>, <i>WALL</i>, <i>CURRENT</i>, <i>FORWARD</i>, <i>BACKWARD</i>, <i>RIGHT</i>, <i>LEFT</i>, <i>MOVE_FORWARD</i>, <i>MOVE_BACKWARD</i>, <i>GET</i>, <i>PAINT</i>, <i>FACE</i>, <i>IF_WALL</i>, <i>IF_EQUAL</i>, <i>DO</i>}</p> <p><i>BLANK</i> through <i>LEFT</i> are terminals that can be used to refer to particular cells or their contents, depending on the function receiving them as inputs. <i>MOVE_FORWARD</i> and <i>MOVE_BACKWARD</i> are arity-zero functions that cause the agent to attempt to move either in the direction it is facing or in the opposite direction, respectively.</p> <p><i>GET</i> takes a single argument indicating the cell to examine, and returns the terminal corresponding to that cell's contents.</p> <p><i>PAINT</i> takes a single argument which is the direction of an arrow to be painted on the current cell, and returns the contents of the cell before painting.</p> <p><i>FACE</i> takes a single argument which is the direction the agent should face, and returns the contents of the current cell.</p> <p><i>IF_WALL</i> takes three arguments. The first specifies a cell to examine. If that cell is a <i>wall</i> cell, the second argument is evaluated and return, else the third.</p> <p><i>IF_EQUAL</i> takes four arguments. The first two are evaluated. If they return equal values, the third is evaluated and returned, else the fourth.</p> <p><i>DO</i> takes two arguments. They are evaluated in order, and the return value of the second is returned.</p> <p>The return value at the root of an agent's GP tree is discarded.</p>

Halting condition	100 generations
Population size	50
Elitism	1 elite individual
Duplicates	Duplicates are permitted
Variation operators	50% crossover, 50% mutation Both operators are designed to limit the tree size to 500 nodes or less. Crossover does this by choosing a replacement subtree and then stepping deeper into the subtree until a subtree of acceptable size is found. Mutation does it by choosing a replacement subtree size uniformly at random between 1 and the largest acceptable size, and then generating such a subtree.
Selection	Tournaments of size 3
Population initialization	A tree size is chosen uniformly at random from 1 to 40. The tree starts a single terminal. It is then subjected to repeated calls to a leaf-expansion function until it meets or exceeds the desired size. If it exceeds it, a leaf-raising function is called repeatedly until it meets or falls below the desired size. This session of repeated growing and shrinking is repeated no more than five times. The final tree is no bigger than the desired size.

Table 10.1 Parameters of the maze solver GP system.

The **BLANK** and **WALL** terminals refer to the contents of a cell. **FORWARD**, **BACKWARD**, **RIGHT**, and **LEFT** are terminals that can be used either to refer to directions or painted arrows that point in those directions. Their use depends on the function interpreting them.

All directions are interpreted as being relative to the direction in which the agent is facing. The agent has neither a sense of any absolute direction, nor access to any information about its own position in the maze aside from being able to query the contents of nearby cells, all of which are seen relative to the agent's facing direction. For example, if an agent faces east while on a cell containing a south-pointing arrow, the agent will see the arrow as pointing to the right but will not know that south is to its right.

10.2 Results

Figure 10.1 shows the mean best fitness graphs for five different configurations of the maze runner GP system. The configurations are E0, E10, E20, E30, E40 and E60, with 0, 10, 20, 30, 40, and 60 generations of preadaptation, respectively. E0 therefore represents a normal non-exaptive configuration, and all others are exaptive.

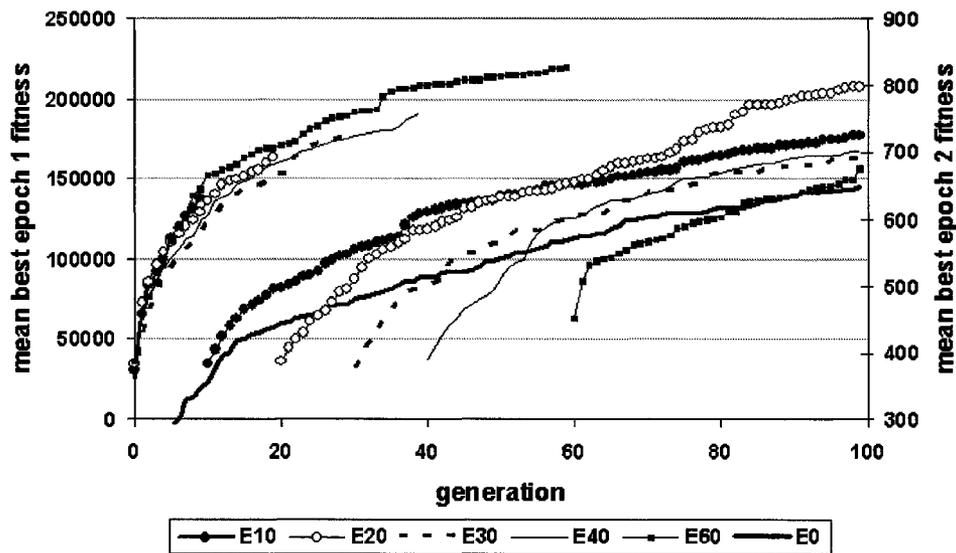


Figure 10.1 Mean best fitness vs. generation for the maze runner GP system.

Statistics for both fitness value and the number mazes solved are given in Table 10.2, below. The Mann-Whitney U test p values in the table are for comparisons between each configuration and E0.

Generations of preadaptation	Mean best fitness	Standard deviation	95% confidence interval	Mean mazes solved	Standard deviation	95% confidence interval	Samples	Mann-Whitney U test p value
0	648.6	463.7	± 60.6	2.80	0.99	± 0.12	225	
10	725.7	566.9	± 74.5	2.92	1.10	± 0.14	222	0.235
20	797.9	619.1	± 79.5	2.94	1.17	± 0.15	228	0.019

30	689.4	539.3	± 105.7	2.64	1.03	± 0.20	100	0.690
40	702.2	513.5	± 70.9	2.86	1.07	± 0.14	201	0.163
60	675.7	604.7	± 135.0	2.00	1.12	± 0.25	77	0.152

Table 10.2 Best fitness and mazes solved statistics for the maze runner GP system.

Data from the table are displayed graphically in Figure 10.2 and Figure 10.3, below.

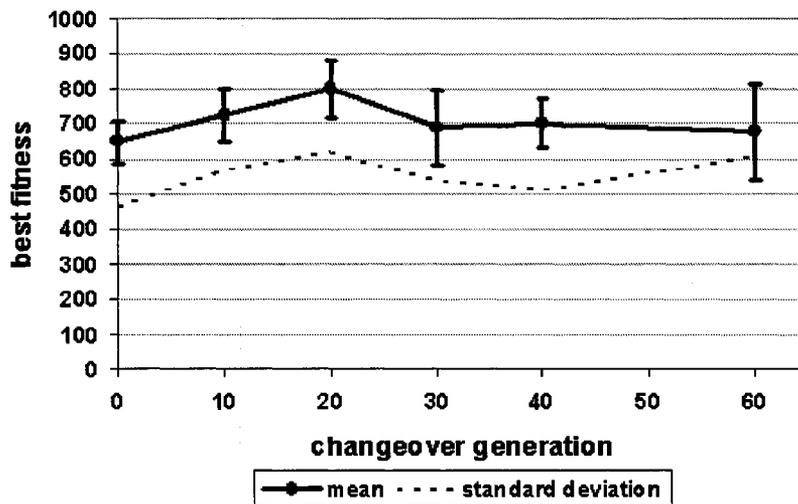


Figure 10.2 Best fitness and standard deviation vs. amount of preadaptation in the maze runner GP system.

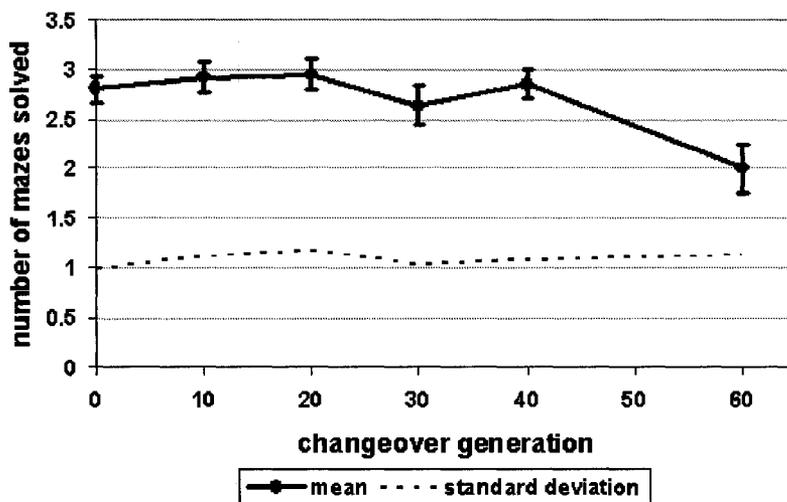


Figure 10.3 Number of mazes solved and standard deviation vs. amount of preadaptation in the maze runner GP system.

While all configurations solved the same number of mazes, on average, the exaptive configurations, particularly E20, have higher mean best fitness than the non-exaptive configuration.

10.3 Conclusions

Each of the five exaptive configurations produces a mean best fitness trajectory that climbs above that of the non-exaptive configuration. This can be seen in Figure 10.1. By the final generation, the E20 configuration, with its 20 generations of preadaptation, reaches the highest at a ~25% performance increase over E0 as measured by fitness function (10.1.3.1-1). Figure 10.2, plotting mean best fitness as a function of preadaptation time, shows that the preadapting fitness function goes a long way toward solving the final fitness task given that all configurations result in mean best fitness values between 600 and 800, even E60 in which most of each run is spent on the preadapting fitness function. Figure 10.3, however, shows little variation in the number of mazes solved, with the exception of E60 which solves less than the others. That less than half the mazes are solved on average (in fact, none of the runs managed to solve all seven mazes, though many solved six) reflects the difficulty of the problem posed to the GP system. The fitness function makes the task yet more difficult in that it rewards agents for *finishing* near the goal, but not for *passing* near it and then moving away. This complication imposes on the agents the necessity of tweaking a maze-exploration algorithm such that some mazes end up solved while those not solved have traversals that end as close to the goal as possible. This is not a trivial problem, since a single evolved algorithm must be used in all seven mazes. By first learning to maximize maze exploration, the preadapted populations seem better able to accomplish this difficult task.

Chapter 11 Conclusion

This research has developed a number of exaptive EC systems – four versions of an exaptive GA, three exaptive GP systems, and a hybrid GA / GP system – proposed and applied a technique for visually representing exaptive population dynamics in EC systems, and put forth a basic framework of categories for dividing exaptations into multiple types.

11.1 Summary of Contributions and Experiments

The previous chapters describe five systems and their variations looking at numerous aspects of exaptation and functional change in EC. Some involved only minor changes in function, such as the chromosome length penalty adjustments in the single niche exaptive GA. Some involved more abrupt and drastic functional changes, such as in the two niche and four niche exaptive GAs. Some involved exaptation of fixed structures. This was the case for all of the simple exaptive GA examples. The 2D navigator, and GP maze runner examples, on the other hand, involved exaptation of behaviors, control systems, or algorithms. 3DVCE involved both structures and behaviors for two learning tasks. Some of the five systems looked at performance advantages provided by exaptation, such as in the combination-fitness exaptive GA where the combined BG*LEA fitness function outperformed the LEA fitness function alone. The same focus on performance increases were goals of 3DVCE, the ‘piecewise’ and ‘derivative’ symbolic regression GP systems, the 2D navigator, and the maze runner GP system. Others looked at various effects and roles that exaptation could have in EC system behavior and dynamics, such as triggering

speciation in the two niche exaptive GA, or providing a mechanism for the evolution of interlocking complexity in the single niche exaptive GA.

The five example EC systems described above, and the particular aspects of exaptation and functional change that their ten experiments involved, are listed below in Table 11.1.

System	Aspects of Exaptation Involved
The Single Niche Exaptive GA	Gradual functional change as a mechanism for the evolution of interlocking complexity.
The Two Niche Exaptive GA	Exaptation as a means of triggering a population split that leads to speciation, and as a means of satisfying the minimum requirements of a fitness function that are not satisfied by randomly-generated solutions.
The Four Niche Exaptive GA	Exaptation as a means of satisfying the minimum requirements of a fitness function not satisfied by randomly-generated solutions, and as a means of sequentially stepping from one region of search space to another.
The Combination-Fitness Exaptive GA	Exaptation as a potential means of overcoming plateaus or fitness saddles. This scenario showed no clear signs that fitness plateau crossing or fitness saddle crossing played a significant role in the superior performance of the exaptive GA.
The Rough Terrain Traversal 3DVCE Configurations	Exaptation in preadapting a population of embodied virtual robots, and as a means of improving overall performance at a fitness task. This scenario requires more data before a conclusion can be drawn with statistical significance.
The Ball-Transporter 3DVCE Configurations	Exaptation in preadapting a population of embodied virtual robots, and as a means of improving overall performance by stepwise learning of a complex task. This requires more data before a conclusion can be drawn with statistical significance.
'Piecewise' Exaptive Symbolic Regression GP	Exaptation as a means of guiding a population toward a predetermined goal. Results were mixed and of small magnitude.
'Derivative' Exaptive Symbolic Regression in GP	Using an approximation of a target function's derivative to preadapt a population for improved evolution toward the true target function.
2D Navigator Exaptive GP	Exaptation used to improve efficiency in learning a complex behavior – navigating a 2D environment.
Maze Runner Exaptive GP	Exaptation used to improve efficiency in learning a complex behavior – running a set of grid mazes.

Table 11.1 Ten EC systems involving exaptation or functional change.

Table 11.2 applies the terminology and category suggestions from Chapter 5 to the exaptation examples from EC systems used in experiments described in the present

document, to those earlier systems described in Chapter 4, and to exaptation as it's understood in biology. The table shows a mix of both *strong* and *weak* exaptation (although this is the most subjective of all the category divisions), both *cumulative* and *replacement* exaptation, and both *compositional* and *non-compositional* exaptation. Many of the distinctions between types of exaptation described in Chapter 5 are to some degree subjective or fuzzy. For this reason, rationales for the category choices displayed in the table are supplied in Appendix D.

There are two observations about the table that are worthy of note. The first is that none of the EC systems exhibit the full range of exaptation behaviors exhibited in biology. The second, which is perhaps not surprising, is that exaptations in biology are *emergent*, whereas those in EC are usually *deliberate* and planned ahead of time. This need not necessarily be the case, especially in ALife systems like Enact [de Oliveira, 1994] where fitness depends on relative reproductive success, regardless of how that success is achieved. On the other hand, this is one aspect of the potential of exaptation that seems deficient in current non-ALife EC systems. Unforeseeable *emergent* exaptations in non-ALife systems may require greater expressive richness and complexity than most current systems allow. Despite having a whole phenotype evolved in such a (hypothetical richly expressive) system to meet some explicit criteria, modules and inter-dependent components of the whole could come and go over generations in a lineage, each serving some function within the whole that is emergent and not planned ahead of time, and that could change through evolution in coordination with other evolved modules. Perhaps, as algorithm complexity and processing power increase, these types of behaviors can be

incorporated into EC, bringing it closer to what [Banzhaf et al., 2006] call “Computational Evolution”.

	Types of Exaptation							
	Strong	Weak	Emergent	Deliberate	Cumulative	Replacement	Compositional	Non-compositional
Exaptive system								
Biological evolution	•	•	•		•	•	•	•
Systems from Chapter 4								
Spandrels in an ALife 2D Cellular Automaton [de Oliveira, 1994]	•	•	•			•		•
Preadaptation of Neural Networks in a Changing Environment [Lund & Parisi, 1995]		•		•		•		•
Incremental Approaches in the Evolution of Neural Network Robot Control Systems [Kodjabachian & Meyer, 1998a & b]	•			•	•		•	
Symbiotic Composition for Crossing Fitness Saddles [Waton & Pollack, 2001]		•		•		•	•	
Exaptation in Dynamic Landscapes [Fentress, 2005]		•		•		•		•
Gradual Functional Change for Evolving Differentiation in Gene Regulatory Networks [Knabe, Nehaniv, & Schilstra, 2006]	•			•	•			•
Systems from the present document								
The Single Niche Exaptive GA		•		•		•		•
The Two Niche Exaptive GA	•			•		•		•
The Four Niche Exaptive GA	•			•		•		•
The Combination-Fitness Exaptive GA	•			•		•		•
3D Virtual Creature Evolution	•			•		•		•
‘Piecewise’ Exaptive Symbolic Regression GP		•		•		•		•
‘Derivative’ Exaptive Symbolic Regression GP		•		•		•		•
2D Navigator Exaptive GP	•	•		•	•			•
Maze Runner Exaptive GP	•			•		•		•

Table 11.2 Types of exaptation exhibited in biological evolution, in the systems described in Chapter 4, and in the systems used for experiments in the present document.

The systems described in this thesis give a variety of perspectives on the possible roles of functional change in EC. Whether as a means of triggering speciation, exploring new

regions of the search space, or preadapting a population for improved performance, each scenario involves the co-opting of solutions evolved under one fitness function for use under another, whether the differences are those of degree or type. Together, these examples show some of the potential value of exaptation and functional change in EC.

Of central importance are the examples in which the performance of an exaptive system is compared with that of an otherwise identical non-exaptive system. The presentation of such examples supports the importance of exaptation within EC. The results presented in Chapter 6 through Chapter 10 were very mixed in terms of the support they lend to this view of exaptation.

The experiments showing the greatest degree of performance increase through the use of exaptation were those of the 2D GP Navigator and the combination-fitness exaptive GA from section 6.5. Nearly all exaptive configurations in the combination-fitness exaptive GA, regardless of the amount of preadaptation they involved, produced results superior to those of their non-exaptive counterpart. In the case of the E1200 configuration, with 1200 generations of preadaptation in 5000-generation runs, the increased performance peaked at a mean absolute fitness improvement of ~110 points for a fitness function whose range is [1..337]. An unexpected insight provided by the combination-fitness exaptive GA arose from the observation that too little preadaptation in an exaptive system can be detrimental. This was not the case in all exaptive systems examined, but the combination-fitness exaptive GA demonstrates such dynamics are a possibility. In practical terms this means that, should an EC practitioner notice a decrease in performance with some small

amount of preadaptation, it does not follow that more preadaptation is necessarily going to be detrimental.

An important point highlighted by the experiments involving the combination-fitness exaptive GA involves the potential for beneficially altering the evolutionary dynamics of an EC system by incorporating a secondary fitness function to augment the primary one. This is perhaps one of the most approachable strategies for further research into the use of exaptation and preadaptation in EC. The discovery of such mutually beneficial functions, especially for more conventional optimization problems such as TSP, SAT, and other NP-hard problems, may be very useful, especially if improvements can be found comparable in magnitude to those seen in the combination-fitness exaptive GA.

The results providing the least support for a beneficial role for exaptation in EC are those of the two 3DVCE experiments in Chapter 8 and those of the two symbolic regression systems described in Chapter 9. Due to the very high costs in terms of processing time, the number of independent runs collected for 3DVCE was too small to lend statistical significance to the results. The ‘piecewise’ exaptive symbolic regression GP system showed a very small but statistically significant improvement in mean best fitness performance for one of its two exaptive configurations, and a very small but significant decrease in performance for the other. The ‘derivative’ exaptive symbolic regression GP system showed small to moderate statistically significant performance improvements for only one of the five target functions examined.

The maze runner GP system showed moderate but statistically significant improvements in performance at its learning tasks using an exaptive approach.

The mixed results allow at least two conclusions to be drawn regarding a more general and widespread application of exaptive approaches in EC. The first is that it may not be feasible to predict ahead of time whether a particular exaptive augmentation for a given EC system is going to be successful or not. In 3DVCE, for example, the exaptive approaches seemed intuitively sound but the result, though not significant, appeared to be negative. And, as was shown by the combination-fitness exaptive GA, the outcome can be heavily dependent on the amount of preadaptation. On the other hand, both the 2D navigator's preadaptive tasks and that of the maze runner seemed intuitive and did yield an increase in performance. A second and perhaps more important conclusion is that, since a number of successes were clearly demonstrated in this dissertation, the use of exaptation in EC is a viable addition to the repertoire of techniques and ideas in the field, and is worthy of consideration for adoption and for further study and research.

11.2 Difficulties

The most severe hindrance to the adoption of exaptation-based approaches to problem domains in EC seems to stem from a key difference between EC and biology. It is the primary deficiency described in the previous section. It ties in with a related problem in biology, which is that of predicting specific evolutionary changes ahead of time. The issue was stated by Gould in [Gould, 2002], not so much as a source of frustration, but as a part of the nature of exaptation in evolution, and something that made the subject particularly engaging to him.

“This quirky historical character of major evolutionary change in particular lineages – thoroughly explainable after the fact, however unpredictable in

principle beforehand – constitutes the greatest fascination of the subject for many practitioners, myself included.”

The key difference is that, in biological evolution, exaptations are not planned ahead of time. Fitness is implicit, and any functional change can be favoured if it translates into greater average reproductive success for those individuals that exhibit it. The possibilities for functional change are as varied and diverse as the environment is complex. Biologists look back, after the fact, at a trail of evidence, and identify exaptations and functional changes long after they have occurred.

In most EC systems, the process is inverted. One begins with the ultimate desired functionality or criteria in mind. The would-be constructor of an exaptation-based EC system must then piece together a sequence of functions, some of which may seem, on the surface, to be unrelated, with an eye toward efficiency in guiding an evolving population toward a goal. This is not a straightforward process, as the mixed experimental results described above show, and general rules and principles governing how it might best be accomplished remain illusive.

In certain EC domains, particularly ALife domains, the process is not necessarily inverted in the manner described above. In many ALife systems fitness is truly implicit, and any functional changes permitted by the variation operators and the artificial environment can be favoured should they boost average relative reproductive success. In such systems it may be possible to study exaptation in a manner much closer to its study in biology. The more rich and complex the system, the more likely it should be to automatically give rise to exaptation.

For many biologists it is clear that exaptation plays a key role in evolution. Even among those, like [Dennett, 1995], who criticize the concept, exaptation is often seen as merely an unnecessary extra term describing what is already a well-understood aspect of normal Darwinian adaptation. Exaptation, then, even from the viewpoint of such critics, is all the more ubiquitous in biological evolution. Despite its importance and potential, its integration into the field of EC is, at the present time, very rudimentary.

11.3 Future Work

Three systems described in the present document seem to hold the most promise for fruitful future research into exaptation in EC. The first is the Simple Exaptive GA from Chapter 6. It is simultaneously the exaptive system showing the greatest degree of benefit for exaptation, the simplest in terms of its fitness functions, and the fastest in terms of execution time, making it possible to collect large quantities of data easily. One avenue for future work with the Simple Exaptive GA is to determine by what mechanism the combination-fitness exaptive GA is able to outperform the non-exaptive version of the same system. The experiments conducted in this document seem to indicate that the roles of fitness saddle and fitness plateau crossing are not the primary mechanism.

The second is the 2D navigator GP system, which is somewhat of a compromise between the complexity and potential of 3DVCE and the speed of the simple exaptive GA. An obvious next step is to continue the search for a learning task at which the non-exaptive configuration fails utterly. The second version of the 2D GP Navigator provided such limited feedback through the fitness function that it was initially supposed that the non-exaptive system would fail to collect more than one target except by chance. Instead, runs

tended to collect at least three, on average, and the fitness trajectories did not appear to have leveled off. The experiments involving the 2D GP Navigator could also be improved by tightening the constraints on what counts as a “fair” comparison between configurations. In both versions of the 2D GP Navigator, unequal evaluation times in various epochs made comparisons between configurations somewhat difficult.

The third is the 3DVCE system from Chapter 8. Despite the negative results obtained from the two exaptation experiments conducted in 3DVCE, it is arguably the richest in terms of its environment, search space, and possible tasks. One direction for future work with 3DVCE is to complete the experiments described in this document. Though the results appear negative, more data need to be collected before any real conclusions can be drawn. Another direction is to experiment with new preadaptation configurations beyond the two described in Chapter 8. These include small changes to the two configurations in that chapter, such as changes in epoch lengths and other parameters, as well as the introduction of entirely new fitness functions such as a 3D version of the target-depot task used in the 2D GP Navigator system. Some means of punishing creatures for jumping strategies that result in random direction changes could lead to a better method for using jumping to preadapt for terrain traversal. A workable simulated liquid environment would allow for exaptation experiments similar to some of the creature-evolution configurations used by Karl Sims, such as when his swimming creatures were moved onto land in two-epoch runs – one epoch in water followed by one epoch on land.

Non-exaptation-related 3DVCE experiment ideas include an exploration of the roles and effects of built-in oscillator primitives in creature control systems. Based on observations

of evolved creatures in action, it seems to be the case that such oscillator primitives create local maxima in the fitness landscapes in which creatures evolve fixed movement patterns that work well most of the time but do not involve adjustments and moment-to-moment reactions to sensor inputs. Those evolved without oscillators seem to produce different styles of locomotion, and seem to be less likely to get stuck when traveling over rough terrain, for example.

Appendix A. Examples of Exaptation Hypotheses in Biology

Table A.1, below, lists 26 examples of exaptation hypotheses from evolutionary biology.

A brief description is provided for each and references are listed for most.

1. α-lactalbumin	This is a protein produced by lactating mammals. It aides in the production of the milk sugar lactose. α -lactalbumin is hypothesized to be an altered copy of lysozyme, an enzyme that kills bacteria by damaging their cell walls, and is found in many bodily fluids such as tears, mucus, and saliva [Dickerson & Geis, 1969].
2. Avian wings	As birds evolved from theropod dinosaurs (a view now essentially unanimously accepted), their wings would originally have been forelimbs for terrestrial locomotion or grasping before being exapted to their current role in powered flight [Fastovsky & Weishampel, 1996]. Several possible intermediate stages predating their use for powered flight are hypothesized.
3. Bone	The material of bone may have originally functioned as the body's storage mechanism for phosphate and/or other mineral ions, and was later exapted to its role in structural support [Halstead, 1969].
4. Elephant trunks	Once just a nose and upper lip, the elephant's trunk now functions for numerous purposes such as grasping and manipulating objects in the environment, social interactions, and spraying water into the mouth or onto the body for bathing.
5. Feathers	Feathers have been hypothesized to have originally functioned in the catching of insects, and later being exapted for flight. A competing hypothesis is that feathers were adapted primarily for thermoregulation before being exapted for aerodynamics [Ostrom, 1976].
6. Female hyena genitals	<p>The genitals of female hyenas very closely resemble those of the males. This seems to enhance the females' fitness in what's called the <i>meeting ceremony</i> where two hyenas pair to inspect each other's genitals. It is hypothesized that this roll for the female genitalia is an exaptation of a non-aptation. The idea is that the resemblance to male genitalia is merely a side-effect of high androgen levels in the females, the genesis of which is tied to the social behavior and morphology of the females; they are larger than the males and dominate them within the social framework of hyena society.</p> <p>[Gould & Vrba, 1982] put forth this hypothesis, though they did not assert that it is correct, only that it had been an overlooked and perfectly plausible possibility for quite a long time due at least in part to what they perceived as the constraints of a conceptual framework in evolutionary biology devoid of a labeled distinction between aptations, adaptations, and exaptations. In particular they felt the conceptual framework was most lacking in that it had no term for co-opted non-aptations. The hyena hypothesis was meant to illustrate their point.</p>

7. Flat-headedness of the lizard <i>Holaspis guentheri</i>	
	[Arnold, 2002] argues that phylogenetic analyses indicate that the flat head of the lizard <i>Holaspis guentheri</i> first evolved for hunting in the tight spaces under tree bark, and was later exapted to the role it now plays in the gliding behavior of the lizard. The flat-headedness trait now serves both functions.
8. Gills	
	Though they now serve respiration functions, the gills of aquatic vertebrates are thought to have evolved from pharyngeal slits involved in the filter-feeding behavior of chordate ancestors.
9. Insect wings	
	There are a number of competing hypotheses regarding the origins of insect wings. As the fossil record for insect protowings is extremely lacking, the issue is far from resolved. [Kingsolver & Koehl, 1994] discusses the strengths and weaknesses of several of these hypotheses, most of which involve some form of exaptation.
10. Interpterygoid vacuity as housing for the eyeballs in some lizards	
	The interpterygoid vacuity is a cavity within the head, and a particular group of lizards, called cordylids, use this cavity to house their eyeballs when their heads are vertically compressed, such as when squeezing into tight crevices. As this cavity exists in many creatures, and predates the cordylids, it is not an adaptation for housing eyeballs, but has been exapted (and secondarily further adapted) to that role [Arnold, 1994].
11. Lens crystallins	
	Lens crystallins are water-soluble proteins used in eye lenses. There are many known cases where strong molecular and genetic evidence shows these proteins to have been exapted for use in the lens from a previous unrelated function. In many cases the original function is still maintained and the protein serves a dual role. Examples include the delta crystallin of chicken lenses, which is also the enzyme argininosuccinate lyase, the epsilon-Crystallin of ducks, which is the enzyme lactate dehydrogenase, the tau crystallin of turtles, which is the enzyme alpha-enolase, the mu-Crystallin in marsuipials, which is the enzyme ornithine cyclodeaminase [Piatigorsky, 1993], the eta-Crystallin in elephant shrews (which makes up more than 25% of the soluble protein in their lenses) is the enzyme cytoplasmic aldehyde dehydrogenase [Piatigorsky & Wistow, 1991], the omega-Crystallin of octopuses is also aldehyde dehydrogenase, and the S-Crystallin of squid is related to the enzyme glutathione S-transferase. [Piatigorsky, 2003] finds even more - the α A-Crystallin in vertebrates is related to a heat shock protein, the ζ -Crystallin in guinea pigs, camels, and llamas is related to the enzyme quinine oxidoreductase, the π -Crystallin in geckos is related to the enzyme glyceraldehyde-3-phosphate dehydrogenase, and the η -Crystallin in the elephant shrew is related to the enzyme retinaldehyde dehydrogenase. The family of lens crystallin proteins is rife with examples of exaptation.
12. Mammalian ear bones	
	The inner ear bones used by mammals for hearing are exaptations, though they've experienced considerable secondary adaptation. One of the best known series of transitional fossils shows these bones to have originally been parts of the reptilian jaw [Kardong, 2002].
13. Mammary Glands	

	<p>Mammary glands are thought to have evolved from immunoprotective glands that produced secretions that fought bacterial infections on the skin [Vorbach, Capecchi, & Penninger, 2006]. This hypothesis, though still speculative, ties in well with the evidence that the protein α-lactalbumin, found in breast milk, is an exapted version of the bacteria-killing enzyme lysozyme, and (perhaps to a lesser degree) with the fact that some remaining members of early offshoots in the mammalian line, such as platypuses, possess mammary glands but lack teats, instead simply secreting milk directly through the skin.</p>
14. Pandas' thumbs	
	<p>Originally a sesamoid bone in the wrist, modified to allow the giant panda (<i>Ailuropoda melanoleuca</i>) to hold bamboo, the thumb of the giant panda is not a thumb at all [Gould, 1980].</p> <p>Interestingly, this same exaptation occurred independently in a distantly related animal named the red panda (<i>Ailurus fulgens</i>), which is more closely related to skunks, raccoons, and weasels than true pandas. Its sesamoid bone has also been exapted and secondarily adapted to serve as an opposable thumb, and, coincidentally it seems, for processing the very same plant: bamboo [Salesa, 2006]. The red panda's thumb, as a further twist, seems to have itself been exapted for manipulating bamboo from an earlier opposable-digit function, as it was found to be present in the red panda's long-extinct and carnivorous (and therefore non-bamboo-eating) relative, <i>Simocyon batelleri</i>.</p> <p>These are both examples of exaptation, and together make a particularly striking example of <i>convergent evolution</i> (evolution hitting upon similar solutions independently).</p>
15. Penguin feathers	
	<p>The feathers of penguins represent a long chain of exaptations. Originally scales (although there is some dispute over that hypothesis in evolutionary biology), they were exapted, possibly for insulation, to become down and feathers, exapted again for flight, and then in the penguin line, exapted once again for both insulation and swimming in the ocean.</p>
16. Penguin wings	
	<p>As penguins are birds that have lost the ability to fly, their wings have been exapted and secondarily adapted for swimming.</p>
17. Post-synaptic neuron proteins	
	<p>A number of proteins that provide a scaffold underlying the receptors on post-synaptic membranes in neurons are now known to have evolved before the so-called <i>Cambrian explosion</i> in which the major animal groups and body-plans appeared in a very short period of geological time, even before they would have been co-opted into use in nervous systems [Sakarya et al., 2007].</p>
18. Repetitive DNA	
	<p>Repetitive segments of DNA in genomes may be non-aptations [Orgel & Crick, 1980], and [Doolittle & Sapienza, 1980] that can act and have acted as a source of raw materials for molecular exaptations.</p>
19. Shell space for egg-brooding in some snails	

	<p>Some species of snails brood their eggs within a space available in their shells. This space, called the <i>umbilicus</i>, seems to be a mere byproduct of the way shells develop as snails grow. Evidence strongly suggests that the snails that use the space for egg brooding evolved after those that possess it but don't use it, making this a spandrel that was exapted for brooding [Gould, 2002], [Andrews, Gangestad, & Matthews, 2000], and [Lindberg & Dobberten, 1981].</p>
<p>20. Shoulder hump decoration in <i>Megaloceros giganteus</i></p>	
	<p><i>Megaloceros giganteus</i> is also known as the Irish Elk (though it is technically a deer). It was the largest species of deer to have ever existed, and went extinct about 8000 years ago [Stuart et al., 2004]. It had the largest antlers of any deer, weighting up to 35kg on a 2kg skull. To support the mass of this structure, the spines of the vertebrae near the head were enlarged to provide a greater surface area for attachment of large ligaments and muscles. The resulting hump above the animal's shoulder was then exapted for mating displays by taking on colours and stripes as decoration.</p> <p>A twist in the story behind this example is that this type of colouration does not fossilize, and evidence for its existence comes from Cro-Magnon cave paintings of these animals. This example is perhaps more speculative and less firmly established than some of the others, but no less interesting [Gould, 2002 p. 1260].</p>
<p>21. Suborbital foramen as housing for the eyeballs in some lizards</p>	
	<p>The suborbital foramen is a cavity which evolved for other purposes but has been exapted in certain lizards (scincids and lacertids) to house the eyes when the head is squeezed into thin crevices [Arnold, 1994]. Like the exaptation of the interpterygoid vacuity in the cordylid lizards, above, this cavity was exapted to the same function in these lizards. It's an alternative exaptive solution to the same general problem.</p>
<p>22. Sutures in mammalian skulls</p>	
	<p>In mammals, skull sutures serve to aid in parturition (the birthing process), and may be indispensable for that purpose. However, as they exist in the skulls of birds and reptiles alike, and as birds and reptiles have no need of them since their young merely escape from eggs, they most likely exist as a consequence or side-effect of the way skull bones grow alongside one another. In their useful role in mammalian birth, they can be considered exaptations – specifically, exapted spandrels [Gould & Vrba, 1982]. This point was famously made even by [Darwin, 1859] himself, who realized that their utility and origins were functionally unrelated.</p>
<p>23. Swim-bladders</p>	
	<p>The swim bladders present in many fish seem to have originally been lungs [Liem, 1988]. Interestingly, [Darwin, 1859] also hypothesized this particular exaptation, but he had it backwards since swim bladders were more common in fish than lungs, and because air-breathing tetrapods evolved later than fish.</p>
<p>24. Tetrapod limbs</p>	
	<p>As tetrapods evolved from fish, so it seems likely that their limbs would have been exaptations to a function involved with terrestrial locomotion, from an original swimming function.</p>
<p>25. Vertebrate jaws</p>	
	<p>It is thought that the jaws of early vertebrates evolved from skeletal rods called <i>pharyngeal arches</i>. The arches originally served to keep pharyngeal gill slits open in an ancient ancestor.</p>

26. Wing mantling in the Black Heron/Egret, <i>Egretta ardesiaca</i>	
	The Black Egret “mantles” its wings over water (spreads them out to create a large shadow over the water’s surface). This allows it to more easily see its prey below the surface [McLachlan & Liversidge 1978]. The wings differ little from closely-related species whose members don’t hunt this way, and the behavior seems to have a genetic component.

Table A.1 Examples of exaptation hypotheses in biology.

Appendix B. Details of the 3DVCE System

This appendix provides an in-depth and detailed description of the 3DVCE system from Chapter 6. This includes both fixed and configurable system parameters and evolution parameters, as well as numerous details regarding the GA, its variation operators, creature phenotypes and genotypes, specific experimental settings, and more.

For specific details on the inner workings and algorithms behind the ODE rigid-body physics engine itself, the interested reader is directed to [Smith, 2007].

B.1. The Main GA Loop

In the 3DVCE system, a particular experimental setup is referred to as an “evolution”. Evolutions consist of one or more independent GA runs, where each run is identical in terms of settings like population size, tournament size, mutation rates, and so on. The flow-chart in Figure B.1 depicts the sequence of high-level operations called by the main GA loop when executing an evolution.

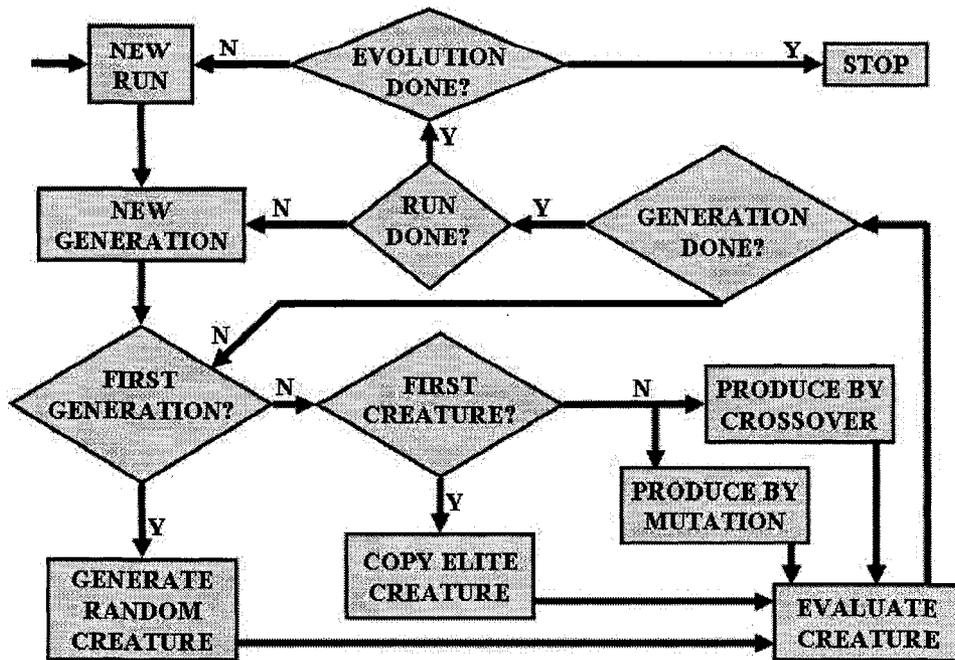


Figure B.1 A high-level flow chart for the 3DVCE main GA loop.

All creatures in the first generation of any run are produced from randomly-initialized genomes. Creatures in generations other than the first are produced either by mutation or crossover of parents selected by tournament from the previous generation. The sole exception is the first creature in such generations, which is always a copy of the best (or elite) creature from the previous generation.

B.1.1. The Halting Condition

When an evolution configuration is being set, the user has the option to specify a halting condition only in terms of the number of generations and the number of runs. There are three possible ways to set the halting condition.

- 1.If one sets the number of generations to be unlimited, the number of runs specified will be ignored and only a single run will be performed in the evolution. Such a run will terminate only when the user intervenes to stop the evolution.
- 2.If one specifies a finite number of generations per run while setting the number of runs to be unlimited, 3DVCE will halt each run at the completion of its final generation and then begin a new one with the same settings. This configuration will also run until the user intervenes to stop it.
- 3.If one specifies both a finite number of generations and a finite number of runs, the system will automatically terminate the evolution after completing the specified number of runs.

In order to collect as much data as possible in the time available, all 3DVCE experiments described in this work used the second option – finite generations, and unlimited runs. This allowed for collection of data from identically-configured independent runs during as much allotted CPU time as could be made available.

B.1.2. Elitism

3DVCE uses single-individual elitism, meaning that the creature with the highest fitness in any given generation is copied intact to the next generation without mutation. Such elite creatures are also re-evaluated during the new generation. All creatures, including the elite, keep both a tally of the number of fitness evaluations they've been subjected to, as well as the sum of all fitness measures from those evaluations. This allows creatures' fitness values to equal an average over all of their evaluations, although only elite individuals can accumulate more than the normal number of evaluations. In this way, a

creature that becomes elite by obtaining a very high fitness score which does not accurately reflect its true quality is unlikely to remain elite for long as repeated evaluations over subsequent generations will tend to pull its fitness closer to the true average.

The use of single-individual elitism is hard-coded into the system. At the time of this writing there is no feature for enabling, disabling, or modifying this aspect of the GA by the user.

B.2. Fixed System Parameters

The following subsections describe a number of parameters and options in the 3DVCE system that, like single-individual elitism, above, are help fixed and constant at all times. These are divided into three categories: settings that affect the OPAL/ODE physics engine, settings that affect creature control systems, and settings that affect creature morphology. Although some of these settings are loaded at program start up time from a text file, and are therefore, in principle, under user control, there are no GUI features in the application for specifying or changing them.

B.2.1. Physics Engine Settings

The following settings are based on a small amount of initial testing and experimentation. They do not reflect any attempt to make the physics engine's behavior match any particular real-world scenario or materials or motors, as has been the case in other virtual creature evolution projects, such as [Lipson & Pollack, 2000].

- Gravity is set to a strength of 120.0.
- The step size for the simulator's integrator is 0.001.
- The *solverAccuracy* value in OPAL is set to its default (*SOLVER_ACCURACY_MEDIUM*).

B.2.2. Creature Control System Settings

The following fixed settings values dictate or constrain various aspects of 3DVCE creature control systems.

- The simulated time interval between control system updates during creature evaluations is 0.01s.
- The maximum number of nodes in any GP tree is 1000 (note that at the time of this writing it is possible for this limit to be exceeded by the special tree crossover operator).
- The maximum number of nodes in any randomly-generated GP tree during population initialization is 10 (note that this is a soft limit in that it can potentially be exceeded in randomly-generated trees – see B.7.2). The small initial tree size was chosen after observing that such small randomly-generated trees tended to produce more subjectively interesting creature behaviors.
- The maximum scaling value (a parameter determining wavelength) for creatures' internal sine wave generators is 10 (see B.3.2.1).
- At both the creature level, and at the level of individual body segments, a state value is repeatedly updated by the control system. The default segment-level and creature-level state value, before the first control system update is completed, is 0.0.

- An output value is also updated by the control system at both the creature level and at the level of individual body segments. The default segment-level and creature-level output value, before the first control system update is completed, is 0.0.
- The default value returned by body segment contact sensors before the first control system update is completed is 0.0.

B.2.3. Creature Morphology Settings

The following fixed settings values dictate or constrain aspects of 3DVCE creature morphology.

- Body segment cuboid dimensions range from 0.4 to 10.0.
- Body segment cuboid density is 0.01.
- The servo motor restore speed constant is 0.05. This value is not used directly as the restore speed setting for creature servo motors. Instead, it is used in conjunction with the output of the body segment's torque GP tree to update servo motor restore speed settings at every control system cycle. The actual servo motor restore speed setting adopted during a given simulation cycle is computed as

$$r \left(1 + \frac{t}{2} \right) \quad (\text{B.2.3-1})$$

where r is the restore speed constant and t is the output of the torque GP tree (a value ranging from -1.0 to +1.0). Instead of enforcing a trade-off between maximum permissible servo speed and maximum permissible torque, both aspects of servo motor performance increase or decrease together, depending on the t signal received from the control system at each cycle.

- The maximum permissible cuboid velocity is 80,000. This limit is intended to help the system avoid the numerical explosions that can arise from time to time during creature evolution, as described in [Shim, Kim, & Kim, 2004]. Should any cuboid in the body of any creature exceed this limit, the creature is immediately terminated and assigned a fitness of zero. This discourages the evolution of undesirable creatures that take advantage of weaknesses in the physics engine.
- The joints between body segments are set to break if they experience torques beyond a given threshold. This serves the same purpose as the cuboid velocity limit, above. So-called numerical explosions can cause joints to experience exceptionally high torques. This threshold is a means to detect and penalize such occurrences. The breaking threshold value for a given joint is computed as

$$2.0 \times 10^6 (m_1 + m_2) \quad (\text{B.2.3-2})$$

where m_1 and m_2 are the masses of the two segments connected by the joint. This formula allows joints between larger segments to sustain correspondingly larger torques. The specific form of the equation and the coefficient of 2.0×10^6 was determined through experimentation during development of the 3DVCE system.

- The maximum allowable torque that can be exerted by a servo motor in its effort to reach its desired angle is computed as

$$\left[5.45 \times 10^3 (m_1 + m_2) + 7.0 \times 10^2 (m_1 + m_2)^2 \right] (1 + t) \quad (\text{B.2.3-3})$$

where m_1 and m_2 are the masses of the two segments connected by the joint, and t is the output of the torque GP tree of the body segment associated with the servo. The term in square brackets is a joint-specific constant that is calculated and stored when

the joint in first instantiated. Like the joint-breaking threshold formula, above, this formula is the result of some initial experimentation during system development, based on a reduction in the frequency of numerical explosions and on the subjective aesthetics of the resulting creature behavior. It is not intended to accurately approximate the physics of real-world muscles where strength increases linearly with cross-sectional area.

- The body segment scale variability constant is set to 0.5. This sets the range of possible values for the scaling factor gene in body segments. The scaling factor is a real-value which shrinks or expands entire subtrees in the creature body. Each step along the path from the body's root segment to a leaf segment results in a multiplication of the scaling value applied to body segment dimensions during embryogenesis. This allows subtrees in the body in grow or shrink with distance from the root – a phenomenon familiar in biological organisms, such as the branching patterns of some types of plants, or in circulatory systems. The scaling factor gene in the genetic specification of each segment type is constrained to lie within the range $[1.0 - v \dots 1.0 + v]$ where v is the variability constant.
- The number of branch specifications per body segment type in the genome is 5. This specifies that five branches are genetically encoded for each type of body segment. Each of these may be either enabled or disabled in any given segment type, and each may be either mirrored or not mirrored (see B.3.2.3.1). This allows for a creature body segment to have anywhere from 0 to 10 actual branches extending from it in the phenotype.

- The number of available body segment colours is 10. These colours are stored in a global array and are simply indexed by the colour gene for each segment type. Colours have no effect on the physics simulation and are therefore invisible to selection, despite being genetically determined.
- Each joint has two degrees of rotational freedom (see B.3.1.1). The rotation of a body segment about each of the two joint axes is constrained to stay within the range -110.0 degrees to +110.0 degrees. When a joint is first instantiated, the relative orientation of the two segments it joins, at the moment of instantiation, is interpreted as the zero-degree position for each axis of rotation. The desired joint angle fed to the joint's servo motor at each cycle of the control system is set to

$$110.0(a) \quad (B.2.3-4)$$

where a is the output of the angle GP tree associated with the segment, for a given axis of rotation. a is a value between -1.0 and +1.0.

- The branch specification y-axis angular offset range is -15 degrees to +15 degrees. A branch is initially oriented such that its local z-axis points directly away from the centre of its parent segment's cuboid (calculated as if the parent segment's cuboid were truly cube-shaped, so all angular offsets are multiples of 45 degrees). The y-axis angular offset gene from the branch specification is then used to pivot the branch on its joint anchor point, about its local y-axis, by a small amount within this fixed range.
- The maximum value for the max repeats gene for a body segment type is set to 12. The gene itself can specify any number of repeats between zero and this hard limit. The morphology of the actual creature may have any number of repeats of a given segment type (along any root-to-leaf path in the body) up to the number specified by

the gene. The number of segments of the same type along a root-to-leaf path will also depend on whether the segment type is expressed at all in the phenotype, whether the relevant branches are enabled that would cause the type to be repeated, and whether the process of embryogenesis truncates body growth because of segment intersections, segment depth, or total segment counts.

B.3. Creatures

This section provides a detailed account of the 3DVCE creature genotypes, and phenotypes. This includes all aspects of both morphology and control systems, as well as the embryogenic process that translates genotype to phenotype.

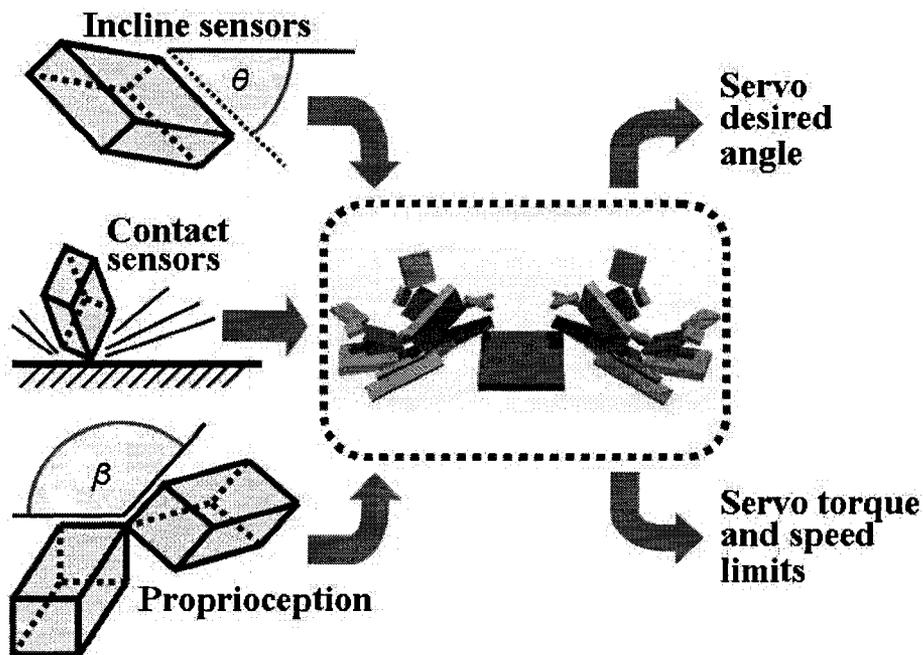


Figure B.2 Creature inputs and outputs in 3DVCE.

Figure B.2 shows the sensors and effectors of creatures in 3DVCE. Creatures have three senses through which they can obtain information about the world and about their own bodies. Incline sensors provide information to creatures about the tilt of their body parts.

Contact sensors indicate whether body parts are in contact with objects that are not part of the body. Proprioceptors provide information about the current angles at the joints that connect body parts. The creature control system sends outputs to the servo motors at each joint, controlling their desired angles, maximum torque values, and maximum speeds.

B.3.1. Phenotypes

Each creature is a finite-depth tree of rigid cuboids (technically, “rectangular parallelepipeds”) of a fixed and uniform density. Each is connected to its parent in the tree by a joint. Associated with each cuboid (or “body segment”) are several types of sensors providing information about the body and its relationship to the environment. Associated with each joint in the body are two servo motors, one for each degree of rotational freedom in the joint.

This section divides the topic of creature phenotypes into three areas: the structure of the body itself, the manner in which it’s constructed when a creature is instantiated (termed “embryogenesis”), and a mathematical function that maps a creature’s body to a single real-valued number used as a measure of creature size.

B.3.1.1. Body Structure

Each creature’s body is composed of a number of cuboids called “body segments”. The segments are arranged in a tree structure with each segment connected to its parent by a type of physical constraint known as a universal joint. Segments are divided into groups called types, with the properties of each type encoded in the genotype (see B.3.2). Each cuboid is a rigid solid with uniform density. The density value is a fixed constant of the

3DVCE system (see B.2.3). Simulation of the dynamics of the segments (friction, collisions, linear and angular momentum, acceleration, velocity and so on) is handled the by the physics engine.

At each joint in the body, there are two servo motors, one for each degree of rotational freedom in the joint. Servo motors use built-in PID (Proportional-Integral-Derivative) controllers to automatically monitor joint angles and regulate torques in an attempt to orient the jointed segments toward some desired relative angle.

Figure B.3 shows a schematic diagram depicting how a real-world universal joint might be constructed. Universal joints have two axes of rotational freedom. In 3DVCE these axes are always orthogonal and always intersect at a single shared point which lies on the surface of the two segments being joined. This point is called the joint anchor. The two segments can pivot about the joint anchor, but cannot twist.

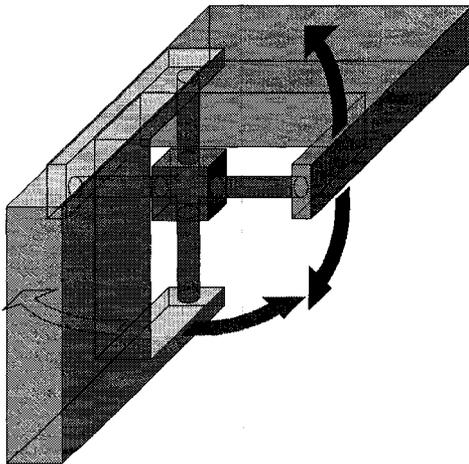


Figure B.3 A schematic diagram of a universal joint.

The diagram in Figure B.3 shows a group of specially-shaped solid parts interconnecting and interacting to give rise to the desired rotational constraints of the universal joint. In

the OPAL / ODE physics engine used in 3DVCE, however, universal joints take the form of mere constraints on object positions, and do not involve any extra simulated group of solid objects aside from the two segments being joined. The positional constraints are enforced directly by the physics engine, although they are continually violated. These violations are usually very small, and are constantly compensated for by the normal operation of the engine. In extreme cases, when numerical errors accumulate in a phenomenon called numerical explosions [Chaumont, Egli, & Adami, 2007], these constraints can be severely violated (the creatures explode!). Various methods of detecting, controlling, and discouraging numerical explosions have been implemented to get around this problem (see B.2.3).

B.3.1.2. Embryogenesis

The process by which the creature genotype is decoded and developed into a final creature phenotype is here labeled “embryogenesis”. The similarities with the namesake process in biology are fairly loose, however. The final phenotype is influenced by three factors: the creature’s genotype, a developmental tree-growth process, and several parameters of the 3DVCE system and its evolution configuration.

A creature’s genotype dictates a potentially-recursive (both indirectly and directly recursive) branching pattern. The genome encodes properties for a fixed number of segment types, and associated with each type is a fixed number of branch specifications that dictate the way in which a segment of a given type sprouts branches of other segment types. The encoded information also specifies the scaling, positioning, and other aspects of the branching. An example of this is depicted in Figure B.4 where the tree encoded in

the genome is drawn to a depth of six segments. In that example, segments of type 0 have a single branch of type 4. Segments of type 4 have four branches of types 1, 4, 0, and 5. Segments of type 1 are the base of a chain of three segments of types 1, 2, and 3. Types 3 and 5 do not branch.

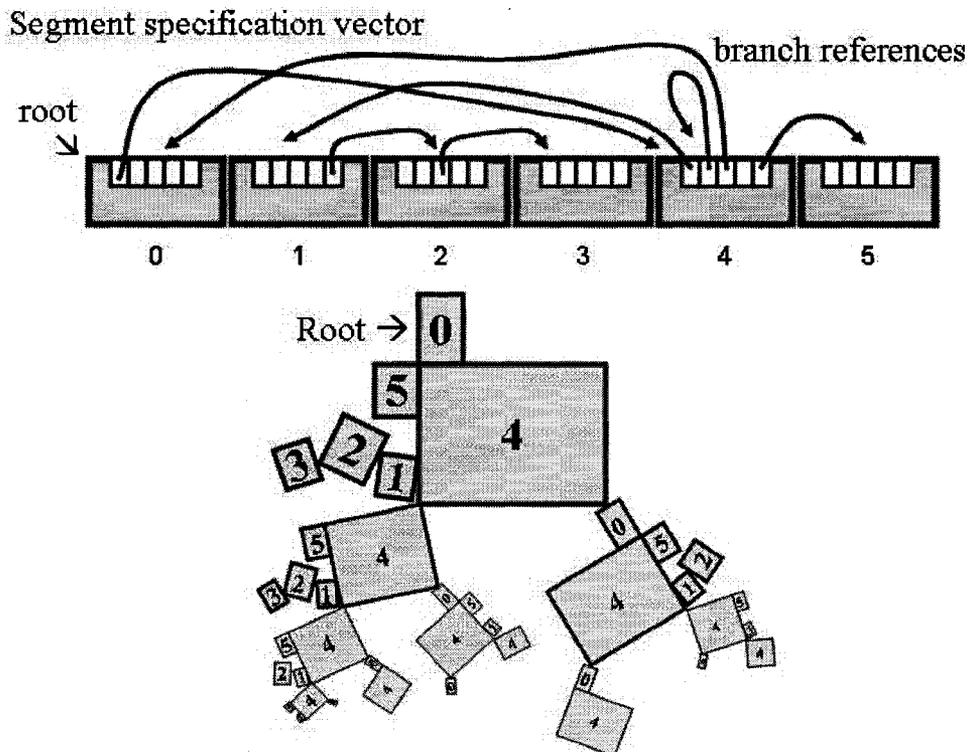


Figure B.4 An illustration in two dimensions of the way branch references and segment specifications dictate a tree structure.

The following constraints limit the growth of such recursive trees.

- **Segment type repetition limits:**

The genome encodes a limit on the number of times that a segment of a given type can appear along any root-to-leaf path in the creature's body. This limit can vary from one segment type to another within the genome and can range in value between 0 and 12. Another gene specifies the type of segment that continues the growth when this

limit is reached. In this way, a recursive tree structure can abruptly change its branching pattern after a number of repetitions of segments of the same type. This may allow for the evolution of structures roughly equivalent to hands or feet at the ends of segment chains forming appendages, for example. If a segment of a given type specifies that the same type should continue when repeats are exhausted, growth will simply terminate at that point in the tree. Alternatively, if the type of segment chosen to continue when repeats are exhausted has already had its own type's repeats exhausted, this will also terminate growth.

- **Segment intersections:**

The body tree of a creature is grown in a depth-first manner. The root segment is first constructed, and then the entire subtree of its first enabled branch specification is constructed, followed by the next enabled branch, and so on. This process is recursive, so each subtree is constructed in the same manner. When a new segment has been instantiated the physics engine is queried to obtain a list of all objects that intersect the segment. A segment is permitted to intersect its own parent segment since this is necessary anyway to allow for the full range of joint motion during simulation. However, if a newly-instantiated segment is found to be intersecting some other previously-instantiated part of the creature's body, other than its own parent segment, the newly-instantiated segment will be deleted and none of its subtrees will be constructed. This truncation of growth at intersecting branches ensures that a newly-instantiated creature will not find itself suddenly experiencing the enormous repulsion forces that would be generated by the physics engine attempting to resolve

and remove such intersections. It can also cause a disruption of symmetry in some bilaterally-symmetric creatures or their limbs.

- **Bounds on segment depth and count:**

There are two bounds on creature bodies that act together to constrain the growth of the creature during embryogenesis. The first is a hard limit on the total number of body segments allowed. The second limit is a bound on the depth of the body's tree structure (the number of segments along any root-to-leaf path). This second bound is iteratively adjusted until both it and the segment count limit are satisfied. When the depth-first construction of the body reaches this second limit, it will not continue to grow deeper subtrees, even if the genome dictates it. Both of these bounds are supplied by the user when an evolution is configured. The iterative adjustment proceeds as follows. First, the creature is instantiated using the depth bound provided in the evolution configuration. If, during that process, the total number of body segments exceeds the segment count limit, the body is deleted, the depth bound is decreased, and the body is constructed again. This process will repeat, decreasing the depth bound each time, until the segment count falls within the specific limit.

The final creature body produced during embryogenesis is guaranteed to be free of segment intersections, except those between segments directly connect by a join. It should be noted that in such cases the physics engine will not generate repulsion forces to resolve the intersection. Segments connected by a common joint are permitted by the physics engine to pass unimpeded through one another. The final creature produced is also guaranteed to be within the segment depth and count bounds specified in the evolution configuration.

There are a number of values encoded in the genome that determine the positions of branches in relation to their parent segments, as well as their sizes, proportions, orientations, and so on. These are described below, in B.3.2.

B.3.1.3. Creature Size

Certain methods involved in fitness calculation (see B.4.3), and in the generation of terrain in the rough terrain environment (see B.4.1.2) require a measure of creature size. This is needed in order to scale various values in compensation for the overall size of a creature's body. For instance, when measuring distance traveled by a creature, it may be desirable to require larger creatures to travel farther than smaller ones to attain equivalent fitness scores. The formula used for this measurement, called simply the creature size, is computed as

$$size(C) = \sqrt[3]{\sum_{i=1}^{N(C)} v(c_i)} \quad (B.3.1.3-1)$$

where C is the creature, $N(C)$ is the number of cuboids in the body of creature C , c_i is cuboid i , and $v(c_i)$ is the volume of cuboid c_i .

In simple terms, this is the length that would be required to make a cube whose volume would be equal to the total volume of all body segments in the creature's body.

B.3.2. Genotypes

A creature's genotype encodes details of both its morphology and its control system. It consists of a large number of real-valued numbers, integers, and Booleans as well as a number of GP trees. Figure B.5, below, shows a hierarchical breakdown of all the

components that make up the genome. The genotype consists of a sine wave generator, two creature-level GP trees (depicted in the figure as triangles), and a fixed-length vector of segment specifications. Each segment specification consists of its own sine wave generator, six reals, three integers, four GP trees, and a fixed-length vector of branch specifications. Each branch specification consists of one real, five integers, and three Boolean values.

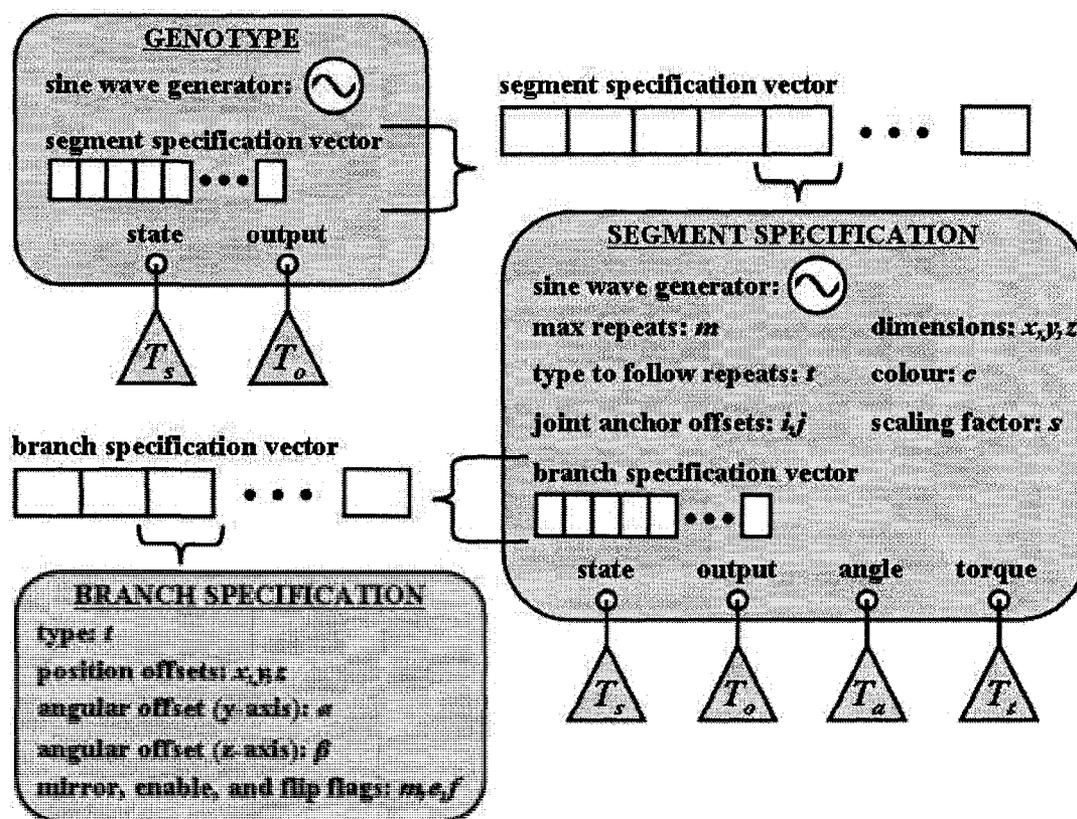


Figure B.5 A hierarchical breakdown of the 3DVCE creature genotype.

This section describes each of these facets of the genome in detail, beginning with the sine wave generators, followed by the GP trees, and finally the segment and branch specifications.

B.3.2.1. Sine Wave Generators

Creatures possess several oscillators that produce sine wave signals that vary with simulation time. The signals have a fixed amplitude of 1.0 with phase and wavelength parameters encoded in the genome.

The use of oscillators in creature control systems is meant to contribute to the sorts of rhythmic limb movements needed for sustain locomotion. These kinds of intrinsic pattern generators are thought to have their counterparts in biology as well. [Delcomyn, 1985] describes the role of such internal pattern generators in the locomotion of some arthropods.

Each sine wave generator has two real-valued genetic parameters. The first is the start angle parameter, which amounts to a phase shift. The second is the scaling value parameter, which dictates the wavelength of the generated signal. The start angle parameter ranges from 0 to 2π , and the scaling value ranges from -10.0 to +10.0.

Sine wave generators can only be accessed within the GP trees of creature control systems. Specifically, through the LSINE and GSINE GP function nodes. Each creature has a single sine wave generator which is available to any creature-level GP tree in the control system as well as any segment-level GP tree. Each body segment also has its own sine wave generator which is directly available only to GP trees within that segment. The LSINE function is found only in segment-level GP trees and provides access to the sine wave generator stored at the segment. The GSINE function can be found in any GP tree in the control system, whether at the creature level or at the segment level. It and provides

access to the creature-level sine wave generator. The “L” and “G” stand for “local” and “global”, respectively.

The LSINE and GSINE functions take only a single parameter (arity = 1), and their output is computed as

$$\sin(a + (st) + (2\pi x)) \quad (\text{B.3.2.1-1})$$

where x is the input to the function, a is the start angle value of the generator, s is the scaling value parameter, and t is the amount of simulated time that has passed since the creature was first instantiated in the 3D environment.

B.3.2.2. GP Trees

While most other 3D virtual creature evolution projects have based their creature control systems on some form of artificial neural network, the 3DVCE control systems are based on GP. Each creature is controlled by a number of different GP trees which are stored in the genome.

Associated with the creature as a whole are two GP trees. The first is the output tree which computes, during each control system update cycle, a value which is available to all other GP trees in the creature or any of its segments via the COUTPUT terminal node. The other, called the state tree, operates in very much the same manner. It computes, during each control system update cycle, a value which is available to all GP trees via the GSTATE terminal node. Though the terms “output” and “state” carry different semantic connotation, these two trees play essentially identical roles. One could just as easily label them state tree one and state tree two, or output tree one and output tree two.

Associated with each body segment are four GP trees. A creature with b body segments will therefore possess a control system consisting of $4b+2$ trees, four for each segment and two for the whole creature. The structure of the four trees at the segment level is encoded in the segment specification genotype for the segment. Thus all segments of the same type carry identical GP trees, although tree outputs can differ between two segments of the same type since the inputs from sensors and adjoining segments will be different. The four trees at the segment level are the output, state, angle, and torque trees, described below.

The output and state trees operate in the same manner as the output and state trees at the creature level, but with outputs only directly available to the specific segment possessing the trees.

The angle tree determines the desired joint angle values which are sent to the servo motors. The servo motors, in turn, control the torques applied to the two axes of rotational freedom in the segment's joint. The desired angle value sent to the servo is simply the output of the angle tree multiplied by 110.0 (see (B.2.3-4)). This computes a valid angle value within the fixed range of -110 degrees to $+110$ degrees allowed for joints since all GP trees in 3DVCE produce output values between -1.0 and $+1.0$.

The torque tree's output determines both the maximum amount of torque and the maximum speed that the servo will be permitted to use in its attempt to reach the desired angle given by the angle tree (see (B.2.3-1) and (B.2.3-3)).

There is only one angle tree and one torque tree per body segment, but each joint has two degrees of rotational freedom. The control system can specify different desired angles, restore speeds, and torques for the two rotational axes via use of the AXIS terminal node in the angle and torque GP trees. The AXIS node returns the value 1.0 whenever computations are being performed for the servo controlling the second axis of rotation on a joint, and returns 0.0 at all other times (i.e. when controlling the first axis of rotation or when used in the output and state trees). This method of allowing differentiated control of each axis was chosen in the hope that it would facilitate the evolution of coordinated control of the two axes by having both servos' inputs computed from the same GP tree. Depending on the structure of the angle and torque trees, and their use of the AXIS node, the two servos could be controlled in a manner that is tightly synchronized and coordinated, entirely independent, or anything in between. Tightly synchronized and coordinated control of both axes simultaneously could also arise via the co-evolution of two independent GP trees, but it is hoped that this may be more strongly encouraged and readily evolved when control comes from a single tree instead.

Since a creature's root segment has no associated joint, as it has no parent segment of its own to which to join, angle and torque trees serve no function in root segments.

Table B.1 lists all 23 GP nodes types in the 3DVCE system. Some types of nodes are only available to segment-level GP trees, such as those that access the various sensors available to a segment. The GP nodes vary in arity from 0 to 3. Nodes of arity zero are terminals in the GP tree. Nodes of higher arity have one subtree for each of their required

inputs. The table gives a brief explanation of the input(s) and output of each node type, and the functions they compute.

Name	Arity	Creature Level or Segment Level	Output
RND	0	Both	A fixed but random value between -1 and +1
GSTATE	0	Both	The creature's state value
LSTATE	0	Segment	The segment's state value
ROUTPUT	0	Both	The root segment's output value
POUTPUT	0	Segment	The parent segment's output value
COUTPUT	0	Both	The creature's output value
CAVG	0	Segment	The average of all child segment output values
CSENSOR	0	Segment	+1 if segment or any below it in body-tree is contacting a non-body object in the environment -1 otherwise
ISENSOR1	0	Segment	The output of the segment's first incline sensor, normalized to between -1 and +1
ISENSOR2	0	Segment	The output of the segment's second incline sensor, normalized to between -1 and +1
ANGLE1	0	Segment	The segment joint's angle about its 1 st axis of rotation, normalized to between -1 and +1 0 if segment is root
ANGLE2	0	Segment	The segment joint's angle about its 2 nd axis of rotation, normalized to between -1 and +1 0 if segment is root
MIRROR	0	Segment	-1 if the segment is a mirrored copy of another 1 otherwise
AXIS	0	Segment	1 when computing torque or desired angle for segment joint's second axis of rotation 0 otherwise
GSINE	1	Both	Output of creature sine wave generator, with angle shifted by input value: $\sin(\text{start_angle} + (\text{time_since_instantiation} * \text{scale_value}) + (\text{input} * 2\pi))$
LSINE	1	Segment	Output of segment sine wave generator, shifted in the same manner as GSINE
SSQR	1	Both	The signed square of its input Example: SSQR(-0.5) = -0.25
SSQRT	1	Both	The signed square root of its input Example: SSQRT(-0.25) = -0.5
NEG	1	Both	Reverses the sign of its input
MULT	2	Both	The product of its inputs
BADD	2	Both	Bounded addition of inputs $BADD(x,y) = \max(-1, \min(+1, x+y))$
BSUB	2	Both	Bounded subtraction of inputs

			$BSUB(x,y) = \max(-1, \min(+1, x-y))$
IF	3	Both	Conditional $IF(x,y,z) = \text{if } x < 0.0 \text{ then } y, \text{ else } z$

Table B.1 GP node types in 3DVCE.

GP trees in 3DVCE are almost always constrained to possess no more than 1000 nodes. At the time of this writing, this limit is not universally adhered to by all tree-manipulation methods. The special tree mutation operator can potentially violate this constraint. All other GP tree crossover and mutation operators enforce the bound and will not produce trees that exceed it.

B.3.2.3. Body Segment Specifications

A creature's body can consist of segments of multiple types. The number of types is a parameter of the evolution and can be set by the user. All segments of the same type share the same properties such as the proportions of their dimensions, their branching structure, and so on.

Each segment specification in the genome contains the start angle and scaling value for its own sine wave generator (see B.3.2.1). It is this generator that is accessed when GP trees in the segment used the LSINE function.

Segment specifications also contain the following genes:

- Real-valued dimensions for the cuboid: x, y, and z

These dimensions are scaled up or down by a scaling value that percolates and multiplies down the body tree during development. They ultimately dictate segment proportions only, and not the final raw dimension values of any one segment.

- A real-valued scaling factor which is used to shrink or expand the dimension of the segment and all of its subtrees.

- Real-valued joint anchor offsets: x and y

These values determine the position of the joint anchor point on the $-z$ face of the segment. The joint anchor point is a shared point on the surface of both the segment and its parent, and about which they pivot in relation to one another. Offsets of $\{0,0\}$ would place the joint anchor point in the centre of the face, offsets $\{1,1\}$, $\{-1,-1\}$, $\{1,-1\}$, and $\{-1,1\}$ would place it at the four corners of the face. See Figure B.7, below, for an example (on the face labeled “F”).

- An integer called the colour index

This value is merely an index into a global array of available segment colours.

Despite being genetically encoded, segment colour has no effect on fitness.

- An integer called the max repeats value

This specifies the maximum number of times that instances of the type of segment can occur along any root-to-leaf path in the body tree.

- An integer called the type to follow repeats

This specifies that type of segment that will be substituted in the growth of the body tree when the number of repeats has been exhausted along a given root-to-leaf path.

Each segment specification also contains a fixed-length vector of branch specifications.

These specifications dictate the pattern of branches that sprout and grow from segments of the given type.

B.3.2.3.1. Branch Specifications

Each branch specification encodes a number of pieces of information on how segments should branch in the body tree. Each contains the following parameters:

- Three integer branch position offsets: x , y , and z

These can each have values -1 , $+1$, or 0 . Together they specify the location of a joint anchor point on the surface of the segment, where it will pivot relative to a child segment connected to it. The offsets $\{0,0,0\}$ would place the joint anchor point inside the segment, at its centre. This is not a legal location for the anchor point, and the various mutation and initialization operators for branch specifications ensure that it never occurs by changing one of the offset values whenever three zeros are encountered. All of the 26 other possible triplets specify locations on the surface of the segment, either at its corners, or at the centres of its edges and faces. The 26 possible branch position offsets are shown in Figure B.6, below.

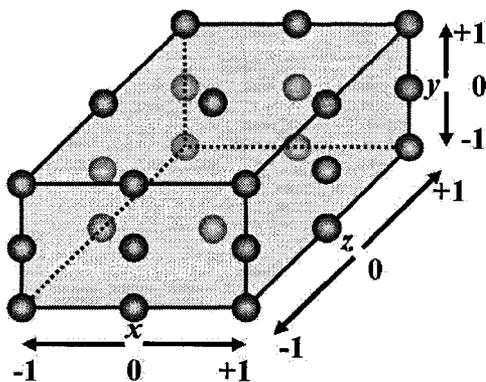


Figure B.6 An illustration of all 26 possible branch positions on a 3DVCE creature body segment.

- A real-valued y -axis angular offset

When a child segment is created, it is placed in space in such a way that the locations of the branch position on the surface of the parent segment, and the joint anchor

position on the $-z$ face of the child segment coincide. The child segment is oriented to point its own local z -axis directly away from the centre of the parent segment (computed as though the parent segment were a perfect cube, orienting the child at multiples of 45 degrees from the axes in three-dimensional space). The child segment is then rotated about the anchor point, around its own local y -axis, by a number of degrees specified by this parameter. This parameter has a small range of between -15 and $+15$ degrees. Large values here could orient the child segment in such a way that it extends into, and possibly through, the parent segment, which is aesthetically undesirable.

- An integer z -axis angular offset

This parameter operates in the same manner as the y -axis angular offset but rotating the child about its local z -axis instead of its local y -axis, giving it a twist. This value can range from 0 to 359 degrees, and is an integer due to an intuitive judgment that single-degree variations are fine enough for this purpose.

- An integer called the type

This is merely an index into the genome's segment specification vector. It determines the segment type of the child branch.

- A Boolean value called the mirror flag

This flag enables a form of symmetry in creature bodies. When set to true, a second copy of the branch will sprout from the parent segment, mirrored about the parent's yz plane. When all branches sprouting from the root segment of a creature are mirrored, the result is a creature that is bilaterally symmetric (except for possible symmetry disruptions during embryogenesis).

- A Boolean value called the flip flag

When set to true, the entire subtree of the branch will be flipped (become a mirrored version of itself) about its own local yz plane.

- A Boolean value called the enable flag

When set to true, the embryogenic process will attempt to generate the branch according to the settings provided by the parameters above. When set to false, the entire branch specification will simply be ignored.

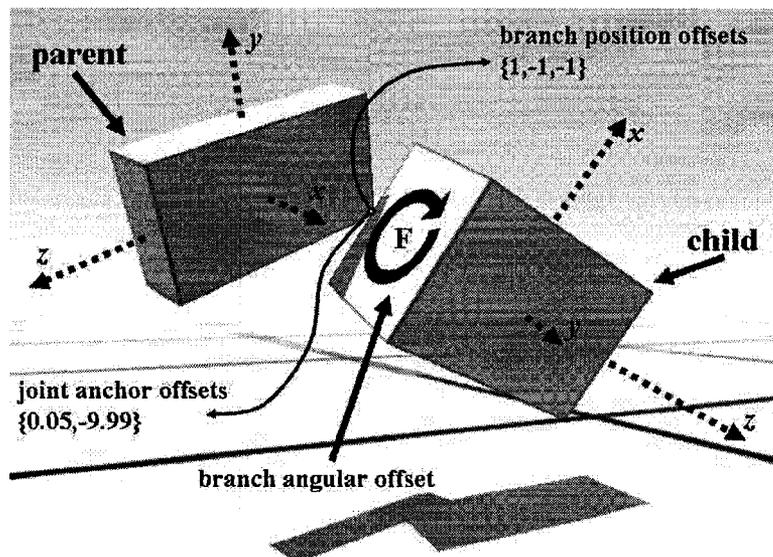


Figure B.7 An augmented screenshot of a 3DVCE creature illustrating several of the parameters related to branch positioning and orientation.

Figure B.7 is an augmented snapshot from 3DVCE showing the effects of some of the branch specification settings described above. The local x, y, and z axes of both the parent and child segments are drawn with dotted arrows. The branch position offset parameters of the parent segment's branch specification ($\{1,-1,-1\}$) determines the point of contact between the parent and child, and the anchor point of the joint, with respect to the parent. The branch specification's z-axis angular offset value determines a rotation

on one of them, as can be seen in the figure. Segment type *C* has a max repeats value of seven, as can be seen in the length of the chain in the figure. It also has type *D* as its type to follow repeats, and therefore a segment of type *D* appears at the end of the chain where normally a segment of type *C* would continue the recursion. Segment type *D* has a branch of type *D* but with a max repeats value of only two, and has type *E* as its type to follow repeats. Other branches for this creature's body tree may be specified in its genotype, but may either have their enabled flags set to false, or may have been pruned during embryogenesis.

B.4. Fitness Functions

The 3DVCE system provides the user with some degree of freedom in specifying their own fitness function for an evolution or for specific epochs in an evolution. This is done by allowing the user to specify the weighting factors that are used in a weighted combination of various measures of fitness (called "components"). Some components of fitness can also be set to be scaled, in which case their values are divided by the creature size measure described in B.3.1.3. There are six fitness components, labeled DIST, MAXH, AVGH, TOG, Sphere, and Transport. All of these components were used in the 3DVCE experiments described in Chapter 6 except AVGH and TOG.

B.4.1. Environments

The 3DVCE system allows the user specify one of four different simulated physical environments for a fitness function. These are named the flat terrain, rough terrain, spheres, and water environments. The following four sections describe each in turn.

B.4.1.1. Flat Terrain Environment

In the flat terrain environment, creatures are simply placed at the Cartesian origin, just above the infinite xz plane. The plane is solid and supports objects against gravity. This infinite solid ground plane is present in all four environments, but each of the other three environments adds something extra.

The flat terrain environment is used in the first epoch of 3DVCE experimental setup number two (see B.8.2).

B.4.1.2. Rough Terrain Environment

The rough terrain environment was originally created to get around a type of undesirable creature behavior that sometimes evolved in the flat terrain environment early in development before various simulation parameters had been tweaked, and before the problem of numerical explosions had been dealt with. Creatures would sometimes evolve that would simply shake rapidly enough to continually nudge themselves sideways, sliding across the flat terrain in miniscule jerky increments of distance. A form of rough terrain provided obstacles that could not simply be slid over as could the smooth flat ground plane. Even after simulation parameters had been tweaked and numerical explosions were under control, the rough terrain environment proved to be a more challenging problem than the flat terrain, but was nevertheless soluble, and was kept as one of the handful of adjustable fitness components.

When in the rough terrain environment, cuboid-shaped bumps are generated by placing static (fixed in place and unable to move) blocks on the ground only in a small circular

region surrounding the creature. As the creature travels, those blocks that fall outside the circle at the trailing edge are removed and new ones are generated to keep the circle filled at the leading edge. The blocks are aligned with one another in a grid pattern and have a height above the ground which increases quadratically with distance from the Cartesian origin in the xz plane. The block length and width are scaled to be proportional to the value of the same creature size formula used in fitness evaluation, described in B.3.1.3. The block dimensions are computed as follows.

Let x and z be the coordinates of the block's centre.

Let $size(C)$ be the creature size function from (B.3.1.3-1).

Let x' and z' be $x / size(C) * 1.5$ and $z / size(C) * 1.5$, respectively.

Let rnd be a uniformly-chosen random number between 0.0 and 1.0.

Let b be a constant bumpiness factor ($b = 0.3$).

Then block dimensions are calculated as:

$$\text{Height:} \quad = \left(\frac{x'^2 + z'^2}{100} \right) + 0.01 + (rnd)b(size(C)) \quad (\text{B.4.1.2-1})$$

$$\text{Width:} \quad = 1.5(size(C)) \quad (\text{B.4.1.2-2})$$

$$\text{Length:} \quad = 1.5(size(C)) \quad (\text{B.4.1.2-3})$$

The diameter of the generated terrain boundary circle is chosen to be large enough to make it very unlikely that a creature can reach beyond it, and small enough to keep the extra physics engine computations to a minimum.

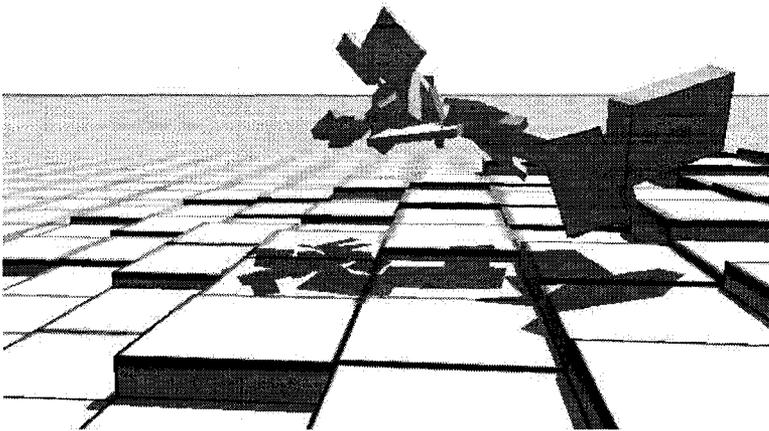


Figure B.9 A screenshot of a 3DVCE creature in the rough terrain environment.

The rough terrain environment is used in 3DVCE experimental setup one and in the second epoch of setup two (see B.8.1 and B.8.2).

B.4.1.3. Spheres Environment

In the spheres environment, a new sphere of radius 1.0 and density 0.5 is spawned every 100 simulation cycles (once per second of simulated time) starting at the 50th simulation cycle and continuing until creature evaluation time runs out. The spheres appear at altitude $y = 100.0$, with x and z coordinates chosen uniformly at random between 0.0 and 1.0. Contact with the ground triggers immediate removal of a sphere from the environment.

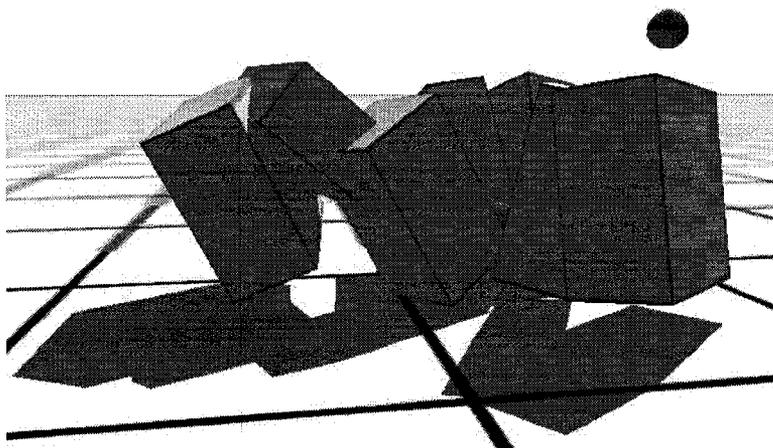


Figure B.10 A screenshot of a 3DVCE creature in the spheres environment.

In the spheres environment, creatures are selected for their performance in some kind of interaction with the spheres before they reach the ground. Using the Sphere component of the fitness function encourages creature populations to evolve the ability to keep as many spheres as possible from touching the ground for as long as possible (see B.4.2.5). Using the Transport component of the fitness function encourages populations to evolve the ability to catch at least one sphere and then carry it as from the Cartesian origin in the xz plane as possible (see B.4.2.6).

The spheres environment is used in 3DVCE experimental setup three and in both epochs of setup four (see B.8.3 and B.8.4).

B.4.1.4. Water Environment

In the water environment, linear and rotational drag forces are applied to all objects, including creature body segments to simulate the effect of motion through a liquid. Despite much tweaking and experimentation, this nearly always results in so-called

numerical explosions that destroy creatures. For this reason the water environment is not used in any 3DVCE experiments described in this work.

B.4.2. Components of Fitness

This section describes the creature performance measurements that constitute the six components that are combined in a user-configurable weighted combination for the final fitness function.

B.4.2.1. DIST Fitness Component

DIST is short for distance, and this component is simply the distance traveled by the creature during its allotted evaluation time. This component is used to encourage the evolution of creatures that can traverse, by whatever means, the greatest possible distance across whichever terrain is provided in their environment. At the time of this writing this is either the flat terrain or the rough terrain.

The DIST component of fitness is computed as the maximum value, over all evaluation cycles, of the Euclidean distance in the xz plane (i.e. horizontal distance only) between the creature's centre of mass and the initial position of its centre of mass.

The initial position of the creature's centre of mass is not determined at the moment of instantiation, immediately after embryogenesis, but is instead determined and recorded immediately after an initial settling period of 100 simulation cycles. The inclusion of this initial settling period is used to avoid rewarding undesirable creatures which are very tall and traverse a large horizontal distance simply by falling over.

If the scaling flag for the DIST component is set by the user, the measure will be divided by $1.5 * \text{size}(C)$ where $\text{size}(C)$, the size of creature C , is computed using the formula (B.3.1.3-1). This prevents large creatures from attaining high fitness values simply because of their size. Fleas can jump exceptionally high, but humans can jump higher.

B.4.2.2. MAXH Fitness Component

MAXH is short for maximum height, although this doesn't accurately reflect the nature of this component's measurement of creature behavior. This component is used to encourage the evolution of creatures that can jump as high as possible.

The creature's centre of mass is checked during every simulation cycle to update a variable tracking its minimum-so-far value, the smallest value of its y coordinate encountered since the creature was instantiated. During every simulation cycle, the difference between the y-coordinate of the centre of mass and its minimum-so-far value is calculated. If the scaling flag for MAXH is set to true, this difference is divided by $1.5 * \text{size}(C)$ where $\text{size}(C)$ is the creature size formula (B.3.1.3-1). The largest such difference encountered during evaluation, and occurring only after the initial settling period of 100 cycles, becomes the MAXH component of fitness.

This approach ensures that the maximum height above the ground used in the difference calculation comes after the minimum used, and thus reflects a net upward movement over time. Simply subtracting the overall minimum from the overall maximum would reward undesirable creatures that are just very tall and traverse a large vertical distance by merely falling over.

B.4.2.3. AVGH Fitness Component

AVGH is short for average height. This component measures the average height of the creature above the ground. During every simulation cycle a sum, S , is increased by the creature's current height above the ground, which is simply the y coordinate of its centre of mass. Also, during every cycle, a counter, N , is incremented to tally the number of height values that went into the sum, S . The final AVGH value is computed as S/N or, alternatively, $S/(N*1.5*size(C))$ if the AVGH scaling flag is set. $size(C)$ is computed using the creature size formula (B.3.1.3-1).

B.4.2.4. TOG Fitness Component

TOG is short for time on ground, and can be used to discourage jumping behaviors from evolving when used in weighted combination with other fitness components. The TOG value is simply the fraction of evaluation cycles in which the creature has at least one body part with a triggered contact sensor. It is computed as m/n where m is the number of cycles with a triggered contact sensor, and n is the total number of cycles. This measure, therefore, is a value between 0.0 and 1.0.

B.4.2.5. Sphere Fitness Component

This component of fitness is an integer that begins at zero when the creature instantiated and increments once every simulation cycle for every sphere that is still present. For example, if there are five spheres present during a given cycle then the integer is incremented by five during that cycle. The value is initialized to zero, not only when the

creature is first instantiated, but also immediately after the initial settling period of 100 cycles.

This component is used to encourage the evolution of creatures that can prevent as many falling spheres as possible from touching the ground for as long as possible.

B.4.2.6. Transport Fitness Component

The Transport component of fitness is used to encourage the evolution of creatures that catch a falling sphere and then carry it as far as possible from the Cartesian origin in the xz plane, without allowing it to touch the ground. For obvious reasons, this component should only be used for fitness functions that specify the spheres environment.

In every cycle, all spheres still present have their distances calculated from both the Cartesian origin and from the creature's centre of mass. These distances are calculated in the xz plane only, so that only the horizontal component of distance is measured. For each sphere, a value, ΔD , is calculated by subtracting the sphere's distance from the creature's centre of mass from its distance to the origin. Subtracting the distance to the creature's centre of mass when calculating ΔD discourages the rewarding undesirable creatures that simply deflect or throw a sphere a large distance instead of truly carrying it. The final Transport component's measure of fitness is set to the maximum ΔD over all spheres still present in the final simulation cycle. A consequence of this is that a sphere carried far and then dropped counts for nothing.

B.4.3. The Fitness Formula

The final fitness calculation involves a weighted combination of the six fitness components: DIST, MAXH, AVGH, TOG, Spheres, and Transport. Each weighting factor is a value between 0.0 and 1.0, and is specified by the user when configuring the fitness function for an evolution or for an epoch in an evolution.

The formula is

$$\begin{aligned}
 f(C) = & \\
 & W_{DIST} C_{DIST} + W_{MAXH} C_{MAXH} + W_{AVGH} C_{AVGH} + \\
 & W_{Spheres} C_{Spheres} + W_{Transport} C_{Transport} \\
 fitness(C) = & f(C) [1 - (1 - C_{TOG}) W_{TOG}]
 \end{aligned}
 \tag{B.4.3-1}$$

where C is the creature, W_i is the weight assigned to component i , and C_i is the value of fitness component i .

B.5. Evolution Configuration

Evolutions in 3DVCE require a number of settings choices from the user. This section covers the available settings, divided into sections for the GA itself, creature morphology, control systems, and fitness schedules.

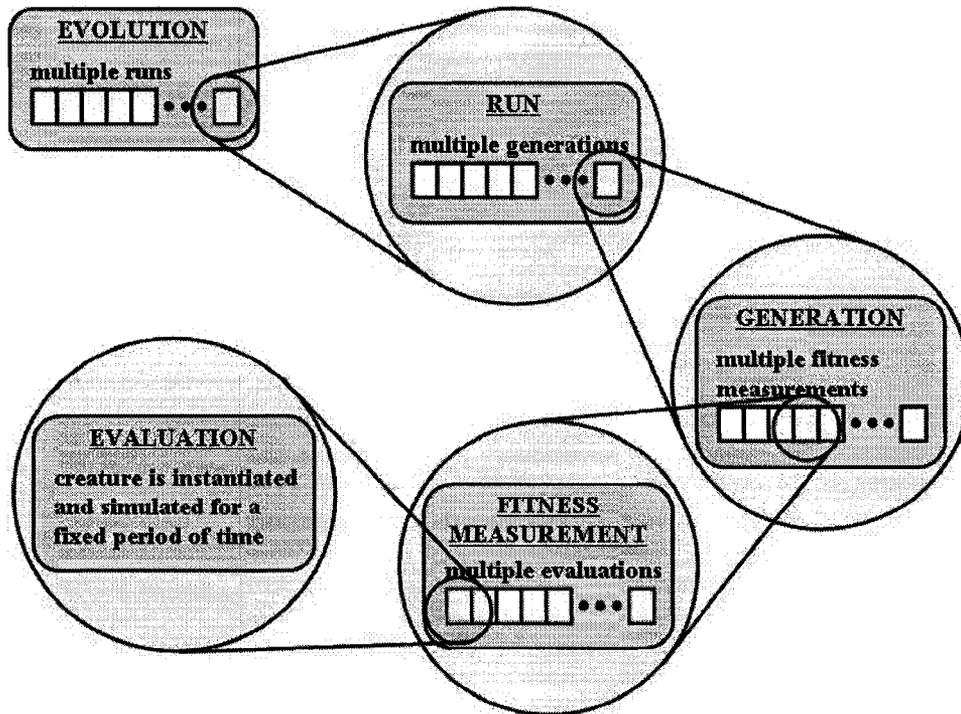


Figure B.11 A hierarchical breakdown of an evolution in 3DVCE.

Figure B.11 shows part of the overall structure of an evolution in 3DVCE. Any given evolution consists of a fixed or indefinite number of runs. Each run, in turn, involves a fixed or indefinite number of generations where offspring creatures are produced from parents chosen by selection from the previous generation. Each such offspring creature undergoes a fitness measurement, which itself involves one or more evaluations of the creature. Multiple evaluations may be specified in order to reduce noise caused by the physics engine's non-determinism. The final fitness measurement is the average over all evaluations for the creature.

B.5.1. GA Settings

Most of the GA-related settings are the usual ones associated with GAs, parameters like population size and mutation rate. Since there are several different mutation operators

and several different crossover operators (see B.6), there are operator rate settings unique to each. The GA-related settings in an evolution are as follows.

- **Population size**

This is one more than the number of new individuals created in each generation, since the first individual in a generation is not new, but a copy of the best individual from the previous generation.

- **Evaluation repeats**

This is the number of times that a creature will be evaluated during a fitness measurement (see Figure B.11). This value is set to one in all of the experimental setups described in this work (see B.8) although it can be set higher to reduce noise. The trade-off is one of noise reduction vs. computation time.

- **Tournament size**

Whenever a parent is needed, this setting dictates the number of individuals to choose at random from the previous generation. These individuals constitute a tournament. The creature with the highest fitness wins and becomes the selected parent. The higher this value is in relation to the population size, the higher the selection pressure will be and the more rapid the loss of diversity will be.

- **Crossover rate**

This is the probability that a new creature, in any generation other than the first, will be produced by the crossing of two parents from the previous generation.

- **Random tree crossover rate**

This is the probability, when crossing two GP trees from the control systems of two

parents, that one of the trees will be chosen at random from the entire control system of one of the parents instead of choosing the corresponding tree.

- **Special tree crossover rate**

Once a GP tree from each of two parents has been chosen for crossover, this is the probability that the special tree crossover operator will be chosen instead of the regular tree crossover operator (see B.6.1.2.3).

- **Control system mutation rate**

This is the mutation rate used in mutation operators affecting the control system parts of the genome.

- **Body mutation rate**

This is the mutation rate used in mutation operators affecting the morphology parts of the genome.

- **Run limit**

This the index of the final run to be executed in an evolution. The index begins at zero, not one. If this parameter is set to -1 then the number of runs will be indefinite, terminating only when the user intervenes.

- **Generation limit**

This is the index of the final generation to be executed in a run. The index begins at zero, not one. If this parameter is set to -1 then the number of generations will be indefinite, terminating only when the user intervenes.

B.5.2. Creature Morphology Settings

The following configurable evolution settings deal with aspects of creature morphology.

- **Number of distinct body segment types in genotype**

Each cuboid in a creature's body is of a particular type. Parameters for the type are encoded in the segment specification vector in the genome. This is a fixed-length vector, determining the number of cuboid types available for constructing the body. This parameters dictates the length of that vector.

- **Maximum body recursion depth**

This parameter sets the maximum number of cuboids along any root-to-leaf path in the creature's body.

- **Maximum number of body segments**

This parameter sets the maximum total number of cuboids allowed in a creature's body. The embryogenic process (see B.3.1.2) constructs the body in such a way as to ensure compliance with this limit and with the maximum body recursion depth, described above.

- **Body segment interpenetration flag**

Normally, during simulation, a cuboid in the creature's body is permitted to pass smoothly through any other cuboid to which it is directly connected by a joint. When encountering any other cuboid, a collision is simulated and the two are prevented from interpenetrating, according to the rules of rigid-body physics. When this parameter is set to true, all cuboids in a creature's body will pass smoothly through all others, as if there were not there. In all of the experimental setups described in this work (see B.8) this flag is set to false.

B.5.3. Creature Control System Settings

The following evolution parameters affect creature control systems. The complete list of GP node types is given in Table B.1.

- **Available creature-level GP node types**

This is the set of GP node types available for construction of GP trees at the creature level.

- **Available segment-level GP node types**

This is the set of GP node types available for construction of GP trees at the segment level.

B.5.4. Fitness Function Schedules

The fitness function used to measure the fitness of creatures in the population can change from one generation to the next in an evolution according to a fitness schedule arranged ahead of time by the user. The set of all consecutive generations using the same fitness function is called an epoch. Epochs and the fitness schedule allow populations evolved for one task to change direction, so to speak, and continue evolving for a new task.

A fitness schedule is a vector of fitness schedule entries. Each entry corresponds to a change of fitness function at a given generation. Once the fitness function changes, it will not change until the next generation specified by another entry is reached. Thus, epochs and schedule entries have a one-to-one correspondence.

The following settings determine a fitness schedule entry.

- **Starting generation**

This is the generation at which the new fitness function takes effect. This value must be unique for every entry in the schedule. Every schedule must contain exactly one entry with the starting generation set to zero.

- **Evaluation time**

This is the total number of simulation cycles expended for each evaluation in a fitness measurement. Each cycle is 0.01 seconds of simulated time. Some fitness functions may require more evaluation time than others. For instance, a function designed to evolve maximum jumping height may require only enough simulation time to allow the creature to jump up and fall back down again. A travel-distance fitness function may require more time to ensure that the creature's motion is a sustained one, and that horizontal travel distance is not just a byproduct of some transient initial phase in the control system.

- **DIST weight, MAXH weight, AVGH weight, TOG weight, Sphere weight, and Transport weight**

These are all values between zero and 1, used in the weighted combination of their namesake components in the fitness formula of equation (B.4.3-1).

- **DIST scaling flag, MAXH scaling flag, and AVGH scaling flag**

These are Boolean values used to specify whether their namesake components in the fitness function formula are to be divided by the creature size measure given in B.3.1.3.

B.6. Variation Operators

There are a number of different variation operators that create new versions of creature genomes, or parts of creature genomes, from older ones. These are all random variation operators, in the sense of being random with respect to their effects of fitness. They are all either mutation operators, which create random variations of a genome or part of a genome, or crossover operators which combine two genomes or parts of two genomes.

B.6.1. Crossover

Creature genomes are structured in part like those of a regular GA, with linear vectors of values. Specifically, the segment specifications and their branch specifications are such vectors. They are also structured in part like those of a GP system. Specifically the two GP trees at the creature level and the four trees in each segment specification. For this reason, there are quite different crossover operators for each of the various types of encoded genetic information.

B.6.1.1. Creature Crossover

Figure B.12 shows a flow-chart depicting the activity of the crossover operator that crosses two entire creature genomes. It first copies the settings for the creature-level sine wave generator from one of the two parents chosen at random with equal probability. It then iterates through the segment specification vector and calls a segment specification crossover method on each entry. Its final step is to use the GP crossover method (see 6.2) to cross the two creature-level GP trees.

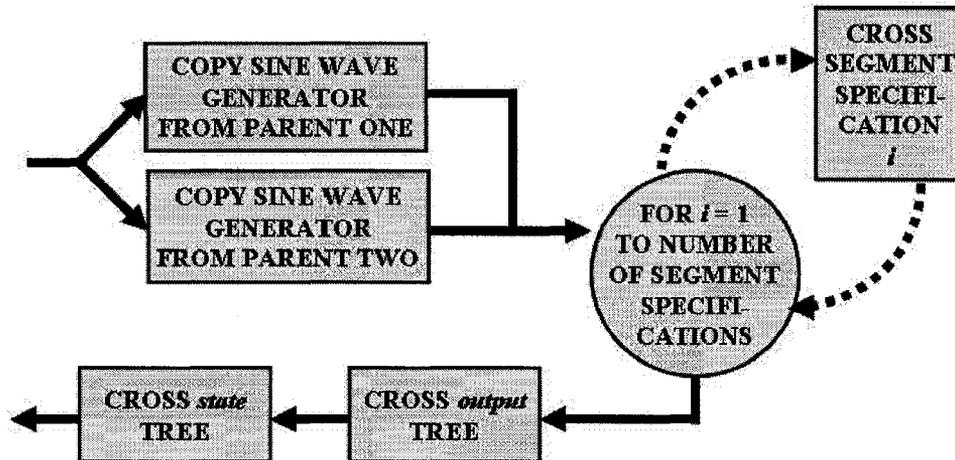


Figure B.12 A flow-chart for the 3DVCE creature crossover operator.

The crossover of pairs of individual segment specifications follows much the same pattern. All real-, integer-, and Boolean-valued parameters are inherited entirely from the segment specification of one parent, chosen at random and independently for each segment specification. The four GP trees are then crossed using the GP tree crossover operator. These steps are depicted in Figure B.13.

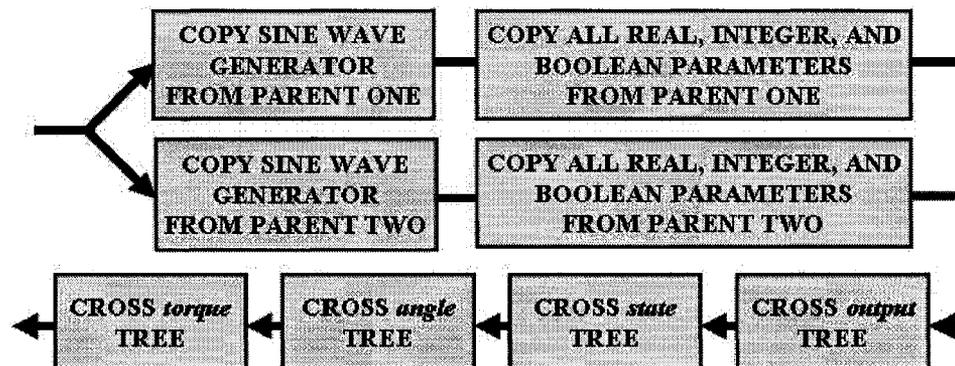


Figure B.13 A flow-chart for the 3DVCE segment specification crossover operator.

B.6.1.2. GP Tree Crossover

The crossover operator for GP trees in 3DVCE is based on the usual GP tree crossover operator [Koza, 1992] but with some modifications. No claim is being made here that

these modifications are either superior or inferior to the standard GP crossover operator for the problem of creature evolution in 3DVCE. Experiments could be devised to determine whether the changes are an improvement for this problem, but such experiments are too prohibitive in terms of computing time, and are not worth the effort for the purposes of this work.

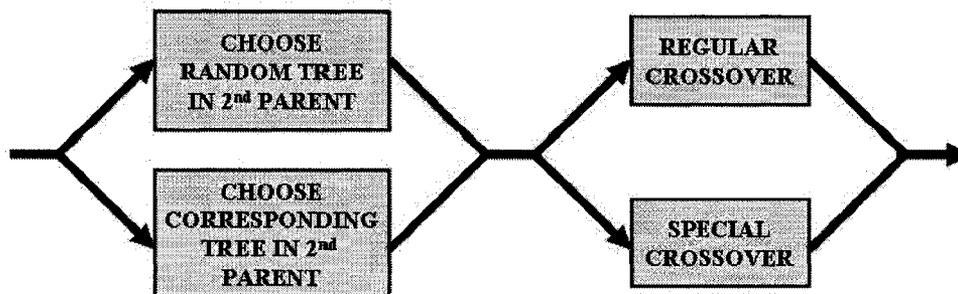


Figure B.14 A flow-chart for the 3DVCE GP tree crossover operator.

Figure B.14 depicts, at a high level, the two-step process of GP tree crossover in 3DVCE. The first step involves determining whether the tree from the first parent should be crossed with the corresponding tree in the second parent, or with a tree chosen randomly from all compatible trees in the second parent. Two trees are considered compatible if they are both creature-level trees, or both segment-level trees. This decision between the corresponding tree and a randomly-chosen tree uses the random tree crossover rate, and is meant to introduce variation by mixing material from trees that serve different functions.

The second step, once the two trees are chosen, is to apply either the regular tree crossover operator, or the special tree crossover operator. The regular tree crossover operator is very similar to standard GP tree crossover, but the special tree crossover is quite different. Both are explained in the sections, below.

B.6.1.2.1. Random Tree Crossover

As explained above, random tree crossover is merely the usual crossover procedure (either regular or special) but applied to trees that may not corresponding to one another in the two parent genomes. The probability of this type of crossover occurring is kept very low since the two trees involved are evolved to perform quite different functions, even though they are indirectly related in that they both form parts of the same control system.

B.6.1.2.2. Regular Tree Crossover

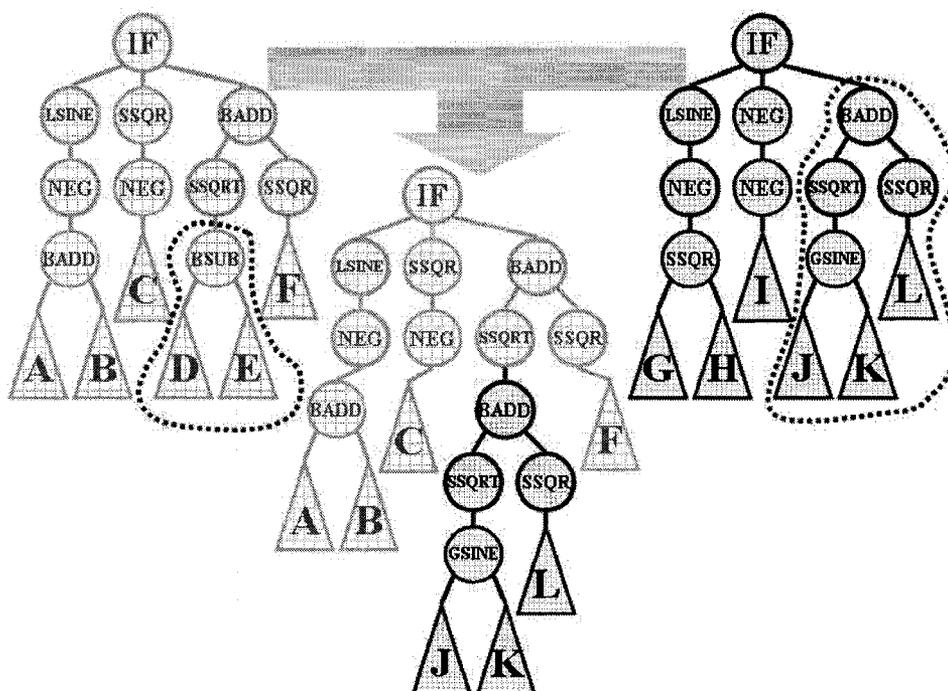


Figure B.15 An illustration of regular tree crossover in 3DVCE.

As shown in Figure B.15, regular tree crossover replaces a subtree in one parent with a subtree from the other parent. In the standard GP crossover operator, the choice of subtrees is completely unrestricted and uniformly random in both parents. In this case

however, the operator ensures that the resulting tree remains within the node count limit (see B.2.2) which is a fixed constant setting of the 3DVCE system. It does this by first choosing a subtree from the first parent. This is done uniformly at random, with every subtree having an equal chance of being selected. It then calculates the largest number of nodes that a replacement subtree could contain without violating the node count limit in the resulting offspring tree. This is done in a while loop that repeatedly chooses a subtree uniformly at random from the second parent until one is found that is small enough. The loop must eventually terminate since, in the worst case, it is always permissible for the replacement subtree to be a leaf node, and all GP trees contain at least one leaf.

B.6.1.2.3. Special Tree Crossover

The special tree crossover operator is significantly different than regular tree crossover operator, and is depicted in Figure B.16.

This operator attempts to conserve much of what is common between the two trees in the nodes nearest to the root, performing subtree replacements only at points where the two trees differ. This is accomplished by traversing both trees recursively and depth-first, in lock-step. Whenever the two nodes being visited exactly match one another, the node is copied to the offspring tree, and the method is then called, recursively, on each of that node's children, if any. When the two nodes do not exactly match, the offspring inherits the entire subtree from one of the parents, with each parent having an equal chance of contributing.

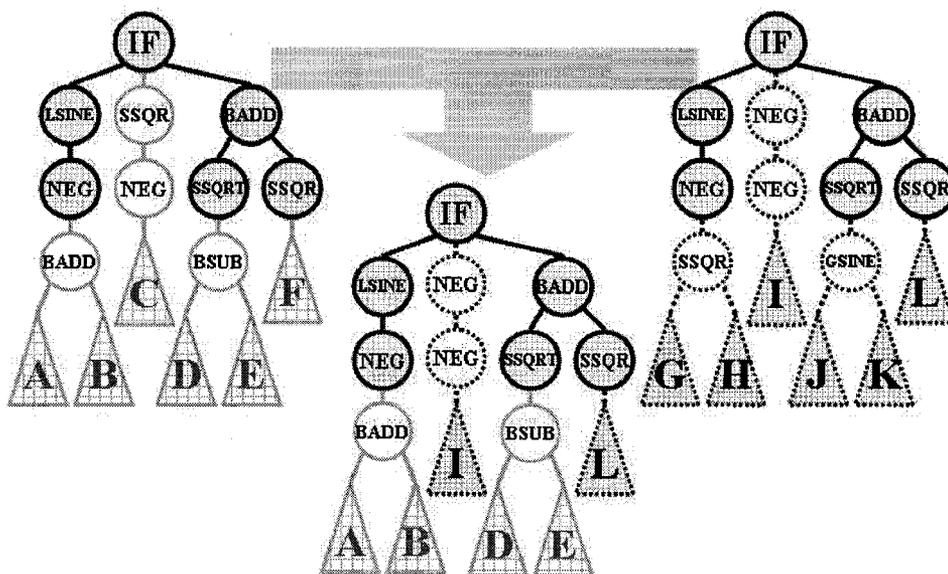


Figure B.16 An illustration of special tree crossover in 3DVCE.

B.6.2. Mutation

As with the crossover operator, by necessity the mutation operator consists of a number of different types of mutation that affect different types of objects and data stored in the genome. The mutation rate parameter for the GA has been divided into two separate rates, one for creature morphology, and one for the control system.

B.6.2.1. Creature Mutation

As depicted in Figure B.17, creature mutation begins by mutating the creature-level sine wave generator. There are two parameters involved. These are the start angle and scaling value, both of which are real values. Each is considered for mutation independently using the control system mutation probability. When a parameter is chosen for mutation, it is modified using the jiggle-real operator. This mutation operator generates three values within the legal range for the gene, each chosen uniformly at random, and replaces the original with whichever of the three is closest to it.

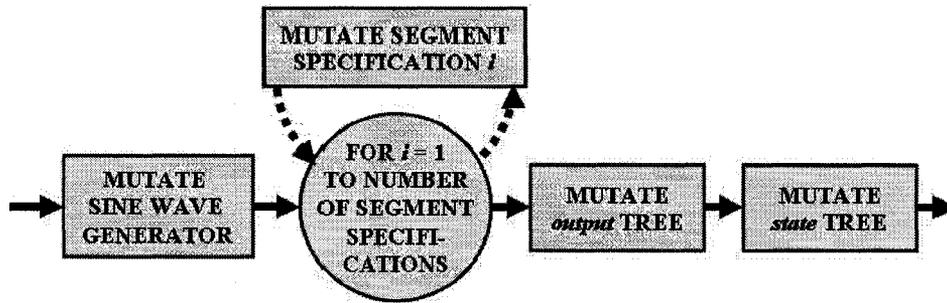


Figure B.17 A flow-chart for the 3DVCE creature mutation operator.

Once creature-level sine wave parameters have been mutated, the creature mutation operator then moves to the vector of segment specifications in the genome. The mutation of a segment specification is depicted in Figure B.18.

Each segment specification in the genome is first considered for a segment copying mutation (see B.6.2.2), using the body mutation probability.

Following the check for application of the segment copying operator, each segment specification in the vector has its real, integer, and Boolean parameters mutated using the body mutation rate.

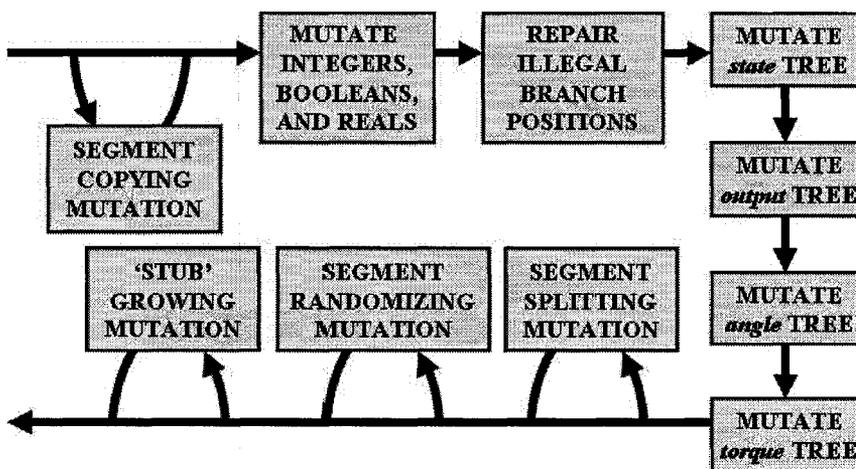


Figure B.18 A flow-chart for the 3DVCE creature segment mutation operator.

Integer-valued parameters may have a jiggle-int operator applied to them. The jiggle-int operator merely increments or decrements an integer value, except when it's at the upper or lower limits of the legal range, in which case the value is decremented or incremented, respectively. Once this is done, any illegal branch position offset coordinates are corrected. Branch position offset coordinates, x , y , and z , are considered illegal if they are all set to zero, which would place the branch's joint anchor point inside the parent segment (see B.3.2.3.1). Illegal branch position offset coordinates are corrected by choosing one of the three offsets (x , y , or z) uniformly at random and setting it to either -1 or $+1$, each value having a 50% probability of being chosen.

Following mutation of the integer values, real-valued parameters are checked for mutation, also using the body mutation rate. Real values that are chosen for mutation have the jiggle-real operator (described above) applied to them.

After the real-valued parameters come the Boolean-valued parameters. These are also each considered independently for mutation using the body mutation rate. A Boolean value chosen for mutation is simply toggled.

Next are the four GP trees for the segment specification. Each is mutated using the GP tree mutation operator (see B.6.2.6).

The final phase in the segment specification mutation involves three special mutation operations, each of which is done independently with a probability equal to the body mutation rate. These are the segment splitting mutation, the segment randomizing

mutation, and the ‘stub’ growing mutation. These are described in B.6.2.3, B.6.2.4, and B.6.2.5, respectively.

B.6.2.2. Segment Copying Mutation

The segment copying mutation replaces the entire segment specification entry in the vector with a copy of another chosen uniformly at random from the vector, but excluding the original. The new segment specification is not an exact copy of the original, but contains adjustments meant to preserve self-loops in the branching pattern of the body tree. This mutation is represented by the following pseudocode:

```
// Inputs:  SegmentSpec SS, int d, int s
//          SS is the segment specification to be copied
//          d is the destination index in the specification vector
//          s is the source index (of SS) in the specification vector
// The destination specification is the 'self' object whose
// become_special_copy() member is called
// -----
become_special_copy(SS, d, s)
  copy all real-valued parameters from SS to self
  copy all integer-valued parameters from SS to self
  copy all Boolean-valued parameters from SS to self
  copy desired joint angle GP tree from SS to self
  copy max servo torque GP tree from SS to self
  copy output GP tree from SS to self
  copy state GP tree from SS to self

  // Source segment self-referencing branches should become
  // destination segment self-referencing branches
  for b = 1 to max_branches do
    if segment type index of branch b of SS = s then
      set segment type index of branch b to d in self
```

A self-loop arises when branch specifications dictate that segments of a particular type should have child segments of the same type. The segment copying mutation preserves this direct recursion in the tree structure when copying a segment specification from one index in the segment specification vector to another.

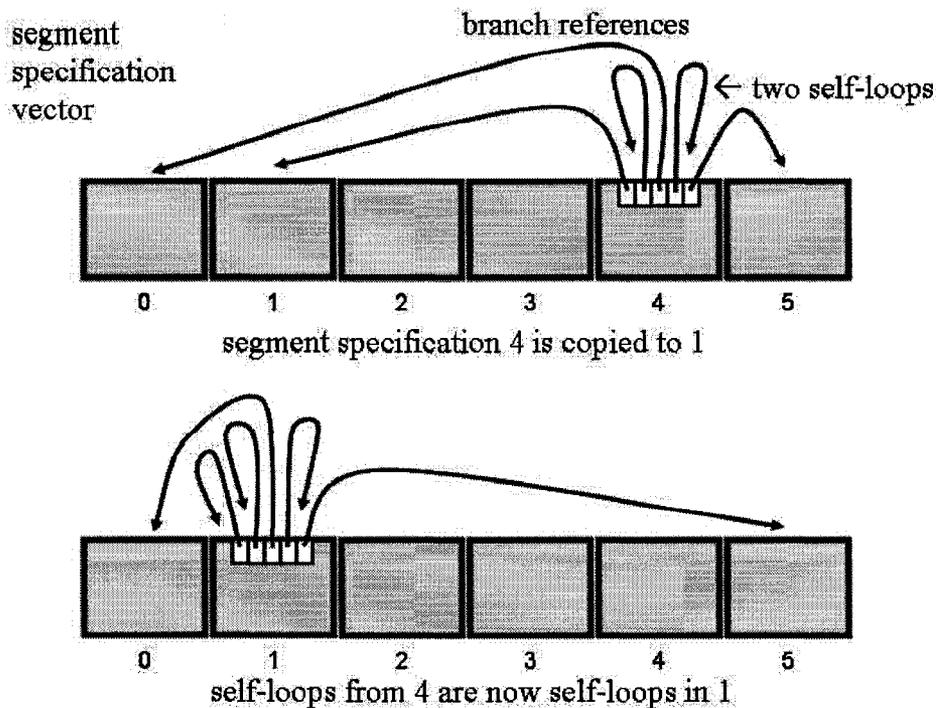


Figure B.19 An illustration of preserved self-loops when copying segment specifications in 3DVCE.

Figure B.19 illustrates an example of preserved self-loops in the segment copying mutation. The segment specification at index 4 is copied to index 1. Its branches of types 0, 1, and 5 are carried over unaltered, but its branches of type 4 become type 0.

B.6.2.3. Segment Splitting Mutation

The segment splitting mutation method is meant to be a conservative means of increasing body complexity. It attempts to replace all segments of a chosen type in the body with two segments connected by a joint but together occupying roughly the same space in the body as the segments they replace. The mutation proceeds according to the following pseudocode.

```
// Input:   int stos
//          stos is the index of the segment to be split
// Let:     ss[] be the vector of segment specifications
```

```

//          min_dim be the minimum allowable cuboid dimension length
// -----
segment-split(stos)
  // ovw is the index to a segment specification that will
  // be overwritten in the segment-split operation
  ovw = a random segment specification index other than stos

  // Halve the z dimension
  int halfz = max(ss[stos].z,min_dim)
  ss[stos].z = halfz
  ss[ovw].z = halfz

  copy GP trees from ss[stos] to ss[ovw]
  set servo angle GP tree of ss[ovw] to a tree that outputs zero
  clear enabled flag in all branch specifications of ss[ovw]
  copy specifications of all enabled branches in ss[stos] to ss[ovw]
  clear enabled flag in all branch specifications of ss[stos]
  set type of first branch specification in ss[stos] to ovw
  set enabled flag in first branch specification of ss[stos]

  // Branch position offset 0,0,1 is the middle
  // of face opposite the joint anchor point
  set the x,y,z position offsets of the branch to 0,0,1

  // Angular offset zero keeps the branch parallel to the parent
  set the z-axis and y-axis angular offsets of the branch to zero
  set the type to follow repeats to ovw
  clear the branch mirror flag

  // Joint anchor offsets of zero place the
  // joint anchor in the centre of the face
  set joint anchor offsets of ss[ovw] to zero
  copy all other parameters from ss[stos] to ss[ovw]
  copy sine wave generator parameters from ss[stos] to ss[ovw]
  set scaling factor of ss[stos] to 1.0

```

The effect of the segment splitting mutation method is to keep the creature morphology more or less the same, though this is by no means always the outcome, while splitting all segments of the chosen type into segments of half the original length. The split occurs half of the way along the segments' local z-axes, between the face upon which they have a joint anchor point joining them to their parent segments, and the opposite face.

Segment splitting is illustrated using a two-dimensional example in Figure B.20. In the figure, segment type *A* has been chosen for splitting, and segment type *C* has been chosen to be overwritten to form the second half of segments of type *A*. The effect of this mutation would be much more disruptive if segments of type *C* were already in use in the

body. In this case, however, type *C* is a spare segment specification in the genome and can be safely overwritten. The result is a tree with the same overall structure, but with the original segments of type *A* split into two halves of types *A* and *C*, with type *C* possessing all of type *A*'s old branches. The angle GP tree in segments of type *A* is set to a tree that outputs a constant of zero. This helps to keep the joint rigid and straight, thus minimizing the impact of the mutation on the morphology and behavior of the creature.

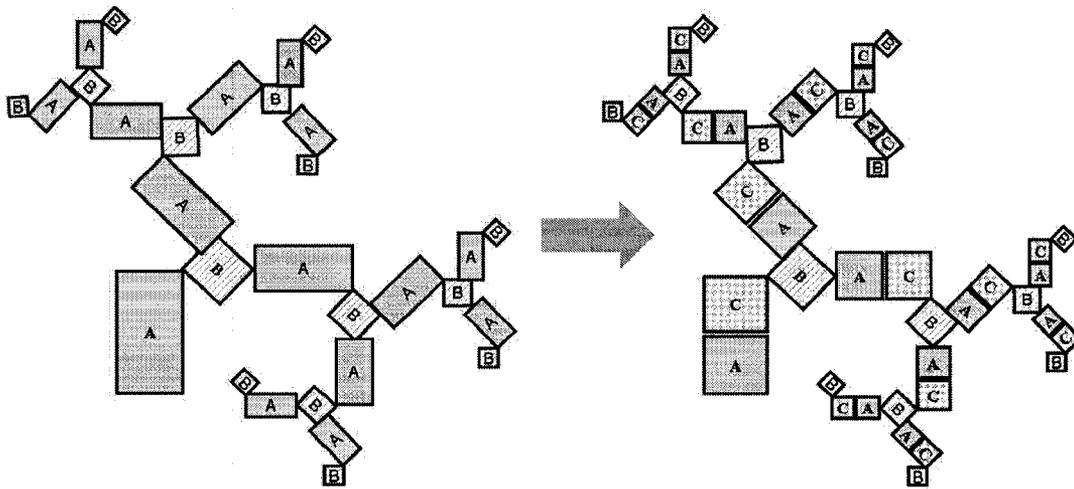


Figure B.20 A two-dimensional illustration of the effect of segment splitting in 3DVCE.

B.6.2.4. Segment Randomizing Mutation

The segment randomization mutation simply deletes a segment specification and replaces it with a randomly-generated one. The new segment specification is created by calling a segment specification constructor and passing it a bushiness parameter between 0.1 and 0.9, chosen uniformly at random. The bushiness parameter is used directly as the probability that any given branch specification in the segment specification will have its enable flag set to true instead of false. A high bushiness parameter is more likely to produce a segment with many branches.

The random segment specification constructor is shown by the following pseudocode:

```
// Input:      EvolutionSettings s, real b
//            s is an object containing important evolution parameters
//            b is the bushiness parameter
// Let:       biased_log(i,j) be a function which generates two
//            values, 10^m and 10^n, both between i and j with
//            exponents m and n chosen uniformly at random from the
//            range [log10(i)..log10(j)], and returning whichever
//            of the two values is closer to j.
//            rand(i,j) be a function that returns a value chosen
//            uniformly at random in the range [i..j]
// -----
SegmentSpecification(s,b)
    // Choose the dimensions of the segment,
// with a bias toward larger dimensions
    x = biased_log(min_cuboid_dimension,max_cuboid_dimension)
    y = biased_log(min_cuboid_dimension,max_cuboid_dimension)
    z = biased_log(min_cuboid_dimension,max_cuboid_dimension)

    // scale_variability is a fixed constant (0.5)
    scaling_factor = 1.0 + rand(-1.0,+1.0) * scale_variability

    joint_anchor_offset_x = rand(-1.0,+1.0)
    joint_anchor_offset_y = rand(-1.0,+1.0)
    colour = colour chosen uniformly at random from available colours
    max_repeats = a number chosen uniformly at random between zero and
    a fixed constant (12)
    type_to_follow_repeats = a segment specification index chosen
    uniformly at random between one and s.num_seg_specs
    for bs = 1 to the number of branch specifications (5) do
        // The following values are chosen uniformly at random
        branch_bs's_z_axis_angular_offset = an int between 0 and 359
        branch_bs's_y_axis_angular_offset = a real between +/-15.0
        branch_bs's_type = an int between 1 and s.num_seg_specs
        branch_bs's_position_offsets (x,y,z) = -1, 0, or +1
        illegal_branch_position_offsets (0,0,0) are repaired as per
        the method described above
        branch_bs's_mirror_and_flip_flags = true or false

        // The enable flag is set to true with a probability equal
        // to b, the bumpiness parameter passed to the constructor
        // and is set to false the rest of the time
        branch_bs's_enable_flag = true or false

    // The following two values are parameters of the segment
    // specification's sine wave generator
    start_value = a random value between 0.0 and 2*pi
    scale = a random value between +/-max_wave_scale (fixed at 10)
    angle, torque, output, and state GP trees are generated as
    described below
```

B.6.2.5. 'Stub' Growing Mutation

The 'stub' growing mutation method causes a segment to sprout a new branch which itself has no branches. Many mutations that result in new branches often produced large

complicated structures that, should they appear on a creature already fairly evolved, are almost always extremely detrimental. This mutation was created as a means of providing a more subtle mutation that adds complexity to a creature's morphology with what is intended to be a reduced impact on fitness. The method proceeds as follows.

```
// Input:      int i
//            i is the index of the segment specification to be mutated
// Let:        ss[] be the vector of segment specifications
// -----
stub-growth(i)
    // A segment specification other than i will also be modified in
    // this mutation. j is its index and is chosen uniformly at
    // random
    int j = a randomly chosen segment index other than i

    // Since ss[j] we be a 'stub' its max repeats will be set to one.
    // Since no segments should follow the repeats, the type to
    // to follow repeats will be set to j itself
    max repeats for ss[j] = 1
    type to follow repeats for ss[j] = j

    // Disable all branches extending from segments of type j
    for bs = 1 to the number of branch specifications (5) do
        clear the enable flag of ss[j]'s branch specification #bs

    // Pick a branch index in ss[i] to point to ss[j] (uniform random)
    int k = a number from 1 to the number of branch specifications (5)

    // Connect segment j as a branch from segment i and enable it
    set the type of ss[i]'s branch #k to j
    set the enable flag of ss[i]'s branch #k

    // The 'stub' should only be a single branch, not a mirrored pair
    clear the mirror flag of ss[i]'s branch #k
```

The check for the 'stub' growing mutation is the final step in mutating a creature's segment specifications.

B.6.2.6. GP Tree Mutation

Figure B.21 shows a flow-chart of the steps involved when mutating a GP tree in 3DVCE. In the creature-level and segment specification-level mutation operators, the tree mutation method is always called instead of using the control system mutation rate to

determine whether or not to call it. The reason is that a GP tree is a large structure consisting of many parts, and the mutation rate is used to determine whether those parts (nodes, leaves, subtrees, etc) are altered.

The GP tree mutation method first considers the tree for a shrink-tree mutation, which reduces the size of the tree. This is followed by consideration for a grow-tree mutation, followed by a point-mutate-tree mutation. The shrink-tree and grow-tree mutations are applied using the control system mutation rate, whereas the point-mutate-tree method is always called. The point-mutate-tree method, however, has the control system mutation rate passed to it as a parameter since it considers a potentially large number of tree components for mutation independently. The shrink-tree, grow-tree, and point-mutate-tree methods are explained in B.6.2.6.1, B.6.2.6.2, and B.6.2.6.3, below.

Once these mutations are complete, the tree's node count is tallied. If it exceeds the fixed limit (1000 nodes) then the shrink-tree mutation is applied repeatedly in a while loop until the tree's node count drops to within the legal range. Pseudocode for the GP tree mutation method is as follows.

```
// Input:    real csmr, GPNode root
//           csmr is the control system mutation rate
//           root is the root node in the GP tree to be mutated
// Let:      max_node_count be the fixed node count limit (1000)
// -----
mutate-tree(csmr, root)
    if root.node_count > 1 then do
        with probability csmr call shrink-tree(root)
        with probability csmr call grow-tree(root)
    point-mutate-tree(csmr, root)
    while (root.nodcount > max_node_count) do
        shrink-tree(root)
```

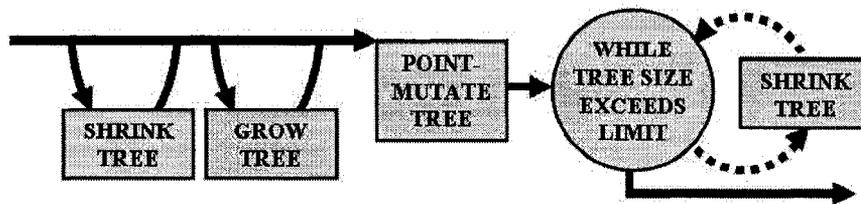


Figure B.21 A flow-chart for the 3DVCE tree mutation operator.

B.6.2.6.1. Shrink-Tree Method

The shrink-tree method first chooses a terminal node uniformly at random from the GP tree. It then replaces the terminal's parent node with the terminal itself. This reduces the overall tree size by at least one node and, in extreme cases, up to $n-1$ nodes, where n is the number nodes before the operation started.

B.6.2.6.2. Grow-Tree Method

The grow-tree method first chooses a terminal node uniformly at random from the GP tree. It then replaces the terminal with a non-terminal node chosen uniformly at random from the set of available non-terminal (also called "function") node types. The non-terminal node is then given enough terminal node branches, each chosen at random from the available types, to match its arity.

B.6.2.6.3. Point-Mutate-Tree Method

The point-mutate-tree method recursively traverses the tree depth-first. At each node encountered, it considers whether or not to make an alteration, using the control system mutation rate. If an encountered node is a terminal it may be replaced by another terminal chosen uniformly at random from the set of available terminal types. RND random

constant terminals may have their associated real values altered using the jiggle-real method described above (see B.6.2.1).

If a non-terminal node is encountered, it may have its type altered and replaced with another type chosen uniformly at random from the set of available non-terminal node types. If the arity of the new node type matches that of the old, the children remain in tact (apart from what further recursive calls to `point-mutate-tree` may do to them) and in their original order. If the arity of the node changes then an adjustment must be made. If the arity has been reduced then enough children will be removed to bring the number of children into agreement with the new node's arity. Each child removed is chosen uniformly at random from those that remain at each step. If the arity has increased then enough new children will be inserted into the vector of children to bring it into agreement with the new node's arity. Each new child is a copy of a subtree chosen uniformly at random from the entire GP tree. Such new children are not simply appended to the vector, but are inserted at a location in the vector that is chosen uniformly at random each time. Finally, the `point-mutate-tree` method simply continues down the tree via recursive calls to itself.

B.7. Population Initialization

The initial population of genomes for the GA is generated randomly, and with genomes that adhere to the various settings specified in the evolution configuration. That first generation of creatures is counted as generation zero.

B.7.1. Random Creature Generation

Each new random creature is produced according to the following algorithm.

```

// Input:      EvolutionSettings s
//            s is an object containing many important evolution settings
// Let:        ss[] be the vector of segment specifications
//            max_sine_wave_scale be a fixed constant (10)
// -----
Creature(s)
  // Construct random segment specifications
  // Use a random bushiness value
  real bushiness = random value between 0.1 and 0.9
  for i = 1 to s.number_of_segment_specifications do
    ss[i] = new SegmentSpecification(s,bushiness)

  // Set random sine wave generator parameters
  start_angle = a random real between 0.0 and 2*pi
  scale_value = a random real between +/-max_sine_wave_scale

  // Construct random (but small) GP trees
  output = new-random-tree(s)
  state = new-random-tree(s)

```

The same bushiness value is used in the construction of all segment specifications in the genome, although that value, chosen randomly, differs from one creature to the next. This produces an initial population with a wider variety of body plans than if no such parameter were used and each branch were enabled or disabled with equal probability. See B.6.2.4 for details on the generation of a random segment specification. That same random constructor is used in population initialization as is used in segment randomizing mutation.

B.7.2. Random GP Tree Generation

Randomly-generated GP trees, produced during population initialization are limited in size. The reason behind this approach is that it seems, based on some experimentation during development of 3DVCE, that smaller random trees produce behaviors that are

subjectively more interesting than if the full range of 1 to 1000 nodes is used. Of course, the tree size can increase as the generations pass.

The method first chooses a desired tree size (number of nodes), which it may not necessarily obtain. The size lies within a limited range of only 1 to 10 nodes. It then generates a random tree and tries a limited number of times to adjust its size toward the desired goal size if it is not already this size. Eventually, it either reaches the desired size or settles for whatever the final adjustment has left it with. The new-random-tree method proceeds as in the pseudocode below.

```
// Input:      EvolutionSettings s
//            s is an object containing many important evolution settings
// Let:        max_random_tree_node_count be the (intended) upper limit on
//            the number of nodes in a randomly-generated GP tree (10)
// -----
new-random-tree(s)
    // Choose a desired node count
    int c = a random number from 1 to max_random_tree_node_count
    begin with a randomly-chosen terminal as the root node

    // Decide on the number of attempts to reach the desired size
    // before giving up
    int tries = 2*max_random_tree_node_count

    // j will count the number of tries made
    int j = 0

    // Try to reach the desired node count
    while (node_count != c) and (j < tries) do
        if node_count < c then do grow-tree(root)
        if node_count > c then do shrink-tree(root)
        j++
```

B.8. Experimental Setups

The four experimental setups described below are used in two tests that compare the performance of a single-fitness-function evolution configuration with an otherwise identical configuration that uses an initial ‘preadaptation’ epoch under a different fitness function in an effort to prime the population with useful morphologies and control

systems that can be exapted to serve the true intended fitness function. In each case the total amount of simulation time in a run, the population size, and the various mutation, crossover, and other parameters are the same between pairs of experimental setups.

Setups one and two provide data for the first comparison. Setups three and four provide data for the second.

B.8.1. Setup One: Traversal of Rough Terrain without Exaptation

Experimental setup number one is a single-fitness-function configuration that attempts to evolve a population of creatures to traverse a rough terrain. Details of rough terrain generation are given above in B.4.1.2. This setup uses its final desired fitness function from start to finish, in runs with a single epoch.

The configuration parameters for setup one are shown below in Table B.2. At 30s per evaluation, 1 evaluation per fitness measurement, 100 creatures, and 500 generations, the total simulation time per run is $30s \cdot 1 \cdot 100 \cdot 500 = 1.5 \times 10^6 s$, or about 17.36 days of simulated evolution.

Parameter, Option, or Measure	Value or Name
Fitness function	Traverse rough terrain
Number of generations	500
Evaluation time per creature	30s
DIST component weight	1.0
MAXH component weight	0.0
AVGH component weight	0.0
TOG component weight	0.0
Sphere component weight	0.0
Transport component weight	0.0
DIST scaling	True
MAXH scaling	False
AVGH scaling	False
Environment	Rough terrain

Population size	100
Simulation time per run	1.5×10^6 s
Number of evaluation repeats	1
Tournament size	5
Crossover rate	0.5
Random tree crossover rate	0.05
Special tree crossover rate	0.5
Control system mutation rate	0.0001
Body mutation rate	0.001
Body segment interpenetration	False
Max body segment count	40
Max body depth	12
Number of segment types in genome	10
Available creature GP node types	All
Available segment GP node types	All

Table B.2 The settings used in 3DVCE experimental setup one.

B.8.2. Setup Two: Traversal of Rough Terrain with Exaptation

Setup two is identical to setup one except for having the simulation time in each run divided into two epochs. The second epoch uses the same fitness function as that used in setup number one – traversal of rough terrain. The first epoch attempts to preadapt the population by evolving creature that can jump. The intuition behind this choice is that creatures that can jump may have an easier time traveling horizontally across the rough terrain since they might be able to jump over the jagged bumps and dips. The two setups are an attempt to see whether, under these conditions, the lost processing time under the goal fitness function is balanced by the preadaptation of the population.

The parameters for the evolution in experimental setup number two are identical to those in setup number (see Table B.2), except for the fitness function parameters. In setup two there are two sets of fitness function parameters. The first, shown in Table B.3, apply to the initial preadaptation epoch of each run. This epoch expends 10s of simulation time per evaluation, has 1 evaluation per fitness measurement, has a population size of 100

creatures, and lasts for only the first 150 generations. This gives is a total of 1.5×10^5 s of simulated time per run, or about 1.74 days. The second epoch (see Table B.4), at 30s evaluation time, 1 evaluation per measurement, 100 creatures, and 450 generations consumes 1.35×10^6 s, or about 15.62 days of simulated time. The combined total for the two epochs is equal to the simulation time from experimental setup number one, putting them on equal footing for comparison.

Parameter, Option, or Measure	Value or Name
Fitness function	Jump
Number of generations	150
Evaluation time per creature	10s
DIST component weight	0.0
MAXH component weight	1.0
AVGH component weight	0.0
TOG component weight	0.0
Sphere component weight	0.0
Transport component weight	0.0
DIST scaling	False
MAXH scaling	True
AVGH scaling	False
Environment	Flat terrain
Simulation time for the epoch	1.5×10^5 s

Table B.3 The settings used in 3DVCE experimental setup two, epoch one.

Parameter, Option, or Measure	Value or Name
Fitness function	Traverse rough terrain
Number of generations	450
Evaluation time per creature	30s
DIST component weight	1.0
MAXH component weight	0.0
AVGH component weight	0.0
TOG component weight	0.0
Sphere component weight	0.0
Transport component weight	0.0
DIST scaling	True
MAXH scaling	False
AVGH scaling	False
Environment	Rough terrain
Simulation time for the epoch	1.35×10^6 s

Table B.4 The settings used in 3DVCE experimental setup two, epoch two.

B.8.3. Setup Three: Sphere Transporting without Exaptation

Like setup one, number three is a single-fitness-function configuration with only one epoch. It uses a fitness function designed to encourage the evolution of creatures that can catch at least one falling sphere in the spheres environment (see B.4.1.3 and B.4.2.6), and carry it as far as possible in the allotted evaluation time.

Table B.5 shows the settings for these runs. At 20s of simulated time per evaluation, 1 evaluation per measurement, 80 creatures, and 400 generations, these runs consume 6.4×10^5 s of simulated time, or about 7.41 days.

Parameter, Option, or Measure	Value or Name
Fitness function	Transport sphere
Number of generations	400
Evaluation time per creature	20s
DIST component weight	0.0
MAXH component weight	0.0
AVGH component weight	0.0
TOG component weight	0.0
Sphere component weight	0.0
Transport component weight	1.0
DIST scaling	False
MAXH scaling	False
AVGH scaling	False
Environment	Spheres
Population size	80
Simulation time per run	6.4×10^5 s
Number of evaluation repeats	1
Tournament size	4
Crossover rate	0.5
Random tree crossover rate	0.05
Special tree crossover rate	0.5
Control system mutation rate	0.0005
Body mutation rate	0.005
Body segment interpenetration	False
Max body segment count	40
Max body depth	12
Number of segment types in genome	12

Available creature GP node types	All
Available segment GP node types	All

Table B.5 The settings used in 3DVCE experimental setup three.

B.8.4. Setup Four: Sphere Transporting with Exaptation

Setup number four uses settings identical to those in Table B.5, except with respect to the fitness function. It also encourages the evolution of creatures that can catch and transport at least one falling sphere in the spheres environment (see B.4.1.3), as in experimental setup number three, but it splits the task into a brief preadaptation epoch followed by an epoch using the desired fitness function. In the preadaptation phase, creature are evolved for their ability to simply catch as many of the falling spheres as possible and prevent them from touching the ground for as long as possible in the allotted evaluation time. The intuition behind this approach is that it may be beneficial for creatures to first learn just to catch spheres, and to develop the sphere-container part of their morphology into a space large enough to hold many spheres. The hope is that such a container will help to prevent a single sphere from falling out once the fitness function changes.

The settings applicable to epoch number one are shown in Table B.6. At 20s of simulated time per evaluation, 1 evaluation per measurement, 80 creatures, and 80 generations, these epochs consume 1.28×10^5 s of simulated time per run, or about 1.48 days. Epoch two's settings are shown in Table B.7. At 20s per evaluation, 1 evaluation per measurement, 80 creatures, and 320 generations, it consumes 5.12×10^5 s, or about 5.93 days per run.

Parameter, Option, or Measure	Value or Name
Fitness function	Catch spheres
Number of generations	80

Evaluation time per creature	20s
DIST component weight	0.0
MAXH component weight	0.0
AVGH component weight	0.0
TOG component weight	0.0
Sphere component weight	1.0
Transport component weight	0.0
DIST scaling	False
MAXH scaling	False
AVGH scaling	False
Environment	Spheres
Simulation time for the epoch	1.28×10^5 s

Table B.6 The settings used in 3DVCE experimental setup four, epoch one.

Parameter, Option, or Measure	Value or Name
Fitness function	Transport sphere
Number of generations	320
Evaluation time per creature	20s
DIST component weight	0.0
MAXH component weight	0.0
AVGH component weight	0.0
TOG component weight	0.0
Sphere component weight	0.0
Transport component weight	1.0
DIST scaling	False
MAXH scaling	False
AVGH scaling	False
Environment	Spheres
Simulation time for the epoch	5.12×10^5 s

Table B.7 The settings used in 3DVCE experimental setup four, epoch two.

Appendix C. Mazes

The following figures depict the seven mazes used in fitness evaluation for the GP system described in Chapter 10. White cells represent open space, black cells are maze walls, and the letters “S” and “E” mark the start and end cells, respectively.

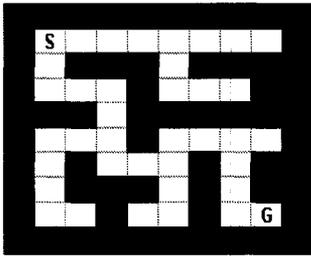


Figure C.1 Maze #1: 10 rows, 10 columns.

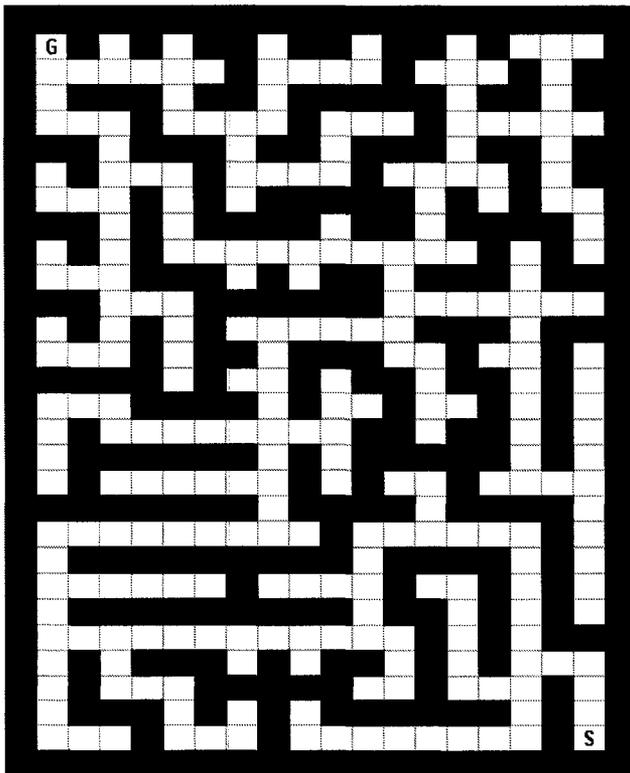


Figure C.2: Maze #2: 30 rows, 20 columns.

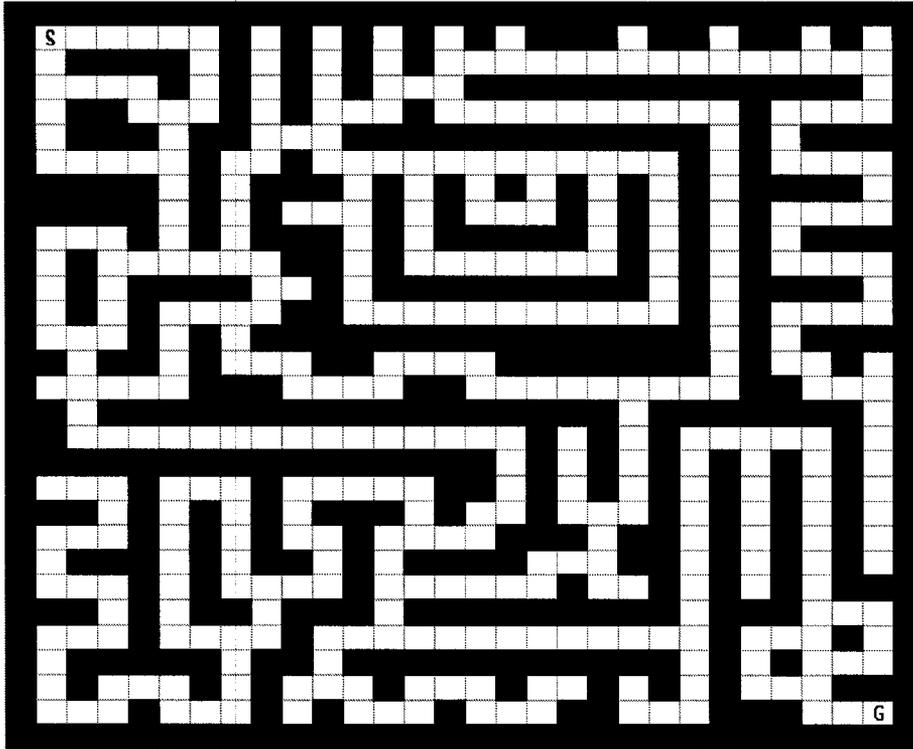


Figure C.3 Maze #3: 30 rows, 30 columns.

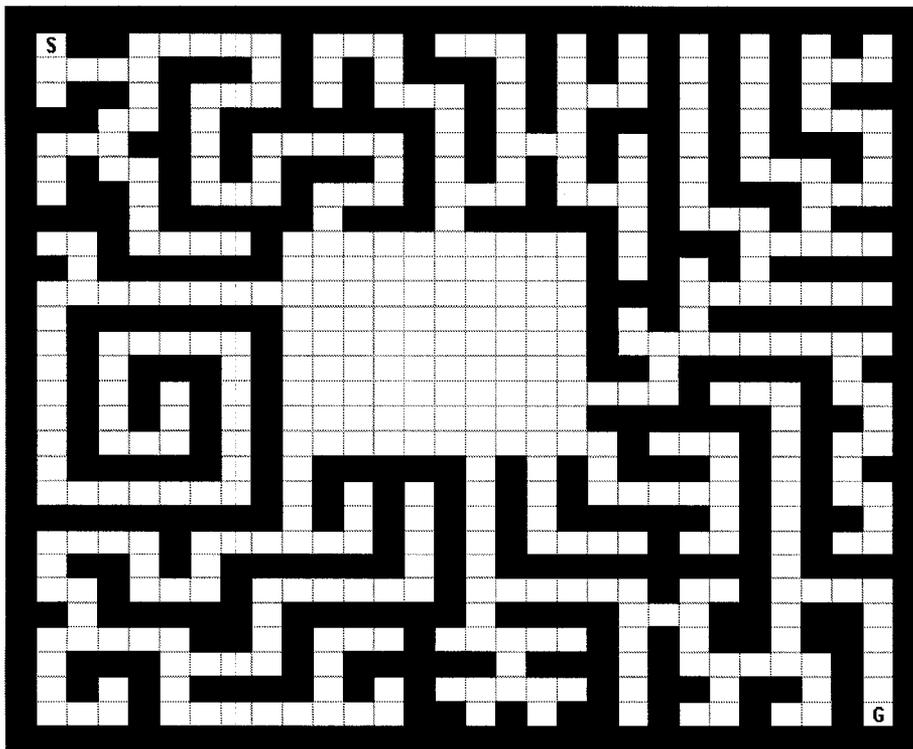


Figure C.4 Maze #4: 30 rows, 30 columns.

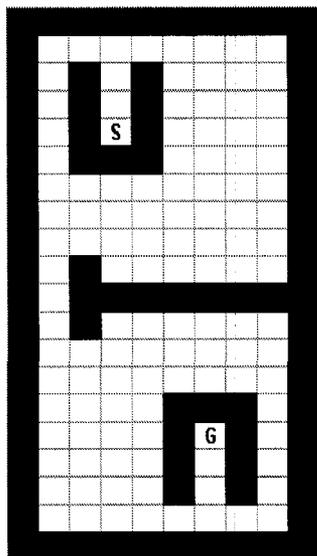


Figure C.5 Maze #5: 20 rows, 10 columns.

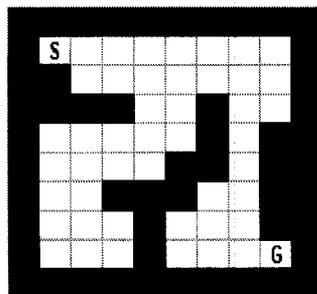


Figure C.6 Maze #6: 10 rows, 10 columns.

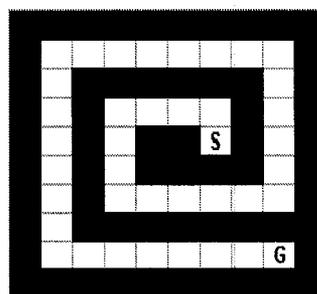


Figure C.7 Maze #7: 10 rows, 10 columns.

Appendix D. Exaptation Category Rationales

This appendix provides the present author's rationales for the exaptation category choices behind Table 11.2 from section 11.1. The table lists multiple exaptive EC systems and categorizes them according to the exaptation types described in section 5.3 of Chapter 5. Most of the types of exaptation and the distinctions between them are to some degree subjective, fuzzy, or are extremes of what, in reality, is a spectrum or gradient. For this reason, any categorization of an exaptive system using those terms is not absolute and is best accompanied by some explanation. As this would largely distract from the flow and purpose of section 11.1, it has been relegated to this appendix.

Spandrels in an ALife 2D Cellular Automaton

This system, called Enact, is described in [de Oliveira, 1994], and is listed in Table 11.2 as exhibiting both *strong* and *weak* types of exaptation. Since the Enact system is designed to contain no adaptations, none of the genetic evolutionary changes are building on top of previous ones to improve some earlier function. From this perspective, all exaptation moves from a population member with one behavior to a new population member with a new behavior. Since the second behavior cannot be seen as an adaptive enhancement of the first because there are no adaptations, it is a different behavior. For this reason the exaptations in Enact have been categorized as *strong*. For the same reason, the exaptations are categorized as *replacement* exaptations and not *cumulative*; the old function is lost when the new one is gained. On the other hand, the behaviors involved are simple horizontal and diagonal movement sequences that determine the paths taken by individuals in the environment. From this perspective, all behaviors are of the same

type and can only differ within the narrow constraints provided by Enact's representation scheme, and thus the exaptation may also be seen as *weak*. Since Enact is an ALife EC system, it lacks an explicit fitness function. This makes the details of any exaptation unplanned. For this reasons the exaptations are categorized as *emergent* as opposed to *deliberate*. Since no evolved entities are uniting to become components of larger higher-level entities, Enacts exaptation are clearly *non-compositional*.

Preadaptation of Neural Networks in a Changing Environment

This system is described in [Lund & Parisi, 1995]. In it, populations evolved to exploit a specific "food" resource in their environment via a specialist strategy were found to adapt more quickly to a changed environment requiring a specialist strategy for a different food than an otherwise identical population produced by random initialization. Table 11.2 categorizes this as a *weak* exaptation since both abilities are of essentially the same type. Since the changed environment was part of the experimental setup, the exaptations are *deliberate*. Since the specialization for the old food resource is lost and the new one gain, the changes are *replacement* exaptations. Finally, because no evolved entities are united to form components of a higher-level entity, the exaptations are *non-compositional*.

Incremental Approaches in the Evolution of Neural Network Robot Control

Systems

This system is described in [Kodjabachian & Meyer, 1998a & b]. It involves the evolution of a complex control system behavior in several stages. At the end of each stage the evolved system becomes a fixed and unchangeable module in a larger system which continues to evolve. Since the task in each new epoch is fundamentally different

from those in previous epochs (locomotion, gradient-following, obstacle avoidance, etc) the exaptations are categorized as *strong*. Since they are planned ahead of time in detail, they are *deliberate*. Since the functionalities of the fixed unchangeable modules are neither lost nor altered after epoch transitions, the exaptations are *cumulative*. Since the entire control system becomes a component in a larger evolving whole, the exaptations are *compositional*.

Symbiotic Composition for Crossing Fitness Saddles

This system is described in [Watson & Pollack, 2001]. In it, a composition mechanism that allows individuals to join together to create solutions to higher levels of the fitness function's problem is used to allow a GA population to cross saddles in the fitness landscape that are much more difficult to cross without the mechanism. Since the changes that result from the composition mechanism provide solutions to higher-level versions of the same fitness function, the exaptations are *weak* exaptations. Since the changes from one level to the next are intentional constructions planned in detail by Watson and Pollack, the exaptations are *deliberate*. Since individuals combined into one solve a new higher level of the fitness function and are no longer evaluated on the old lower level, the exaptations are *replacement* exaptations. Since the evolved individuals are combined to form larger individuals, the exaptations are *compositional*.

Exaptation in Dynamic Landscapes

This system is described in [Fentress, 2005]. It involves island populations adapting to variations on the desired target problem. Those island populations contain solutions which are exapted to help solve the main population's fitness function when it changes.

Since the fitness functions are each of the same type, and the changes are due to the landscape being dynamic, the exaptations are categorized as *weak*. Since the functional changes during the exaptations are orchestrated ahead of time by the programmer, they are *deliberate*. Since an individual serves only the function of the population in which it resides, the ability to solve the function of the island population from which it originated is not maintained. For this reason the exaptations are *replacement* exaptations. Since solutions are not combined to form higher-level solutions, the exaptations are *non-compositional*.

Gradual Functional Change for Evolving Differentiation in Gene Regulatory Networks

This system is described in [Knabe, Nehaniv, & Schilstra, 2006]. It involves the evolution of a switch-like behavior in an artificial gene regulatory network wherein the goal is to evolve a network which exhibits two different behaviors depending on the presence or absence of an external signal from the environment. They found the evolutionary system produced superior performance if the requirement for switch-like behavior was gradually introduced via a changing fitness function, compared to an otherwise identical system with switch-like behavior evaluated in full from the start. Since the fitness functions at the beginning and end of each run were very different – one involving a single behavior and one involving two behaviors controlled by an external signal – the exaptation is categorized as *strong*. Since the fitness function changes are planned ahead of time in detail, they are *deliberate*. Since the early behavior of the network is still part of the requirements of the final desired behavior, the exaptations are

cumulative. Finally, since no individuals are united into a new evolving whole, the exaptations are *non-compositional*.

The Simple Exaptive GA (single niche, two niche, four niche, and combination-fitness)

Each of these GAs is quite similar to the others. They are all described in Chapter 6. With the exception of the single niche exaptive GA, which involves changes in just a single parameter of the fitness function, each of these GAs involved individuals adapted to one niche or fitness function being exposed to an entirely different one. For this reason, the single niche exaptive GA's exaptations are categorized as *weak* while all others are categorized as *strong*. Since all four GAs involved fitness functions that were chosen in detail ahead of time, their exaptations are all *deliberate*. Since none of the exapted individuals' populations were required to maintain or increase their scores from fitness functions applicable before their exaptation events took place, they are all *replacement* exaptations¹⁸. None of the simple exaptive GAs implemented composition mechanisms and are therefore categorized as *non-compositional*.

2D Navigator Exaptive GP

This system is described in Chapter 7. It compares an exaptive and non-exaptive approach to evolving the controller for embodied agent navigating a 2D environment. The exaptive approach builds to the final learning task in stages. Since some epoch transitions involve substantial changes to the fitness function, some exaptations are

¹⁸ One minor exception to this is the *cumulative* exaptation configuration plotted in the 2D fitness space graphs presented in section 6.3.6 for the modified version of the two niche exaptive GA.

strong, and since some epoch transitions keep the same fitness function but merely change the environment, some exaptations are *weak*. Since all epoch transitions follow a fixed predetermined schedule, they are *deliberate*. Since behaviors learned in one epoch are still needed to successfully perform the task of the next, the exaptations are *cumulative*. Since navigator controllers are not combined with others to create larger higher-level controllers, the exaptations are *non-compositional*.

3D Virtual Creature Evolution

This system is described in Chapter 8. It involves the evolution and exaptation of both the body morphology and the control of embodied agents in the 3D environment of a rigid-body physics simulation. The exaptations are categorized as *strong* since both the transition from jumping to traversing rough terrain and the transition from catching many spheres to transporting a single sphere are quite different. This distinction is heavily subjective, but neither type of transition seems a merely subtle change. Since both types of transitions operate on a fixed schedule whereby the applicable fitness function changes abruptly at a predetermined generation in a predetermined way, the exaptations are *deliberate*. Since a rough terrain traversing agent is not necessarily required to jump as it travels, and since a sphere transporting agent is not required to catch and hold as many spheres as possible, the exaptations are all of the *replacement* type. Since no 3D agent is combined with another to form a larger agent, the exaptations are *non-compositional*.

‘Piecewise’ Exaptive Symbolic Regression GP

This system is described in section 9.1. It compares the performance of a normal GP configuration for a symbolic regression problem with a known target function to one in

which the target function starts off simple and, over multiple epochs, builds to the final target function. Since the target function is changed only slightly between epochs, the exaptations are *weak*. Since the changes are planned they are *deliberate*. Since no individual is ever reevaluated using the target function of an earlier epoch, the transitions are *replacement* exaptations. Since no composition mechanism is used, the exaptations are *non-compositional*.

‘Derivative’ Exaptive Symbolic Regression GP

This system is described in section 9.2. It compares the performance of a normal GP configuration for several symbolic regression problems to one in which the population is preadapted by attempting to fit an approximation of the target function’s derivative. Since the target functions are so closely related, and since both are symbolic regression problems, the exaptations are categorized as *weak*. Since both the function and its approximated derivative are decided at the outset to be the targets used in each epoch, the exaptations are *deliberate*. Since candidate solutions being compared to the target function are never reevaluated as candidate derivative solutions, the exaptations are *replacement* exaptations. They are also *non-compositional* since no two solutions are united as components of a new whole.

Maze Runner Exaptive GP

This system is described in Chapter 10. It compares an exaptive and a non-exaptive approach to the evolution of a maze running algorithm that guides an agent through a set of seven grid mazes. Since the preadapting fitness goal of exploring the entire maze, and the final goal of reaching its finish are conceptually very different goals, the exaptations

are categorized as *strong*. Since the change follows a fixed predetermined schedule, they are *deliberate*. Since thorough maze exploration is not necessary to satisfy the goal of the final learning task, the exaptations are *replacement* exaptations. Since no two maze running algorithms are combined to form a new higher-level maze running algorithm, the exaptations are *non-compositional*.

References

[Ackley & Littmann, 1992] Ackley D., and Littmann M., 1992. Interactions Between Learning and Evolution. *Artificial Life II*, Langton C.G., Taylor C., Farmer J.D., and Rasmussen S., eds., Addison-Wesley.

[Adami, 2006] Adami C., 2006. Reducible Complexity. *Science*, Volume 312, Number 5770, 7 April 2006. p. 61-63.

[Andrews, Gangestad, & Matthews, 2002] Andrews P.W., Gangestad S.W., and Matthews D., 2002. Adaptationism – How to Carry Out an Exaptationist Program. *Behavioral and Brain Sciences*, Volume 25, Number 4, Cambridge University Press. p. 489-553.

[Applegate et al., 2007] Applegate D.L., Bixby R.E., Chvátal V., and Cook W.J. 2007. *The Traveling Salesman Problem: A Computational Study*. Princeton University Press. ISBN 978-0-691-12993-8.

[Arnold, 1994] Arnold E.N., 1994. Investigating the origins of performance advantage: adaptation, exaptation and lineage effects. In *Phylogenetics and Ecology*, London: The Linnean Society of London.

[Arnold, 2002] Arnold E.N., 2002. Holaspis, a lizard that glided by accident: mosaics of cooption and adaptation in a tropical forest lacertid (Reptilia, Lacertidae). *Bulletin of the Natural History Museum, Zoology Series*, Volume 68, Issue 2, November 2002. p. 155-163.

[Ashlock & Joenks, 1998] Ashlock D., and Joenks M., 1998. ISAc Lists, a Different Representation for Program Induction. In Koza J.R., Banzhaf W., Chellapilla K., Deb K., Dorigo M., Fogel D.B., Garzon M.H., Goldberg D.E., Iba H., and Riolo R., eds. Proceedings of the Third Annual Conference, Genetic Programming 1998. Morgan Kaufmann. p. 3-10.

[Baldwin, 1896] Baldwin J.M., 1896. A new factor in evolution. American Naturalist, Volume 30. p. 441-451.

[Banzhaf, 2003] Banzhaf W., 2003. On the Dynamics of an Artificial Regulatory Network. In Advances in Artificial Life, 7th European Conference (ECAL'03), Lecture Notes in Artificial Intelligence, Volume 2801. Springer. p. 217-227.

[Banzhaf et al., 2006] Banzhaf W., Beslon G., Christensen S., Foster J.A., Képès F., Lefort V., Miller J.F., Radman M., and Ramsden J., 2006. Guidelines: From artificial evolution to computational evolution: a research agenda. Nature Reviews Genetics, Volume 7. p. 729-735.

[Barnum, Bernstein, & Spector, 2000] Barnum H., Bernstein H.J., and Spector L., 2000. Quantum circuits for OR and AND of ORs. Journal of Physics A: Mathematical and General, Volume 33. p. 8047-8057.

[Beer & Gallagher, 1992] Beer R., and Gallagher J., 1992. Evolving dynamical neural networks for adaptive behavior. Adaptive Behavior, Volume 1, Number 1. p. 91-122.

[Behe, 2006] Behe M.J., 2006. Darwin's Black Box. Free Press, New York, NY.

[Bochkanov & Bystritsky, 2008] Bochkanov S., and Bystritsky V., 2008. ALGLIB Project, Mann Whitney U Test. <<http://www.alglib.net>> (last accessed December 2008).

[Boxhorn, 1994] Boxhorn J., 1994. Observed Instances of Speciation. In The TalkOrigins Archive, <<http://www.talkorigins.org/faqs/faq-speciation.html>> (last accessed January 2008).

[Branke, 1999] Branke J., 1999. Memory enhanced evolutionary algorithms for changing optimization problems. In Proceedings of the 1999 IEEE Congress on Evolutionary Computation, CEC99, Volume 3. p. 1875-1882.

[Buss et al., 1998] Buss D.M., Haselton M.G., Shackelford T.K., Bleske A.L., and Wakefield J.C., 1998. Adaptations, Exaptations, and Spandrels. *American Psychologist*, Volume 53. p. 533-548.

[Cage, Kroo, & Braun, 1995] Cage P.G., Kroo I.M., and Braun R.D., 1995. Interplanetary trajectory optimization using a genetic algorithm. *Journal of Astronautical Science*, Volume 43. p. 59-75.

[Cariani, 2002] Cariani P.A., 2002. Extradimensional bypass. *BioSystems*, Volume 64, Number 1, Elsevier, January 2002. p. 47-53.

[Chaumont, Egli, & Adami, 2007] Chaumont N., Egli R., and Adami C., 2007. Evolving Virtual Creatures and Catapults. *Artificial Life*, Volume 13. p. 139-157.

[Cliff, Harvey, & Husbands, 1993] Cliff D., Harvey I., and Husbands P., 1993.

Explorations in Evolutionary Robotics. *Adaptive Behavior*, Volume 2, Number 1. p. 71-104.

[CMLabs, 2007] Vortex, 2007. <<http://www.cm-labs.com>> (last accessed December 2007).

[Conrad, 1990] Conrad M., 1990. The Geometry of Evolution. *BioSystems*, Volume 24. p. 61-81.

[Darwin, 1859] Darwin C., 1859. *On the Origin of Species*. Murray J.: London.

[Dasgupta, 1999] Dasgupta D., 1999. *Artificial Immune Systems and Their Applications*. Springer-Verlag, Inc. Berlin.

[Dawkins, 1986] Dawkins R., 1986. *The Blind Watchmaker*. Harlow Longman.

[de Garis, 1991] de Garis H., 1991. *Genetic Programming: GenNets, Artificial Nervous Systems, Artificial Embryos*. PhD thesis, Université Libre de Bruxelles, Belgium.

[de Jong, 2008] de Jong G., 2008. *Darwin at Home*. <<http://www.darwinathome.org/>> (last accessed January 2008).

[de Oliveira, 1994] de Oliveira P.P.B., 1994. Simulation of exaptive behavior. In *Parallel Problem Solving from Nature*, 3, Lecture Notes in Computer Science 866, Davidor Y.,

Schwefel H.-P., and Maenner R., eds., Springer-Verlag, Berlin, Germany, October 1994. p. 354-364.

[Deb & Goldberg, 1989] Deb K., and Goldberg D., 1989. An investigation of niche and species formation in genetic function optimization. In Proceedings of the 3rd International Conference on Genetic Algorithms, Morgan Kaufmann, San Francisco. p. 42-50.

[Deb et al., 2002] Deb K., Pratap A., Agarwal S., and Meyarivan T., 2002. A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II. In IEEE Transactions on Evolutionary Computation, Volume 6, Number 2. p. 182–197.

[Delcomyn, 1985] Delcomyn F., 1985. Factors regulating insect walking. Annual Review of Entomology, Volume 30. p. 239-256.

[Dennett, 1995] Dennett D.C., 1995. Darwin's Dangerous Idea. Simon & Schuster, New York.

[Dickerson & Geis, 1969] Dickerson R.E., and Geis I., 1969. The Structure and Action of Proteins. Harper and Row, New York.

[Doolittle & Sapienza, 1980] Doolittle W.F., and Sapienza C., 1980. Selfish genes, the phenotype paradigm, and genome evolution. Nature, Volume 284. p. 601-603.

[Fastovsky & Weishampel, 1996] Fastovsky D.E., and Weishampel D.B., 1996. The Evolution and Extinction of the Dinosaurs. Cambridge University Press.

[Fentress, 2005] Fentress S.A., 2005. Exaptation as a Means of Evolving Complex Solutions. Master's Thesis, University of Edinburgh.

[Ferreira, 2001] Ferreira C., 2001. Gene Expression Programming: A New Adaptive Algorithm for Solving Problems. *Complex Systems*, Volume 13, Issue 2. p. 87-129.

[Ficici & Pollack, 1998] Ficici S., and Pollack J.B., 1998. Challenges in Coevolutionary Learning: Arms-Race Dynamics, Open-Endedness, and Mediocre Stable States. In *Proceedings of the 6th International Conference on Artificial Life*, Adami C., Belew R.K., Kitano H., and Taylor C., eds., The MIT Press, Cambridge, MA. p. 238-247.

[Fogel, Owens, & Walsh, 1966] Fogel L.J., Owens A.J., and Walsh M.J., 1966. *Artificial Intelligence through Simulated Evolution*. Wiley, New York.

[Fonseca & Fleming, 1995] Fonseca C.M., Fleming P.J., 1995. Multiobjective genetic algorithms made easy: selection, sharing, and mating restriction. In *Proceedings of the 1st International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications*, Sheffield, U.K., The Institution of Electrical Engineers. p. 45-52.

[Franks & Noble, 2002] Franks D.W., and Noble J., 2002. The Origins of Mimicry Rings. In *Proceedings of the 8th International Conference on Artificial Life*, Standish, Abbass, and Bedau, eds., The MIT Press. p. 186-191.

[Frazzetta, 1975] Frazzetta T.H., 1975. *Complex Adaptations in Evolving Populations*. Sinauer Associates, Sunderland, MA.

[Gauthier & Gall, 2001] Gauthier J., and Gall L.F., eds., 2001. *New Perspectives on the Origin and Early Evolution of Birds*. Peabody Museum, New Haven, Connecticut.

[Gomi & Griffith, 1996] Gomi T., and Griffith A., 1996. Evolutionary Robotics – An overview. In *Proceedings of the IEEE Third International Conference on Evolutionary Computation*, IEEE Press.

[Gould & Lewontin, 1979] Gould S.J., and Lewontin R.C., 1979. The Spandrels of San Marco and the Panglossian Paradigm: A Critique of the Adaptationist Programme. In *Proceedings of the Royal Society of London, Series B, Biological Sciences, Volume 205, Number 1161, The Evolution of Adaptation by Natural Selection*. p. 581-598.

[Gould, 1980] Gould S.J., 1980. *The Panda's Thumb*. W.W. Norton & Company.

[Gould & Vrba, 1982] Gould S.J., and Vrba E., 1982. Exaptation: A missing term in the science of form. *Paleobiology*, Volume 8, Number 1. p. 4-15.

[Gould, 1997] Gould S.J., 1997. The exaptive excellence of spandrels as a term and prototype. In *Proceedings of the National Academy of Sciences of the United States of America*, Volume 94, Number 20. p. 10750-10755.

[Gould, 2002] Gould S. J., 2002. *The Structure of Evolutionary Theory*. The Belknap Press of Harvard University Press, Cambridge, Massachusetts and London, England.

[Graham, 2008a] Graham L., 2008. 3D Virtual Creature Evolution.

<http://www.stellaralchemy.com/lee/virtual_creatures.html> (last accessed February 2008).

[Graham, 2008b] Graham L., 2008. Animation of an evolved 2D navigator.

<<http://www.youtube.com/watch?v=GrnHnGwmZ7A>>

[Graham & Oppacher, 2007a] Graham L., and Oppacher F., 2007. Speciation through Exaptation. In Proceedings of the 2007 IEEE Symposium on Artificial Life, ALIFE07, Hawaii, April 2007. p. 433-439.

[Graham & Oppacher, 2007b] Graham L., and Oppacher F., 2007. Speciation through Selection and Drift. In Proceedings of the 11th IASTED International Conference on Artificial Intelligence and Soft Computing, del Pobil A.P., ed., ACTA Press, August 2007. p. 213-218.

[Graham & Oppacher, 2007c] Graham L., and Oppacher F., 2007. A Multiple-Function Toy Model of Exaptation in a Genetic Algorithm. In Proceedings of the 2007 IEEE Congress on Evolutionary Computation (CEC 2007), September 2007. p. 4591-4598.

[Graham, Oppacher, & Christensen, 2007] Graham L., Oppacher F., and Christensen S., 2007. Irreducible Complexity in a Genetic Algorithm. In Proceedings of the 2007 IEEE Congress on Evolutionary Computation (CEC 2007), September 2007. p. 3692-3697.

[Gruau, 1994] Gruau F., 1994. Automatic definition of modular neural networks. Adaptive Behavior, Volume 3, Number 2. p. 151-184.

[Gruau & Quatramaran, 1996] Gruau F., and Quatramaran K., 1996. Cellular encoding for interactive evolutionary robotics. Technical report, University of Sussex, School of Cognitive Sciences, EASY Group, Brighton, UK.

[Halstead, 1969] Halstead L.B., 1969. The Pattern of Vertebrate Evolution. Oliver and Boyd, Edinburgh.

[Harvey, Husbands, & Cliff, 1994] Harvey I., Husbands P., and Cliff D., 1994. Seeing the light: Artificial evolution, real vision. In From Animals to Animats 3, Proceedings of the Third International Conference on Simulation of Adaptive Behavior, Cliff D., Husbands P., Meyer J.-A., and Wilson S.W., eds., The MIT Press / Bradford Books, Cambridge, MA. p. 392-401.

[Hillis, 1990] Hillis D., 1990. Co-evolving Parasites Improve Simulated Evolution as an Optimization Procedure. Physica D: Nonlinear Phenomena, Volume 42, Issue 1-3, Elsevier, June 1990. p. 228-234.

[Hinton & Nowlan, 1987] Hinton G.E., and Nowlan S.J., 1987. How Learning Can Guide Evolution. Complex Systems 1. p. 495-502.

[Holland, 1993] Holland J.H., 1993. Adaptation in Natural and Artificial Systems. 2nd Edition, MIT Press, MA.

[Holland & Reitman, 1978] Holland J.H., and Reitman J.H., 1978. Cognitive Systems Based in Adaptive Algorithms. In Pattern-directed Inference Systems, Waterman & Hayes-Roth, editors. Academic Press.

[Husbands & Meyer, 1998] Husbands P., and Meyer J.-A., 1998. Evolutionary Robotics. Springer, Berlin.

[Isaak, 2005] Isaak M., 2005. Claim CB200. In The TalkOrigins Archive.
<<http://www.talkorigins.org/indexcc/CB/CB200.html>> (Last accessed December 2007).

[Jones III, 2005] Jones III J.E., 2005. Memorandum Opinion. Kitzmiller et al. v Dover Area School District et al., 400 F.Supp.2d 707 (M.D. Pa. 2005), Case No. 04cv2688, December 20, 2005.

[Kardong, 2002] Kardong K.V., 2002. Vertebrates: Comparative Anatomy, Function, Evolution. Third edition, McGraw Hill, New York.

[Kimura, 1983] Kimura M., 1983. The Neutral Theory of Molecular Evolution. Cambridge University Press.

[Kingsolver & Koehl, 1994] Kingsolver J.G., and Koehl M.A.R., 1994. Selective Factors in the Evolution of Insect Wings. Annual Review of Entomology, Volume 39. p. 425-451.

[Klein, 2003] Klein J., 2003. BREVE: A 3D environment for the simulation of decentralized systems and artificial life. In Proceedings of the 8th International

Conference on Artificial Life, Standish K., Bedau M.A., and Abbass A., eds., MIT Press, Cambridge, MA. p. 329-334.

[**Klein, 2007**] Klein J., BreveCreatures,
<<http://www.spiderland.org/breve/breveCreatures.html>> (last accessed December 2007).

[**Knabe, Nehaniv, & Schilstra, 2006**] Knabe J.F., Nehaniv C.L. and Schilstra M.J.,
2006. Evolutionary Robustness of Differentiation in Genetic Regulatory Networks. In
Proceedings of the 7th German Workshop on Artificial Life, (GWAL-7), Akademische
Verlagsgesellschaft Aka, Berline. p. 75-84.

[**Kodjabachian & Meyer, 1998a**] Kodjabachian J., and Meyer J.-A., 1998. Evolution
and Development of Modular Control Architectures for 1-D Locomotion in Six-Legged
Animats. Connection Science, Volume 10. p. 211-254.

[**Kodjabachian & Meyer, 1998b**] Kodjabachian J., and Meyer J.-A., 1998. Evolution
and Development of Neural Controllers for Locomotion, Gradient-Following, and
Obstacle-Avoidance in Artificial Insects. IEEE Transactions on Neural Networks,
Volume 9, Issue 5, September 1998. p. 796-812.

[**Komosinski & Ulatowski, 1999**] Komosinski M., and Ulatowski S., 1999. Framsticks:
Towards a Simulation of a Nature-Like World, Creatures and Evolution. In Proceedings
of the 5th European Conference on Artificial Life, ECAL'99, Floreano D. et al., eds.,
Springer, Berlin. p. 261-265.

[Komosinski, 2000] Komosinski M., 2000. The world of framesticks: simulation, evolution, interaction. In Proceedings of the 2nd International Conference on Virtual Worlds (VW2000), Paris, Springer-Verlag (LNAI 1834). p. 214-224.

[Koza, 1992] Koza J.R., 1992. Genetic Programming. MIT Press, MA.

[Koza et al., 2000] Koza J.R., Keane M.A., Yu J., Bennett F.H. III, and Mydlowec W., 2000. Automatic creation of human-competitive programs and controllers by means of genetic programming. Genetic Programming and Evolvable Machines, Volume 1. p. 121-164.

[Koza et al., 2003] Koza J.R., Keane M.A., Streeter M.J., Mydlowec W., Yu J., and Lanza G., 2003. Genetic Programming IV: Routine Human-Competitive Machine Intelligence. Kluwer Academic, Massachusetts.

[Krčah, 2007] Krčah P., 2007. Evolving Virtual Creatures Revisited. In Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'07), London, England, UK. July 7-11, 2007.

[Larrañga & Lozano, 2002] Larrañga P., and Lozano J.A., 2002. Estimation of distribution algorithms: A new tool for evolutionary computation. Kluwer Academic Publishers, Boston.

[Lassabe, Luga, & Duthen, 2007] Lassabe N., Luga H., Duthen Y., 2007. A New Step for Artificial Creatures. In Proceedings of Artificial Life, IEEE-ALife'07. p. 243-251.

[Lembcke, 2007] Lembcke S., 2007. Chipmunk Game Dynamics.

<<http://wiki.slembcke.net/main/published/Chipmunk>> (last accessed July 2008).

[Lenski et al., 2003] Lenski R.E., Ofria C., Pennock R.T., and Adami C., 2003. The evolutionary origin of complex features. *Nature*, Volume 423. p. 139-144.

[Lewis, Fagg, & Solidum, 1992] Lewis M.A., Fagg A.H., and Solidum A., 1992.

Genetic programming approach to the construction of a neural network for control of a walking robot. In IEEE International Conference on Robotics and Automation, Nice, France. p. 2618-2623.

[Liem, 1988] Liem K.F., 1988. Form and function of lungs: the evolution of airbreathing mechanisms. *American Zoologist*, Volume 28, Number 2. p. 739-759.

[Lindberg & Dobbertein, 1981] Lindberg D.R., and Dobbertein R.A., 1981. Umbilical brood protection and sexual dimorphism in the boreal trochid gastropod *Margarites vorticiferus*. Dall. *International Journal of Invertebrate Reproduction*, Volume 3. p. 347-355.

[Lindenmayer, 1968] Lindenmayer A., 1968. Mathematical Models for Cellular Interactions in Development, Parts I and II. *Journal of Theoretical Biology*, Volume 18. p. 280-315.

[Lipson & Pollack, 2000] Lipson H., and Pollack J., 2000. Automatic design and manufacture of robotic lifeforms. *Nature*, Volume 406, Number 6799, (The GOLEM Project). p. 974-978.

[Lohn et al., 2004] Lohn J.D., Linden D.S., Hornby G.S., Kraus W.F., Rodriguez A., and Seufert S., 2004. Evolutionary Design of an X-Band Antenna for NASA's Space Technology 5 Mission. In Proceedings of the 2004 IEEE Antenna and Propagation Society International Symposium and USNC/URSI National Radio Science Meeting, Volume 3. p. 2313-2316.

[Luke et al., 2008] Luke S., Panait L., Balan G., Paus S., Skolicki Z., Popovici E., Sullivan K., Harrison J., Bassett J., Hubble R., and Chircop A., 2008. ECJ : A Java-based Evolutionary Computation Research System. Version 18.
<<http://cs.gmu.edu/~eclab/projects/ecj/>> (last accessed July 2008).

[Lund & Parisi, 1995] Lund H.H., and Parisi D., 1995. Pre-adaptation in Populations of Neural Networks Evolving in a Changing Environment. In *Artificial Life 2:2*, p. 179-197.

[Margulis, 1981] Margulis L., 1981. *Symbiosis in Cell Evolution*, 1st Edition. Freeman, New York.

[McLachlan & Liversidge, 1978] McLachlan G.R., and Liversidge R., 1978. *Roberts' Birds of South Africa*, 4th Edition (first published in 1940), John Voelcker Bird Book Fund, Cape Town.

[Miconi & Channon, 2005] Miconi T., and Channon A.D., 2005. A virtual creatures model for studies in artificial evolution. In Proceedings of the 2005 IEEE Congress on Evolutionary Computation (CEC 2005), Volume 1, Edinburgh, Corne D. et al., eds., IEEE Press. p. 565–572.

[Miconi & Channon, 2006a] Miconi T., and Channon A.D., 2006. An improved system for artificial creatures evolution. In Proceedings of the 10th Conference on the Simulation and Synthesis of Living Systems, ALIFE X, Rocha L., Yaeger L., Bedau M., Floreano D., Goldstone R., and Vespignani A., eds., MIT Press, Cambridge, MA. p. 255-261.

[Miconi & Channon, 2006b] Miconi T., and Channon A.D., 2006. Analysing co-evolution among artificial 3D creatures. In Proceedings of the 7th International Conference on Artificial Evolution (Evolution Artificielle 2005): Revised Selected Papers, Talbi E.G. et al., eds., Springer, Lille. p. 167–178.

[Miller, 1999] Miller K.R., 1999. Finding Darwin's God. Harper Collins, New York, NY.

[Muller, 1939] Muller H.J., 1939. Reversibility in evolution considered from the standpoint of genetics. Biological Reviews of the Cambridge Philosophical Society, Volume 14. p. 261-280.

[Nietzsche, 2008] Nietzsche F., 2008. On the Genealogy of Morals. (Original publication in 1887), translated by Johnston I.,
<<http://www.mala.bc.ca/~johnstoi/Nietzsche/genealogytofc.htm>> (last accessed January 2008).

[OGRE, 2007] OGRE: Object-Oriented Graphics Rendering Engine.
<<http://www.ogre3d.org>> (last accessed December 2007).

[Oppacher & Wineberg, 1999] Oppacher F., and Wineberg M., 1999. The Shifting Balance Genetic Algorithm: Improving the GA in a Dynamic Environment. In Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-99), Banzhaf W. et al., eds., San Francisco: Morgan Kaufmann. p. 504-510.

[Oppacher & Wineberg, 2000] Oppacher F., and Wineberg M., 2000. Reconstructing the Shifting Balance Theory in a GA: Taking Sewall Wright Seriously. In Proceedings of the 2000 Congress on Evolutionary Computation (CEC2000), Zalzal A., ed., IEEE Press. p. 219-226.

[Orgel & Crick, 1980] Orgel L.E., and Crick F.H.C., 1980. Selfish DNA: the ultimate parasite. Nature, Volume 284. p. 604-607.

[Ostrom, 1976] Ostrom J.H., 1976. Archaeopteryx and the origin of birds. Biological Journal of the Linnean Society, Volume 8. p. 91-182.

[Piatigorsky & Wistow, 1991] Piatigorsky J., and Wistow G.J., 1991. The recruitment of crystallins: new functions precede gene duplication. Science, Volume 252, Number 5009, May 1991. p. 1078-1079.

[Piatigorsky, 1993] Piatigorsky J., 1993. Puzzle of crystalline diversity in eye lenses. Developmental Dynamics, Volume 196, Issue 4. p. 267-272.

[Piatigorsky, 2003] Piatigorsky J., 2003. Gene Sharing, Lens Crystallins and Speculations on an Eye/Ear Evolutionary Relationship. Integrative and Comparative

Biology, Volume 43, Number 4, The Society for Integrative and Comparative Biology. p. 492-499.

[Rechenberg, 1993] Rechenberg I., 1993. *Evolutionsstrategie*, 2nd Edition. Frommann, Stuttgart.

[Robbins, 1994] Robbins P., 1994. The Effect of Parasitism on the Evolution of a Communication Protocol in an Artificial Life Simulation. In *From Animals to Animats 3, Proceedings of the 3rd International Conference on the Simulation of Adaptive Behavior*, Cliff D., Husbands P., Meyer J.-A., and Wilson S.W., eds., The MIT Press / Bradford Books, Cambridge, MA. p. 431-437.

[Rosin & Belew, 1995] Rosin C.D., and Belew R.K., 1995. Methods for Competitive Co-evolution: Finding Opponents Worth Beating. In *Proceedings of the 6th International Conference on Genetic Algorithms (ICGA 95)*, Eshelman L.J., ed., Morgan Kaufmann, San Francisco, CA. p. 373-380.

[Sakarya et al., 2007] Sakarya O., Armstrong K.A., Adamska M., Adamski M., Wang I.F. et al., 2007. A Post-Synaptic Scaffold at the Origin of the Animal Kingdom. *PLoS ONE* 2(6): e506 doi:10.1371/journal.pone.0000506

[Salesa, 2006] Salesa M.J., Antón M., Peigné S., and Morales J., 2006. Evidence of a false thumb in a fossil carnivore clarifies the evolution of pandas. *Proceedings of the National Academy of Sciences of the United States of America*, Volume 103, Number 2. p. 379–382.

[Schuster et al., 1994] Schuster P., Fontana W., Stadler P.F., and Hofacker I.L., 1994. From sequences to shapes and back: A case study in RNA secondary structures. In Proceedings of the Royal Society of London, B, Volume 255. p. 279-284.

[Shim & Kim, 2003] Shim Y.-S., and Kim C.-H., 2003. Generating flying creatures using body-brain coevolution. In Proceedings of ACM SIGGRAPH / Eurographics Symposium on Computer Animation, Rockwood A.P., ed., Eurographics Association. p. 267-285.

[Shim, Kim, & Kim, 2004] Shim Y.-S., Kim S.-J., and Kim C.-H., 2004. Evolving flying creatures with path following behavior. In Proceedings of Artificial Life IX, Pollack J. et al., eds., MIT Press, Cambridge, MA. p. 125-132.

[Sims, 1991] Sims K., 1991. Artificial Evolution for Computer Graphics. Computer Graphics, Volume 25, Number 4, July 1991. p. 319-328.

[Sims, 1992] Sims K., 1992. Interactive Evolution of Dynamical Systems. Toward a Practice of Autonomous Systems: Proceedings of the 1st European Conference on Artificial Life, Varela, Francisco, and Bourguine, eds., MIT Press. p. 171-178.

[Sims, 1994a] Sims K., 1994. Evolving Virtual Creatures. Computer Graphics, Annual Conference Series, (SIGGRAPH '94 Proceedings), July 1994. p. 15-22.

[Sims, 1994b] Sims K., 1994. Evolving 3D morphology and behavior by competition. In Proceedings of the 4th International Workshop on Artificial Life, Brooks R.A., and Maes P., eds., The MIT Press / Bradford Books, Cambridge, MA.

[Smith, 2007] Smith R., 2007. Open Dynamics Engine (ODE). <<http://www.ode.org>> (last accessed December 2007).

[Stephens, 1999] Stephens C., 1999. Effective fitness and the non-triviality of the genotype-phenotype map. In EEBC meetings. Emergent and Evolutionary Behavior, Intelligence and Computation group.

[Stoica et al., 1999] Stoica A., Klimeck G., Salazar-Lazaro C., Keymeulen D., and Thakoor A., 1999. Evolutionary design of electronic devices and circuits. In Proceedings of the 1999 Congress on Evolutionary Computation, IEEE Press, New York. p. 1271-1278.

[Stuart et al., 2004] Stuart A.J., Kosintsev P.A., Higham T.F.G., and Lister A.M., 2004. Pleistocene to Holocene extinction dynamics in giant deer and woolly mammoth. *Nature*, Volume 431, Number 7009. p. 684-689.

[TalkOrigins, 2007] Various authors. 2007. The TalkOrigins Archive <<http://www.talkorigins.org>> (Last accessed December 2007).

[Taylor & Raes, 2005] Taylor J.S., and Raes J., 2005. The Evolution of the Genome, chapter, Small-Scale Gene Duplications, Elsevier Academic Press.

[Taylor & Massey, 2001] Taylor T., and Massey C., 2001. Recent developments in the evolution of morphologies and controllers for physically simulated creatures. *Artificial Life*, Volume 7, Number 1. p. 77-87.

[van Valen, 1973] van Valen L. 1973. A New Evolutionary Law. *Evolutionary Theory*, Volume 1. p. 1-30.

[Vorbach, Capecchi, & Penninger, 2006] Vorbach C., Capecchi M.R., Penninger J.M., 2006. Evolution of the mammary gland from the innate immune system? *BioEssays*, Volume 28. p. 606-616.

[Watson & Pollack, 2000] Watson R.A., and Pollack J.B., 2000. Symbiotic Combination as an Alternative to Sexual Recombination in Genetic Algorithms. In *Proceedings of Parallel Problem Solving from Nature VI (PPSN VI)*, Shoenauer et al., eds., Springer. p. 425-434.

[Watson & Pollack, 2001] Watson R.A., and Pollack J.B., 2001. Symbiotic Composition and Evolvability. In *Proceedings of the European Conference on Artificial Life (ECAL 2001)*. p. 480-490.

[Wineberg & Oppacher, 2000] Wineberg M., and Oppacher F., 2000. Enhancing the GA's Ability to Cope with Dynamic Environments. In *Proceedings of the 2nd Genetic and Evolutionary Computation Conference (GECCO-2000)*, Whitley D., ed., Morgan Kaufmann. p. 3-10.